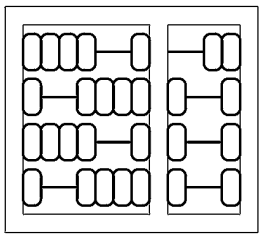
Universidade Estadual de Campinas
Instituto de Computação



Proposta de Tese de Doutorado

O Problema da Ordenação de Permutações Usando Operações de Prefixo e Sufixo

Aluna:
Carla Negri Lintzmayer

Orientador:
Prof. Dr. Zanoni Dias

Campinas - São Paulo
2014

Resumo

Reversões e transposições são os tipos mais comuns de rearranjos de genomas, que são operações que afetam grandes porções dos genomas. Assume-se que a distância mínima entre dois determinados genomas é a distância evolutiva entre eles, e ela pode ser obtida por meio de cenários que envolvam os rearranjos. Assim, um problema de transformar genomas utilizando um ou mais tipos de rearranjos é o problema de encontrar a distância mínima entre dois genomas utilizando os rearranjos permitidos. Se representarmos os genomas por meio de permutações e assumirmos que uma delas é a identidade, o problema se resume em encontrar a quantidade mínima de rearranjos que ordenam a outra. Por esse motivo, os problemas são chamados de ordenação. Quando as operações de rearranjo afetam segmentos do início ou do fim do genoma, dizemos que são rearranjos de prefixo ou de sufixo, respectivamente. Esta proposta apresenta alguns conceitos iniciais e resultados já existentes sobre alguns problemas relacionados, bem como especifica o objetivo do nosso trabalho, que visa lidar com problemas de ordenação usando reversões e transposições nas suas versões de prefixo e sufixo, e apresenta os resultados parciais já obtidos.

1 Introdução

O Problema da Ordenação de Panquecas foi introduzido por Dweighter [20] em 1975 e pode ser descrito de forma simples como sendo o problema de organizar uma pilha de panquecas de tamanhos variados de forma que a menor fique no topo da pilha, a segunda menor fique logo abaixo dela, e assim sucessivamente até que a maior fique na parte inferior. Além disso, essa organização só pode ser feita por inversões de blocos de panquecas que estão no topo da pilha.

Tal problema também é chamado de Problema de Ordenação por Reversões de Prefixo, devido à natureza dos movimentos permitidos para a ordenação da pilha de panquecas. Em 1995, Pevzner e Waterman [37] o reinterpretaram como sendo um problema de Rearranjo de Genomas, pois reversão é um rearranjo e reversão de prefixo é uma reversão restrita. Outros exemplos de rearranjos são transposição, inserção, remoção, duplicação, fissão e fusão [24].

Em 1979, o Problema da Ordenação de Panquecas foi estudado por Gates e Papadimitriou [28] e a preocupação inicial estava em encontrar o número máximo de movimentos que seriam necessários para ordenar qualquer pilha de panquecas [24]. Porém, existem trabalhos na literatura que também se preocupam em encontrar o número mínimo de movimentos suficientes para ordenar uma pilha dada [19, 25, 32]. Mais de trinta anos depois de ser proposto, em 2011 o problema foi provado ser NP-difícil [10].

Sharmin *et al.* [40] recentemente caracterizaram duas novas versões para o problema das panquecas: uma podendo utilizar reversões de prefixo e/ou transposições de prefixo e outra podendo utilizar reversões de prefixo, transposições de prefixo e/ou transreversões de prefixo (uma combinação dos outros dois rearranjos).

Assim como existem operações restritas ao prefixo dos genomas, também é possível considerar restringi-las ao sufixo. Entretanto, note que se algum problema envolve exclusivamente rearranjos de prefixo, não há necessidade de trabalhar em uma versão que utilize os mesmos rearranjos exclusivamente na versão de sufixo, pois ambos problemas seriam equivalentes. Sendo assim, é viável considerar problemas de ordenação cujos rearranjos permitidos envolvam tanto versões de prefixo quanto de sufixo. É importante notar que tal consideração ainda não foi feita na literatura. Desta forma, este trabalho de doutorado tem como objetivo principal estudar os problemas de ordenação de genomas que utilizam rearranjos de reversão e transposição nas suas versões de prefixo e sufixo.

O restante desta proposta está organizado da seguinte forma: a Seção 2 apresenta alguns conceitos teóricos relacionados a este trabalho e alguns problemas relacionados; a Seção 3 descreve os objetivos do trabalho a ser realizado; a Seção 4 apresenta o plano de trabalho e o cronograma a ser seguido para sua realização; a Seção 5 apresenta a metodologia a ser utilizada para cumprir os objetivos; e por fim, a Seção 6 apresenta os resultados parciais já obtidos.

2 Rearranjos de Genomas

Um rearranjo é um tipo de mutação que afeta grandes porções dos genomas, diferente das mutações pontuais, que afetam suas moléculas constituintes. Mutações permitem evoluções, e, segundo Fertin *et al.* [24], Dobzhansky e Sturtevant propuseram, em 1936, utilizar o grau de desordem entre genes de dois genomas diferentes como um indicador da distância de evolução entre eles, considerando cenários que mostrem os rearranjos necessários para transformar um genoma em outro. Como os rearranjos são eventos relativamente raros e pelo princípio da parcimônia, assume-se que a distância de evolução entre dois indivíduos é dada pela sequência mínima de tais rearranjos. Encontrar essa sequência mínima é o objetivo dos Problemas de Rearranjos de Genomas.

Permutações são modelos matemáticos utilizados para formalizar o estudo de arranjos de fragmentos de DNA [24]. Elas representam os cromossomos como sequências de segmentos que são compartilhados pelos genomas que estão sendo comparados. Uma permutação **sem sinal** é uma função bijetora sobre o conjunto $Z_n = \{1, 2, \ldots, n\}$ e é normalmente representada por $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, onde $\pi_i = \pi(i)$ é a imagem de $i \in Z_n$. O **grupo simétrico**, denotado por S_n, é o conjunto de todas as $n!$ permutações sobre Z_n.

Uma permutação **com sinal** deve satisfazer $\pi_{-i} = -\pi_i$ para todo $i \in \{1, 2, \ldots, n\}$. Por isso, pode-se ocultar o mapeamento dos elementos negativos e ela é simplesmente uma permutação $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ sobre o conjunto $Z_n^{\pm} = \{-n, -(n-1), \ldots, -1, 1, 2, \ldots, n\}$. Em outras palavras, cada elemento da permutação pode ter associado a ele um sinal + (muitas vezes omitido) ou − e, em módulo, nenhum elemento é repetido. O **grupo hiperoctaedral**, denotado por $S_n^{\pm}$, é o conjunto de todas as $n!2^n$ permutações sobre $Z_n^{\pm}$. No caso dos genomas, uma

permutação com sinal é um modelo mais real do que uma sem sinal, pois ela consegue representar a orientação dos genes, que é determinada pela sua localização na fita dupla de DNA.

Uma **composição** de permutações é como uma composição de funções [24]. Dadas duas permutações π e σ, escrever $\pi \cdot \sigma$ significa aplicar σ e depois aplicar π, de forma que $\pi \cdot \sigma = (\pi_{\sigma_1}\ \pi_{\sigma_2}\ \ldots\ \pi_{\sigma_n})$. A **permutação inversa** de π é π^{-1}, na qual $\pi^{-1}_{\pi_i} = i$ para $1 \leq i \leq n$. Ela satisfaz $\pi^{-1} \cdot \pi = \iota_n$, onde ι_n é a **permutação identidade**, que é definida como $\iota_n = (1\ 2\ \ldots\ n)$. Outra permutação especial é a **permutação reversa**. Para permutações sem sinal ela é definida por $\eta_n = (n\ n-1\ \ldots\ 1)$ enquanto que para permutações com sinal ela é definida por $\bar{\eta_n} = (-n\ -(n-1)\ \ldots\ -1)$. A permutação **identidade com sinal** $\bar{\iota_n} = (-1\ -2\ \ldots\ -n)$ também é outra permutação especial que consideraremos.

Permutações podem representar também mutações (rearranjos) de tal forma que uma certa mutação λ transforma um genoma π em outro genoma $\pi \cdot \lambda$.

A **permutação estendida**, que também é denotada por π, tem dois elementos fixos $\pi_0 = 0$ e $\pi_{n+1} = n+1$, de tal forma que $\pi = (\pi_0\ \pi_1\ \ldots\ \pi_n\ \pi_{n+1})$. Para permutações com sinal, a convenção é que os elementos π_0 e π_{n+1} tenham sempre sinais positivos. De agora em diante, estaremos sempre nos referindo à versão estendida de qualquer permutação, mesmo se os elementos extras forem omitidos.

Dentre os rearranjos ou **operações** existentes, reversão de prefixo, reversão de sufixo, transposição de prefixo e transposição de sufixo são os mais importantes para este trabalho. A seguir apresentamos as definições de reversão e transposição para então definir os outros quatro.

Definição 1. *Uma **reversão sem sinal** $\rho(i,j)$ com $1 \leq i < j \leq n$, é a permutação da forma* $(1\ \ldots\ i-1\ \underline{j\ \ j-1\ \ \ldots\ \ i+1\ \ i}\ j+1\ \ldots\ n)$. *Logo,* $\pi \cdot \rho(i,j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \ldots\ \pi_{i+1}\ \pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$, *ou seja, ocorre uma inversão do segmento* $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j$ *de* π.

Definição 2. *Uma **reversão com sinal** $\bar{\rho}(i,j)$ com $1 \leq i \leq j \leq n$, é a permutação da forma* $(1\ \ldots\ i-1\ \underline{-j\ \ -(j-1)\ \ \ldots\ \ -(i+1)\ \ -i}\ j+1\ \ldots\ n)$. *Logo,* $\pi \cdot \bar{\rho}(i,j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{-\pi_j\ -\pi_{j-1}\ \ldots\ -\pi_{i+1}\ -\pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$, *ou seja, ocorre tanto uma inversão do segmento* $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j$ *de* π *quanto uma inversão dos sinais dos elementos de tal segmento.*

Definição 3. *Uma **transposição** $\tau(i,j,k)$ com $1 \leq i < j < k \leq n+1$, é a permutação da forma* $(1\ \ldots\ i-1\ \underline{j\ \ j+1\ \ \ldots\ \ k-1}\ \underline{i\ \ i+1\ \ \ldots\ \ j-1}\ k\ \ldots\ n)$. *Logo,* $\pi \cdot \tau(i,j,k) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j+1}\ \ldots\ \pi_{k-1}}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}}\ \pi_k\ \ldots\ \pi_n)$, *ou seja, ocorre uma troca de posições do segmento* $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}$ *com seu segmento consecutivo* $\pi_j, \pi_{j+1}, \ldots, \pi_{k-1}$.

Definição 4. *Uma **reversão de prefixo sem sinal** $\rho_p(j)$ é uma reversão sem sinal $\rho(1,j)$ para $1 < j \leq n$ (apenas segmentos que contêm o primeiro elemento são invertidos).*

Definição 5. *Uma **reversão de prefixo com sinal** $\bar{\rho_p}(j)$ é uma reversão com sinal $\bar{\rho}(1,j)$ para $1 \leq j \leq n$ (apenas segmentos que contêm o primeiro elemento são invertidos).*

Definição 6. *Uma **reversão de sufixo sem sinal** $\rho_s(i)$ é uma reversão sem sinal $\rho(i,n)$ para $1 \leq i < n$ (apenas segmentos que contêm o último elemento são invertidos).*

Definição 7. *Uma **reversão de sufixo com sinal** $\bar{\rho}_s(i)$ é uma reversão com sinal $\bar{\rho}(i,n)$ para $1 \leq i \leq n$ (apenas segmentos que contêm o último elemento são invertidos).*

Definição 8. *Uma **transposição de prefixo** $\tau_p(j,k)$ é uma transposição $\tau(1,j,k)$ para $1 < j < k \leq n+1$ (um segmento que contém o primeiro elemento deve participar da operação).*

Definição 9. *Uma **transposição de sufixo** $\tau_s(i,j)$ é uma transposição $\tau(i,j,n+1)$ para $1 \leq i < j < n+1$ (um segmento que contém o último elemento deve participar da operação).*

Note que não existe versão com sinal para transposições, já que essa operação apenas troca os elementos de posição. Além disso, a partir daqui, deixaremos claro apenas quando estivermos tratando de operações ou permutações com sinal (os termos "sem sinal" serão omitidos).

Para atingir o objetivo dos problemas de Rearranjos de Genomas precisamos computar uma sequência de rearranjos que transforme uma permutação π em outra σ. Para isso, precisamos definir um **modelo de rearranjo** β, que nada mais é do que o rearranjo ou conjunto de rearranjos permitidos para realizar tal transformação. Em outras palavras, precisamos encontrar uma sequência $\beta_1, \ldots, \beta_k$ tal que $\pi \cdot \beta_1 \cdot \ldots \cdot \beta_k = \sigma$, onde β_i faz parte do modelo β (por exemplo, no problema de ordenação por reversões, β_i pode ser apenas ρ e no problema de ordenação por reversões de prefixo e transposições de prefixo, β_i pode ser ρ_p ou τ_p).

Definição 10. *Seja β um modelo de rearranjo e π e σ duas permutações quaisquer. O menor número de operações pertencentes a β necessárias para transformar π em σ é definido como sendo a **distância de rearranjo** entre π e σ com respeito a β e é denotado por $d_\beta(\pi,\sigma)$.*

Os rearranjos $\beta_1, \ldots, \beta_k$ formam uma **sequência de ordenação** se $\pi \cdot \beta_1 \cdot \ldots \cdot \beta_k = \iota_n$.

Definição 11. *Seja β um modelo de rearranjo e π uma permutação qualquer. O menor número de operações pertencentes a β necessárias para ordenar π é definido como **distância de ordenação** de π com respeito a β e é denotado por $d_\beta(\pi)$, uma forma reduzida de $d_\beta(\pi,\iota_n)$.*

A composição de permutações nos permite perceber que calcular $d_\beta(\pi,\sigma)$ é equivalente a calcular $d_\beta(\sigma^{-1}\cdot\pi, \sigma^{-1}\cdot\sigma) = d_\beta(\sigma^{-1}\cdot\pi,\iota_n) = d_\beta(\sigma^{-1}\cdot\pi)$. Por exemplo, seja $\sigma = (3\ 1\ 4\ 2)$ e $\pi = (2\ 1\ 4\ 3)$, então $\sigma^{-1} = (2\ 4\ 1\ 3)$, $\sigma^{-1}\cdot\pi = (4\ 2\ 3\ 1)$ e vemos que $d_\rho(\pi,\sigma) = d_\rho(\sigma^{-1}\cdot\pi) = 2$. Por tal motivo, é comum lidar apenas com as distâncias de ordenação de permutações e chamá-las simplesmente de distância.

Definição 12. *Em um **Problema de Ordenação**, deve-se encontrar a distância de ordenação de uma permutação π com respeito a um modelo de rearranjo β, ou seja, encontrar $d_\beta(\pi)$.*

Definição 13. *A maior distância de uma permutação em S_n (ou $S_n^{\pm}$) com respeito a um modelo de rearranjo β é chamada de **diâmetro** e é denotada por $D_\beta(n)$.*

Enquanto os limitantes dos diâmetros são descritos em termos do tamanho n do grupo simétrico (ou hiperoctaedral), os limitantes das distâncias de ordenação são normalmente descritos em termos de certas características da permutação dada. A mais simples dessas características é o ***breakpoint***[1].

Definição 14. *Um* ***breakpoint de reversão*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $|\pi_{i+1} - \pi_i| \neq 1$*, para* $0 \leq i \leq n$*. A permutação identidade é a única que não tem breakpoints de reversão. Exemplo:* $\pi = (0\,\mathbf{.}\,3\ 2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\,\mathbf{.}\,6)$.

Definição 15. *Um* ***breakpoint de reversão de prefixo*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $|\pi_{i+1} - \pi_i| \neq 1$*, para* $1 \leq i \leq n$*, e* (π_0, π_1) *nunca é um breakpoint de reversão de prefixo. A permutação identidade é a única que não tem breakpoints de tal tipo. Exemplo:* $\pi = (0\ 3\ 2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\,\mathbf{.}\,6)$.

Definição 16. *Um* ***breakpoint de reversão de sufixo*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $|\pi_{i+1} - \pi_i| \neq 1$*, para* $0 \leq i \leq n-1$ *e* (π_n, π_{n+1}) *nunca é um breakpoint de reversão de prefixo. A permutação identidade é a única que não tem breakpoints de tal tipo. Exemplo:* $\pi = (0\,\mathbf{.}\,3\ 2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\ 6)$.

Definição 17. *Um* ***breakpoint de reversão de prefixo e reversão de sufixo*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $|\pi_{i+1} - \pi_i| \neq 1$*, para* $1 \leq i \leq n-1$*, e* (π_0, π_1) *e* (π_n, π_{n+1}) *nunca são breakpoints de reversão de prefixo e reversão de sufixo. A permutação identidade e a permutação reversa são as únicas que não têm nenhum breakpoint desse tipo. Exemplo:* $\pi = (0\ 3\ 2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\ 6)$.

Definição 18. *Um* ***breakpoint de transposição*** *ou um* ***breakpoint de reversão com sinal*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $\pi_{i+1} - \pi_i \neq 1$*, para* $0 \leq i \leq n$*. A permutação identidade é a única que não tem breakpoints de transposição ou breakpoints de reversão com sinal. Exemplo:* $\pi_1 = (0\,\mathbf{.}\,3\,\mathbf{.}\,2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\,\mathbf{.}\,6)$ *e* $\pi_2 = (+0\,\mathbf{.}\,{-5}\ -4\,\mathbf{.}\,{+3}\,\mathbf{.}\,{+2}\,\mathbf{.}\,{-1}\,\mathbf{.}\,{+6})$.

Definição 19. *Um* ***breakpoint de transposição de prefixo*** *ou um* ***breakpoint de reversão de prefixo com sinal*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $\pi_{i+1} - \pi_i \neq 1$*, para* $1 \leq i \leq n$*, e* (π_0, π_1) *nunca é um breakpoint de transposição de prefixo ou de reversão de prefixo com sinal. A permutação identidade é a única que não tem breakpoints de tal tipo. Exemplo:* $\pi_1 = (0\ 3\,\mathbf{.}\,2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\,\mathbf{.}\,6)$ *e* $\pi_2 = (0\ -5\ -4\,\mathbf{.}\,3\,\mathbf{.}\,2\,\mathbf{.}\,{-1}\,\mathbf{.}\,6)$.

Definição 20. *Um* ***breakpoint de transposição de sufixo*** *ou um* ***breakpoint de reversão de sufixo com sinal*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $\pi_{i+1} - \pi_i \neq 1$*, para* $0 \leq i \leq n-1$*, e* (π_n, π_{n+1}) *nunca é um breakpoint de transposição de sufixo ou de reversão de sufixo com sinal. A permutação identidade é a única que não tem breakpoints de tal tipo. Exemplo:* $\pi_1 = (0\,\mathbf{.}\,3\,\mathbf{.}\,2\,\mathbf{.}\,4\ 5\,\mathbf{.}\,1\ 6)$ *e* $\pi_2 = (0\,\mathbf{.}\,{-5}\ -4\,\mathbf{.}\,3\,\mathbf{.}\,2\,\mathbf{.}\,{-1}\ 6)$.

[1]Quando for conveniente, marcaremos os *breakpoints* das permutações com "**.**".

Definição 21. *Um* ***breakpoint de transposição de prefixo e transposição de sufixo*** *ou um* ***breakpoint de reversão de prefixo com sinal e reversão de sufixo com sinal*** *é um par de elementos* (π_i, π_{i+1}) *de* π *tais que* $\pi_{i+1} - \pi_i \neq 1$*, para* $1 \leq i \leq n-1$*, e* (π_0, π_1) *e* (π_n, π_{n+1}) *nunca são breakpoints de transposição de prefixo e transposição de sufixo ou de reversão de prefixo com sinal e reversão de sufixo com sinal. A permutação identidade é a única que não tem breakpoints do primeiro tipo. Ela e a permutação reversa com sinal são as únicas que não tem breakpoints do segundo tipo. Exemplo:* $\pi_1 = (0\ 3\mathbf{.}2\mathbf{.}4\ 5\mathbf{.}1\ 6)$ *e* $\pi_2 = (0\ -5\ -4\mathbf{.}3\mathbf{.}2\mathbf{.}-1\ 6)$.

Intuitivamente, nos problemas sem sinal que envolvem algum tipo de reversão, um *breakpoint* ocorre entre dois elementos consecutivos em π que não são consecutivos em ι_n nem em η_n. Nos problemas que envolvem apenas algum tipo de transposição, ele ocorre entre dois elementos consecutivos em π que não são consecutivos em ι_n. Em problemas com sinal, um *breakpoint* ocorre entre elementos que não são consecutivos em ι_n nem em $\bar{\eta}_n$.

O número de *breakpoints* de um certo tipo β que existe em uma permutação π qualquer é dado por $b_\beta(\pi)$. Por exemplo, na permutação $\pi = (0\ 3\ 2\ 4\ 1\ 5)$, o número de *breakpoints* de reversão é 4, ou seja, $b_\rho(\pi) = 4$, enquanto o número de *breakpoints* de reversão de prefixo com sinal e reversão de sufixo com sinal é 3, ou seja, $b_{\bar{\rho}_p\bar{\rho}_s}(\pi) = 3$. A variação desse número causada por um rearranjo λ qualquer é dada por $\Delta b_\beta(\pi, \lambda) = b_\beta(\pi \cdot \lambda) - b_\beta(\pi)$. Por exemplo, para a permutação π anterior e a reversão de prefixo com sinal $\bar{\rho}_p(2)$, $\Delta b_{\bar{\rho}_p\bar{\rho}_s}(\pi, \bar{\rho}_p) = 0$, uma vez que $\pi \cdot \bar{\rho}_p = (0\ -2\ -3\ 4\ 1\ 5)$ e $b_{\bar{\rho}_p\bar{\rho}_s}(\pi \cdot \bar{\rho}_p) = 3$.

Quando um par de elementos não forma um *breakpoint* dizemos que ele forma uma **adjacência**. No restante deste texto, será comum usarmos apenas a palavra *breakpoint* para caracterizar qualquer um dos tipos de *breakpoints* definidos, fazendo diferenciações quando o contexto não deixar claro qual é o tipo.

Definição 22. *Uma* ***strip***[2] $\langle \pi_i, \ldots, \pi_j \rangle$ *é uma subsequência de* π*, com* $1 \leq i \leq j \leq n$ *tal que (i)* $i = 1$ *ou* (π_{i-1}, π_i) *é um breakpoint; (ii)* $j = n$ *ou* (π_j, π_{j+1}) *é um breakpoint; e (iii) os outros elementos da subsequência formam adjacências.*

Se uma strip tem apenas um elemento, ela é chamada de ***singleton****.*

Em permutações sem sinal, uma strip de comprimento maior ou igual a dois é dita ***ascendente*** *se* $\pi_k = \pi_{k+1} - 1$ *para todo* $i \leq k < j$ *e é dita* ***descendente*** *caso* $\pi_k = \pi_{k+1} + 1$[3]*. Exemplo: a permutação* $\pi = (2\ 5\ 6\ 4\ 3\ 1)$ *possui quatro strips se alguma variação de breakpoint de reversão está sendo considerada (dois singletons* $\langle 2 \rangle$ *e* $\langle 1 \rangle$*, uma strip ascendente* $\langle 5, 6 \rangle$ *e uma strip descendente* $\langle 4, 3 \rangle$*) e possui cinco strips se apenas alguma variação de breakpoint de transposição está sendo considerada (nesse caso,* $\langle 4 \rangle$ *e* $\langle 3 \rangle$ *são singletons).*

Em permutações com sinal, uma strip é dita ***positiva*** *se seus elementos são positivos e é dita* ***negativa*** *caso contrário. Exemplo: a permutação* $\pi = (3\ 4\ -1\ -2\ -6\ -5)$ *possui quatro*

[2]Será comum sublinharmos uma *strip* no decorrer do texto, para melhor destaque.

[3]Claramente, não existem *strips* descendentes para os problemas que envolvem apenas variações de transposições.

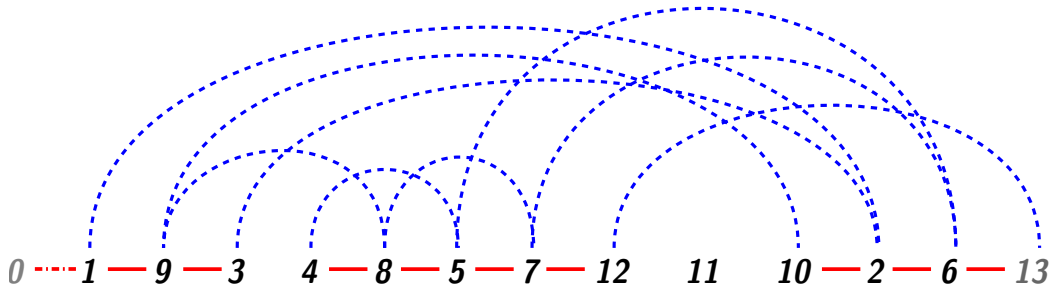


Figura 1: Grafo de *breakpoints* para a permutação $\pi = (0\ 1\ 9\ 3\ 4\ 8\ 5\ 7\ 12\ 11\ 10\ 2\ 6\ 13)$. A aresta $(0, 1)$ sempre deve existir se operações de prefixo estão envolvidas e a aresta $(6, 13)$ sempre deve existir se operações de sufixo estão envolvidas

strips (uma strip positiva $\langle 3, 4\rangle$*, dois singletons* $\langle -1\rangle$ *e* $\langle -2\rangle$ *e uma strip negativa* $\langle -6, -5\rangle$*).*

Quando lidamos com problemas que envolvem algum tipo de reversão, podemos representar uma permutação por meio de um grafo de *breakpoints* [3], conforme descrito a seguir. Em problemas que envolvem variações de transposições, grafos de ciclos [4] são mais comumente utilizados para a representação.

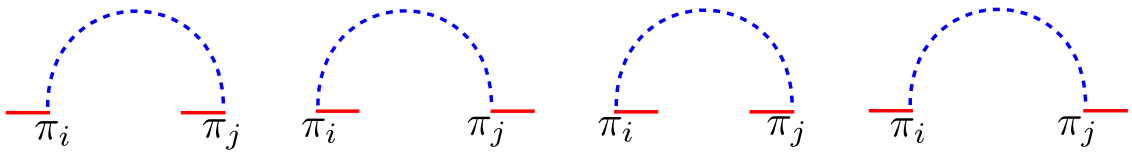
Definição 23. *Um* ***grafo de breakpoints*** *de uma permutação* π *sem sinal é um grafo* $G_b(\pi) = (V, E)$ *no qual o conjunto de vértices é dado por* $V = \{\pi_0, \pi_1, \ldots, \pi_{n+1}\}$ *e o conjunto de arestas por* $E = E_V \cup E_A$*, onde* E_V *é o conjunto de arestas vermelhas e* E_A *é o conjunto de arestas azuis. Existe uma aresta* e *vermelha se e somente se (i)* $\mathrm{e} = (\pi_i, \pi_{i+1})$ *e* (π_i, π_{i+1}) *é um breakpoint,* $0 \leq i \leq n$*; (ii)* $\mathrm{e} = (\pi_0, \pi_1)$ *e operações de prefixo estão envolvidas; ou (iii)* $\mathrm{e} = (\pi_n, \pi_{n+1})$ *e operações de sufixo estão envolvidas. Existe uma aresta* e *azul se e somente se* $\mathrm{e} = (\pi_i, \pi_j)$ *para algum* $0 \leq i < j \leq n + 1$ *com* $\pi_j = \pi_i \pm 1$ *e* $j \neq i + 1$*.*

A ideia do grafo de *breakpoints* é que arestas vermelhas descrevem a ordem dos elementos na permutação atual e arestas azuis descrevem a ordem na permutação identidade (alvo). A convenção é que ele é desenhado da esquerda para a direita, arestas vermelhas são linhas retas e arestas azuis são arcos pontilhados, como pode ser visto na Figura 1. Assim, apesar do grafo não ser direcionado, podemos dizer que uma aresta (π_i, π_j), com $0 \leq i < j \leq n + 1$, começa em π_i e termina em π_j.

Seja (π_i, π_j) uma aresta azul. Como $\pi_j = \pi_i \pm 1$, existe pelo menos uma aresta vermelha que começa ou termina em π_i assim como existe pelo menos uma aresta vermelha que começa ou termina em π_j. Dessa forma, podemos classificar tal aresta azul em um (ou mais) dos quatro tipos diferentes mostrados na Figura 2.

2.1 Problema de Ordenação por Reversões

Um dos problemas mais estudados em Rearranjos de Genomas é o de Ordenação por Reversões (SBR, de *Sorting by Reversals*), para o qual existem vários trabalhos e algoritmos de aproximação.



(a) Tipo 1 (b) Tipo 2 (c) Tipo 3 (d) Tipo 4

Figura 2: Classificação das arestas azuis

Kececioglu e Sankoff [33] em 1993 apresentaram o primeiro algoritmo de aproximação, com fator 2, que consiste em, a cada passo, aplicar uma reversão que remova a maior quantidade de *breakpoints* possível dando prioridade para aquelas que ainda deixam *strips* decrescentes. Em seguida, Bafna e Pevzner [3] introduziram a ideia do grafo de *breakpoints* e com isso apresentaram outros dois algoritmos, com fatores 1,8 e 1,75. Em 1998 Christie [16] introduziu outra estrutura, o grafo de reversões, e foi capaz de melhorar o fator para 1,5. Finalmente, em 2002 Berman *et al.* [8] apresentaram o algoritmo de melhor fator até o momento, 1,375.

Em 1999 Caprara [12] provou que tal problema é NP-difícil. Além disso, Berman e Karpinski [7] mostraram que ele não é aproximável por um fator menor do que 1,0008.

Lema 1. **[3]** *Seja π uma permutação sem sinal qualquer. Então, $d_\rho(\pi) \geq \frac{b_\rho(\pi)}{2}$.*

Lema 2. **[33]** *Seja π uma permutação sem sinal qualquer. Então, $d_\rho(\pi) \leq b_\rho(\pi) - 1$.*

Bafna e Pevzner [3], com a introdução do grafo de *breakpoints*, também foram capazes de provar a conjectura de Golan[4], que afirmava que $D_\rho(n) = n - 1$ e que as únicas permutações com tal distância de reversão eram $\gamma^{(n)}$ e sua inversa, sendo a primeira definida recursivamente da seguinte forma:

$$\gamma^{(n+1)} = \begin{cases} (1) & \text{se } n \text{ é zero} \\ (\gamma_1^{(n)}\ \gamma_2^{(n)}\ \ldots\ \gamma_{n-1}^{(n)}\ n+1\ \gamma_n^{(n)}) & \text{se } n \text{ é ímpar} \\ (\gamma_1^{(n)}\ \gamma_2^{(n)}\ \ldots\ \gamma_{n-2}^{(n)}\ n+1\ \gamma_n^{(n)}\ \gamma_{n-1}^{(n)}) & \text{se } n \text{ é par} \end{cases} .$$

2.2 Problema de Ordenação por Reversões com Sinal

Outro problema relacionado aos nossos é o de Ordenação por Reversões com Sinal (SbSigR, de *Sorting by Signed Reversals*). Ao contrário da sua versão sem sinal, ele é polinomial, conforme mostrado por Hannenhalli e Pevzner [30] com um algoritmo de complexidade de tempo $O(n^4)$. Tal algoritmo foi apresentado de maneira muito mais simples por Bergeron [6].

Desde então, melhorias foram sendo realizadas para diminuir tal complexidade, sendo que $O(n^{3/2}\sqrt{\lg n})$ é a melhor conhecida [42]. Além disso, existe um algoritmo de tempo $O(n)$ que é capaz de calcular a distância de ordenação exata sem mostrar a sequência de ordenação [1]. O diâmetro do SbSigR também é conhecido e vale $D_{\bar{\rho}}(n) = n + 1$ [24].

[4]Holger Golan. Comunicação Pessoal, 1991.

2.3 Problema de Ordenação por Transposições

Outro problema importante em Rearranjos de Genomas é o de Ordenação por Transposições (SBT, de *Sorting by Transpositions*). Esse problema foi introduzido em 1998 por Bafna e Pevzner [4], que apresentaram três algoritmos de aproximação para o problema, com fatores 2, 1,75 e 1,5. O melhor algoritmo de aproximação para o SBT, no entanto, possui fator de aproximação 1,375 e foi apresentado por Elias e Hartman [21].

Lema 3. [4] *Seja π uma permutação sem sinal qualquer. Então, $d_\tau(\pi) \geq \frac{b_\tau(\pi)}{3}$.*

Bafna e Pevzner [4] também apresentaram limitantes para o diâmetro, mostrando que $\left\lfloor \frac{n}{2} \right\rfloor \leq D_\tau(n) \leq \frac{3n}{4}$. Os melhores limitantes inferior e superior atuais foram encontrados respectivamente por Elias e Hartman [21] e Eriksson *et al.* [23], sendo que $\left\lfloor \frac{n+1}{2} \right\rfloor + 1 \leq D_\tau(n) \leq \left\lfloor \frac{2n-2}{3} \right\rfloor$.

Esse problema foi provado ser NP-difícil apenas em 2011 por Bulteau *et al.* [11], com uma redução do problema SAT. Além disso, eles provaram que dada uma permutação π, descobrir se é possível ordenar π com exatamente $\frac{b_\tau(\pi)}{3}$ transposições também é NP-difícil.

2.4 Problema de Ordenação por Reversões e Transposições

Os problemas de Ordenação por Reversões e Ordenação por Transposições, conforme já mencionado, são bem estudados e possuem vários resultados conhecidos. Entretanto, um modelo de rearranjo pode ser composto por mais de um tipo de operação. Walter *et al.* [43] propuseram o Problema de Ordenação por Reversões e Transposições (SBRT, de *Sorting by Reversals and Transpositions*), que permite ambas operações. Além disso, eles forneceram um algoritmo de 3-aproximação para tal problema.

Lema 4. [43] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho\tau}(\pi) \geq \frac{b_\rho(\pi)}{3}$.*

Lema 5. [43] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho\tau}(\pi) \leq b_\rho(\pi)$.*

Teorema 6. [43] SBRT *é 3-aproximável.*

Para a versão sem sinal, o melhor algoritmo de aproximação conhecido é de Rahman *et al.* [39], com fator de aproximação $2k$ onde k é o fator de aproximação do algoritmo para decomposição em ciclos do grafo de ciclos [4]. Na época, os autores usaram o algoritmo para decomposição fornecido por Lin e Jiang [36], cujo fator é de $1.4193 + \epsilon$, de forma que seu algoritmo ficou com fator $2.8386 + \delta$. Vale mencionar que, recentemente, Chen [13] apresentou um fator de aproximação melhor para o problema da decomposição: $1.4167 + \epsilon$.

O SBRT permanece com complexidade desconhecida e não havia resultados na literatura com relação a limitantes para o diâmetro e outros limitantes para a distância.

2.5 Problema de Ordenação por Reversões com Sinal e Transposições

Walter *et al.* [43] também propuseram o Problema de Ordenação por Reversões com Sinal e Transposições (SbSigRT, de *Sorting by Signed Reversals and Transpositions*) e forneceram um algoritmo de 2-aproximação para esse problema. Eles também mostraram que $D_{\bar{\rho}\tau}(n) \geq \left\lfloor \frac{n}{2} \right\rfloor + 2$, o único limitante conhecido para o diâmetro deste problema. Além disso, mostraram que $d_{\bar{\rho}\tau}(\bar{\iota_n}) = \left\lfloor \frac{n}{2} \right\rfloor + 2$, conjecturando que $D_{\bar{\rho}\tau}(n) = d_{\bar{\rho}\tau}(\bar{\iota_n})$.

Gu *et al.* [29] apresentaram um algoritmo de 2-aproximação para o Problema de Ordenação por Reversões, Transposições e Transreversões com Sinal. Essa última operação é uma transposição na qual aplica-se uma reversão a um dos segmentos envolvidos.

Lin e Xue [35] apresentaram algoritmos de 2-aproximação para três problemas de ordenação com permutações com sinal: os dois já citados e um terceiro, considerando reversões, transposições, transreversões e revrevs. Essa última operação consiste em uma transposição na qual os dois segmentos envolvidos aparecem com ordem e orientação invertidas. Para este terceiro problema, os autores apresentaram um algoritmo de 1.75-aproximação. Esse fator foi melhorado por Hartman e Sharan [31] para 1.5.

Todos os problema citados nesta seção permanecem com complexidade desconhecida. Além disso, quase não há resultados com relação aos diâmetros e limitantes para as distâncias.

2.6 Problema de Ordenação por Reversões de Prefixo

O Problema da Ordenação de Panquecas, ou Problema de Ordenação por Reversões de Prefixo (SbPR, de *Sorting by Prefix Reversals*), foi descrito em 1975 por Dweighter [20] com uma pequena história:

> O chefe daqui é descuidado e quando ele prepara panquecas todas saem com tamanhos diferentes. Assim, quando eu as entrego a um cliente, no caminho para a mesa eu as rearranjo (de forma que a menor fique no topo, e assim por diante até que a maior fique na parte inferior) pegando várias do topo e invertendo-as, repetindo isso (variando a quantidade que eu inverto) quantas vezes forem necessárias. Se existem n panquecas, qual é o número máximo de movimentos (em função de n) que eu sempre terei que usar para rearranjá-las?

Em 1979, Gates e Papadimitriou [28] apresentaram os primeiros resultados computacionais para o SbPR. Em seu artigo, foram apresentados limitantes superior e inferior para o diâmetro, que é o problema descrito pelo garçom, e eles mostraram que $\frac{17n}{16} \leq D_{\rho_p}(n) \leq \frac{(5n+3)}{3}$. No entanto, o diâmetro ainda é desconhecido. Sabe-se os valores exatos de $D_{\rho_p}(n)$ para $n \leq 19$ [17]. Heydari e Sudborough [32] mostraram que $D_{\rho_p}(n) \geq \frac{15n}{14}$, o melhor limitante inferior conhecido atualmente. O limitante superior apresentado por Gates e Papadimitriou [28] foi melhorado apenas em 2009 por Chitturi *et al.* [15], que mostraram que $D_{\rho_p}(n) \leq \frac{18n}{11} + O(1)$, o melhor valor até o momento.

Lema 7. [25] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p}(\pi) \geq b_{\rho_p}(\pi)$.*

Fischer e Ginzinger [25] apresentaram um algoritmo de aproximação com fator 2 para o problema, que é o melhor conhecido até o momento. Eles usaram o grafo de *breakpoints* de uma permutação para decidir quais reversões realizar em cada tipo de aresta azul, conforme mostra o Lema 8, adaptado de seu trabalho. Essas arestas são chamadas *arestas boas de prefixo*.

Lema 8. [25] *Seja $\pi \neq \iota_n$ uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo duas reversões de prefixo que pode remover um breakpoint existe se $G_b(\pi)$ contém pelo menos uma das quatro arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 1;

(2) (π_1, π_j) do tipo 3;

(3) (π_i, π_j) do tipo 2 com $i \neq 0$;

(4) (π_i, π_j) do tipo 3 com $i > 1$.

Chamaremos o algoritmo desenvolvido por Fischer e Ginzinger [25] de `2-PR`. Ele percorre o grafo de *breakpoints* da permutação da esquerda para a direita tentando encontrar uma aresta boa de prefixo na ordem em que elas aparecem no Lema 8. No entanto, se uma permutação não contém uma aresta boa de prefixo, ela é da forma dada em (1), conforme mostram os Lemas 9, 10 e 11. O Lema 12 mostra como lidar com tais permutações.

Lema 9. [25] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de prefixo. Então, π não contém nenhum singleton.*

Lema 10. [25] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de prefixo. Então, a primeira aresta de π é do tipo 2, a última é do tipo 1 e todas as outras são do tipo 4.*

Lema 11. [25] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de prefixo. Então π é da forma*

$$(\underbrace{p_1 \ \ldots \ 1}_{\ell_1} \ \underbrace{p_2 \ \ldots \ p_1{+}1}_{\ell_2} \ \ldots\ldots \ \underbrace{t \ \ldots \ p_{b_{\rho_p}(\pi)-1}{+}1}_{\ell_{b_{\rho_p}(\pi)}} \ \underline{t{+}1 \ t{+}2 \ \ldots \ n} \ ) \tag{1}$$

onde $t \leq n$. Em outras palavras, π consiste em $b_{\rho_p}(\pi) \geq 2$ strips decrescentes de tamanho $\ell_i \geq 2$, para todo $1 \leq i \leq b_{\rho_p}(\pi)$, de tal forma que se tais strips fossem crescentes, a permutação seria igual à identidade.

Lema 12. [25] *Seja π uma permutação sem sinal de tamanho n que é da forma dada em (1). Para essa permutação, a sequência de $2b_{\rho_p}(\pi)$ reversões de prefixo*

$$\rho_p(t) \cdot \rho_p(t - \ell_1) \cdot \rho_p(t) \cdot \rho_p(t - \ell_2) \cdot \ldots \cdot \rho_p(t) \cdot \rho_p(t - \ell_{b_{\rho_p}(\pi)}) \tag{2}$$

quando aplicada a π, transforma-a na permutação identidade.

O algoritmo 2-PR tem complexidade de tempo $O(n^2)$, uma vez que encontrar uma aresta boa leva tempo $O(n)$, aplicar a reversão correspondente à aresta encontrada leva tempo $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é $O(n)$. Além disso, ele fornece um limitante superior para SbPR, como mostra o Lema 13, garantindo seu fator de aproximação 2.

Lema 13. [25] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p}(\pi) \leq 2b_{\rho_p}(\pi)$.*

Teorema 14. SbPR *é 2-aproximável.*

O algoritmo implementado por Fischer e Ginzinger [25] também escolhia sempre o tipo 1 de aresta azul sobre os outros tipos, por este necessitar apenas de uma reversão de prefixo para remover um *breakpoint*. Dessa forma, e também porque os autores geraram aleatoriamente apenas 10000 permutações de comprimento até 71 para os testes, o fator de aproximação real apresentou-se bem melhor do que 2 ($\approx 1,2$).

A complexidade do SbPR era desconhecida até 2011, quando Bulteau *et al.* [10] provaram que ele é NP-difícil por uma redução do problema 3-SAT. Além disso, provaram que decidir se uma dada permutação π pode ser ordenada em $b_{\rho_p}(\pi)$ reversões de prefixo também é NP-difícil.

2.7 Problema de Ordenação por Reversões de Prefixo com Sinal

A versão com sinal do SbPR, denominada de Problema da Ordenação de Panquecas Queimadas, ou Problema de Ordenação por Reversões de Prefixo com Sinal (SbSigPR, de *Sorting by Signed Prefix Reversals*), foi introduzida por Gates e Papadimitriou [28] e requer que, além de ordenadas, as panquecas estejam com o lado queimado voltado para baixo. Além de introduzir o problema, os autores apresentaram limitantes inferior e superior iniciais para seu diâmetro, demonstrando que $\frac{3n}{2} - 1 \leq D_{\bar{\rho}_p}(n) \leq 2n + 3$. Cohen e Blum [18] mostraram que $\frac{3n}{2} \leq D_{\bar{\rho}_p}(n) \leq 2n - 2$ e conjecturaram que $\bar{\iota}_n$ seria a permutação mais difícil de se ordenar. Heydari e Sudborough [32] mostraram então que $d_{\bar{\rho}_p}(\bar{\iota}_n) \leq \frac{3n+3}{2}$ para todo $n \equiv 3(\bmod\ 4)$ e $n \geq 23$. Finalmente, Cibulka [17] apresentou o melhor limitante superior para o diâmetro, $2n - 6$, válido para $n \geq 16$, e provou que a conjectura dada por Cohen e Blum [18] estava errada ao mostrar que $\bar{\iota}_n$ não é a mais difícil quando $n = 15$. Além disso, ele mostrou que $d_{\bar{\rho}_p}(\bar{\iota}_n) \geq \left\lfloor \frac{3n+3}{2} \right\rfloor$ para todo n, implicando que $d_{\bar{\rho}_p}(\bar{\iota}_n)$ é igual a tal valor quando $n \equiv 3(\bmod\ 4)$ e $n \geq 15$. Sabe-se os valores exatos de $D_{\bar{\rho}_p}(n)$ para $n \leq 17$ [17].

Lema 15. [18] *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho}_p}(\pi) \geq b_{\bar{\rho}_p}(\pi)$.*

A complexidade do SbSigPR ainda é desconhecida e o melhor algoritmo de aproximação conhecido tem fator 2, apresentado por Cohen e Blum [18]. Para atingir tal fator, eles mostraram como é sempre possível remover um *breakpoint* com no máximo duas operações. Chamaremos tal algoritmo de 2-SPR e o descrevemos a seguir, com uma diferença, em relação ao algoritmo original, de que primeiro deve-se tentar remover um *breakpoint* com uma reversão de prefixo com sinal, ideia essa que foi descrita por Galvão [26]. Os passos principais de 2-SPR são:

(1) Se existe um $\pi_j = -\pi_1 + 1$ em π, com $2 \leq j \leq n+1$, então a reversão $\bar{\rho}_p(j-1)$ remove um *breakpoint*;

(2) Se π tem um elemento positivo fora de ordem, seja $\pi_i = k$ o maior elemento desse tipo:

  (i) Se existe um $\pi_j = -(k+1)$ tal que $i < j \leq n$, então tem-se que $\pi = (\ldots\ldots\ k\ \ldots\ldots\ -(k+1)\ \ldots\ldots)$ e a sequência $\bar{\rho}_p(j) \cdot \bar{\rho}_p(j-i)$ remove um *breakpoint*;

  (ii) Se existe um $\pi_j = -(k+1)$ tal que $j \leq i$, então $\pi = (\ldots\ldots\ -(k+1)\ \ldots\ldots\ k\ \ldots\ldots)$ e a sequência $\bar{\rho}_p(i) \cdot \bar{\rho}_p(i-j)$ remove um *breakpoint*;

  (iii) Se não existe um $\pi_j = -(k+1)$, então $\pi = (\ldots\ldots\ k\ \ldots\ldots\ \underline{k{+}1\ k{+}2\ \ldots\ n})$ e a sequência $\bar{\rho}_p(i) \cdot \bar{\rho}_p(k)$ remove um *breakpoint*.

(3) Se π não tem um elemento positivo fora de ordem:

  (i) Se existe um $\pi_i = -(k+1)$ e um $\pi_j = -k$ para algum $k \geq 1$ tal que $i+1 < j$, então $\pi = (\ldots\ldots\ -(k+1)\ \ldots\ldots\ -k\ \ldots\ldots)$ e a sequência $\bar{\rho}_p(i) \cdot \bar{\rho}_p(j-1)$ remove um *breakpoint*;

  (ii) Se não existem elementos como os descritos no item acima, então π é da forma descrita em (3), conforme o Lema 16 mostra, e a sequência de $2b_{\bar{\rho}_p}(\pi)$ reversões de prefixo com sinal descrita em (4) ordena π, como o Lema 17 mostra.

Lema 16. [26] *Seja π uma permutação com sinal sem elementos positivos fora de ordem para a qual não existem elementos π_i e π_j tais que $\pi_i = -(k+1)$ e $\pi_j = -k$ para algum $k \geq 1$ com $i+1 < j$. Então π é da forma*

$$\pi = (\underbrace{-p_1 \ldots -1}_{\ell_1}\ \underbrace{-p_2 \ldots -(p_1{+}1)}_{\ell_2}\ \ldots\ldots\ \underbrace{-t \ldots -(p_{b_{\bar{\rho}_p}(\pi)-1}{+}1)}_{\ell_{b_{\bar{\rho}_p}(\pi)}}\ \underline{t{+}1\ t{+}2\ \ldots\ n}) \tag{3}$$

onde $t \leq n$. Em outras palavras, π consiste em $b_{\bar{\rho}_p}(\pi) \geq 2$ strips negativas de tamanho $\ell_i \geq 1$, para todo $1 \leq i \leq b_{\bar{\rho}_p}(\pi)$, de tal forma que se tais strips fossem positivas, a permutação seria igual à identidade.

Lema 17. [26] *Seja π uma permutação com sinal da forma descrita em (3). A seguinte sequência de $2b_{\bar{\rho}_p}(\pi)$ reversões de prefixo ordena π:*

$$\bar{\rho}_p(t) \cdot \bar{\rho}_p(t-\ell_1) \cdot \bar{\rho}_p(t) \cdot \bar{\rho}_p(t-\ell_2) \cdot \ldots \cdot \bar{\rho}_p(t) \cdot \bar{\rho}_p(t-\ell_{b_{\bar{\rho}_p}(\pi)}). \tag{4}$$

O algoritmo `2-SPR` tem complexidade de tempo $O(n^2)$, uma vez que procurar pelos elementos requeridos leva tempo $O(n)$, aplicar as operações necessárias também leva tempo $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é $O(n)$. O Lema 18 e o Teorema 19 mostram como ele garante o fator de aproximação comentado.

Lema 18. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}}(\pi) \leq 2b_{\bar{\rho_p}}(\pi)$.*

Teorema 19. SbSigPR *é 2-aproximável.*

2.8 Problema de Ordenação por Transposições de Prefixo

O problema de Ordenação por Transposições de Prefixo (SbPT, de *Sorting By Prefix Transpositions*) também permanece aberto. Foi introduzido em 2002 por Dias e Meidanis [19], que apresentaram dois algoritmos de aproximação para o problema.

O primeiro algoritmo apresentado por Dias e Meidanis [19], de fator 3, surgiu do simples fato que uma transposição pode ser sempre substituída por duas transposições de prefixo, de forma que qualquer algoritmo de k-aproximação para o SbT pode ser transformado em um algoritmo de $2k$-aproximação para o SbPT. Assim, o algoritmo de aproximação com fator 1,5 apresentado por Bafna e Pevzner [4] garante a existência de um algoritmo de 3-aproximação para o SbPT.

O segundo algoritmo apresentado por Dias e Meidanis [19], o qual chamaremos de `2-PT`, é a melhor aproximação conhecida para o SbPT. O que tal algoritmo faz é sempre remover um *breakpoint* usando uma transposição de prefixo para aumentar a primeira *strip*, levando-a para antes de seu próximo elemento, isto é, unindo o último elemento i da primeira *strip*, ao elemento $i + 1$, conforme mostra o Lema 20.

Lema 20. [19] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer. Sempre é possível obter uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p}(\pi, \tau_p) \leq -1$.*

Os autores também demonstraram que existe no máximo uma transposição de prefixo que remove dois *breakpoints* de uma vez, conforme o Lema 21 mostra. Isso leva a outro algoritmo de 2-aproximação para o problema, um guloso [26], que será chamado `2-PTg`.

Lema 21. [19] *Seja π uma permutação qualquer. Existe no máximo uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p}(\pi, \tau_p) = -2$.*

Em suma, `2-PTg` primeiro tenta encontrar uma transposição de prefixo que remove dois *breakpoints* de uma vez e, se isso não for possível, ele remove apenas um, da mesma forma que `2-PT` faz. Além disso, ambos algoritmos são $O(n^2)$ com relação à complexidade de tempo, uma vez que encontrar o último elemento da primeira *strip* leva tempo $O(n)$ no pior caso, aplicar uma transposição também leva tempo $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é $O(n)$. Os Lemas 22, 23 e 24 e o Teorema 25 mostram como ambos algoritmos possuem fator de aproximação 2.

Lema 22. [19] *Seja π uma permutação sem sinal qualquer. Então, $d_\tau(\pi) \geq \frac{b_\tau(\pi)}{2}$.*

Lema 23. [19] *Seja π uma permutação qualquer e $\tau_p(i, j)$ uma transposição de prefixo tal que $\pi \cdot \tau_p = \iota_n$. Então, $\pi_i = 1$ e $\Delta b_{\tau_p}(\pi, \tau_p) = -2$.*

Lema 24. [19] *Seja π uma permutação sem sinal qualquer. Então, $d_{\tau_p}(\pi) \leq b_{\tau_p}(\pi) - 1$.*

Teorema 25. SbPT *é 2-aproximável.*

Além dos limitantes para a distância de ordenação, Dias e Meidanis [19] também mostraram que $\frac{n}{2} \leq D_{\tau_p}(n) \leq n - 1$. No entanto, o valor exato do diâmetro permanece desconhecido e os melhores limitantes inferior e superior conhecidos atualmente foram encontrados por Labarre [34] e Chitturi e Sudborough [14], respectivamente, sendo $\left\lfloor \frac{3n+1}{4} \right\rfloor \leq D_{\tau_p}(n) \leq n - \log_{\frac{9}{2}} n$.

2.9 Problema de Ordenação por Reversões de Prefixo e Transposições de Prefixo

Sharmin *et al.* [40] apresentaram uma nova versão do Problema das Panquecas, a qual eles chamaram de Problema das Panquecas com duas Espátulas, ou Problema de Ordenação por Reversões de Prefixo e Transposições de Prefixo (SbPRPT, de *Sorting by Prefix Reversals and Prefix Transpositions*). Esse problema permanece aberto e não há resultados com respeito ao diâmetro.

Nesse problema, podemos considerar *breakpoints* de reversão de prefixo ou *breakpoints* de transposição de prefixo. Os Lemas 26 e 27 apresentam limitantes inferiores com relação a tais tipos de *breakpoints*. Note também que $d_{\rho_p\tau_p}(\pi) \leq \min\{d_{\rho_p}(\pi), d_{\tau_p}(\pi)\}$, uma vez que as sequências de ordenação obtidas para SbPR e SbPT também são válidas para o SbPRPT, mas não necessariamente ótimas.

Lema 26. [40] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$.*

Dos Lemas 26 e 13, podemos ver que os algoritmos `2-PR` e `2-PRg` são algoritmos de 4-aproximação para o SbPRPT.

Lema 27. *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\tau_p}(\pi)}{n} \right\rceil$.*

Dos Lemas 27 e 24, podemos ver que os algoritmos `2-PT` e `2-PTg` não são algoritmos de aproximação para o SbPRPT. Apesar disso, eles devolvem uma resposta válida para esse problema. Pela falta de um limitante inferior independente de n com os *breakpoints* de transposição de prefixo, neste problema consideramos apenas os *breakpoints* de reversão de prefixo.

Sharmin *et al.* [40] ainda apresentaram um algoritmo aproximativo com fator 3 para o SbPRPT, chamado aqui de `3-PRPT`, que também usa o grafo de *breakpoints* para decidir quais operações realizar. Ele é parecido com o `2-PR`, mas permitir o uso de uma segunda operação faz com que todos os quatro tipos de arestas azuis possam ser consideradas *arestas boas*.

Antes de mostrar como lidar com cada tipo de aresta, precisamos de um conceito importante que é usado por `3-PRPT` e é apresentado no Lema 28, adaptado de Sharmin *et al.* [40].

Lema 28. [40] *Seja (π_i, π_j) uma aresta azul do tipo 4 de uma permutação π qualquer. Então existe pelo menos uma aresta vermelha (π_{k-1}, π_k) para algum $i < k < j$. Tal aresta é chamada de* ***presa****.*

Finalmente, o Lema 29 define o que fazer com cada tipo de aresta azul, também chamadas de *arestas boas de prefixo*.

Lema 29. [40] *Seja $\pi \neq \iota_n$ uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo três reversões de prefixo e transposições de prefixo que remove até dois breakpoints existe se $G_b(\pi)$ contém pelo menos uma das quatro arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 4 com $\pi_1 \neq 1$;

(2) (π_1, π_j) do tipo 1 com $\pi_1 \neq 1$;

(3) (π_i, π_j) do tipo 3 com $\pi_1 = 1$;

(4) (π_i, π_j) do tipo 2 com $\pi_1 = 1$ onde π_i é o último elemento da primeira strip de π.

`3-PRPT` percorre a permutação da esquerda para a direita tentando encontrar uma aresta boa de prefixo na ordem em que as quatro aparecem no Lema 29 e faz a operação ou operações necessárias. Lema 30 e Teorema 31 mostram que `3-PRPT`, é 3-aproximativo.

Lema 30. *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p\tau_p}(\pi) \leq \frac{3b_{\rho_p}(\pi)}{2}$.*

Teorema 31. SbPRPT *é 3-aproximável.*

A complexidade de tempo de `3-PRPT` é $O(n^2)$, uma vez que encontrar qualquer tipo de aresta pode usar tempo $O(n)$ no pior caso, aplicar a(s) devida(s) operação(ões) também é da ordem de $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é da ordem de $O(n)$.

2.10 Problemas de Ordenação com Operações Ponderadas

Normalmente, os problemas de ordenação consideram que todas as operações do modelo considerado têm o mesmo custo. No entanto, considerando ordenação por reversões, por exemplo, resultados preliminares sugerem que em alguns casos as reversões que aconteceram durante a evolução tendem a não ser muito longas [9], indicando que pesos baseados no tamanho do segmento envolvido possam ter um papel importante no processo evolutivo.

Dessa forma, alguns autores começaram, com mais frequência em alguns trabalhos recentes, a considerar rearranjos ponderados. Considerando apenas reversões ponderadas com base em seu comprimento (quantidade de elementos) sobre permutações sem sinal, Pinter e Skiena [38] apresentaram um algoritmo aproximativo com fator $O(\lg^2 n)$. Para o mesmo problema, Bender *et al.* [5] apresentaram uma análise para uma grande variedade de funções de custo $f(l) = l^\alpha$, $\alpha \geq 0$, onde l é o comprimento da reversão. Quando $\alpha = 1$ os autores garantiram fator de aproximação $O(\lg n)$ e quando $\alpha \geq 2$, garantiram fator 2 de aproximação. Swidan *et al.* [41] estenderam tal trabalho para permutações com sinal, conseguindo também garantir aproximação $O(\lg n)$ quando $\alpha = 1$.

Blanchette *et al.* [9] consideraram o problema da ordenação de permutações com sinal por reversões, transposições e transreversões apresentando um algoritmo simples guloso que tenta minimizar a soma ponderada do número de operações. Eriksen [22] também trabalhou com tal problema, considerando peso 2 para as transposições e peso 1 para as reversões e garantindo fator de aproximação $1+\epsilon$. Ainda no mesmo problema, Bader e Ohlebusch [2] complementaram os dois últimos trabalhos, garantindo aproximação com fator 1.5 para qualquer proporção de peso entre 1:1 e 2:1 (transposição:reversão).

Novamente, são razoavelmente recentes os esforços considerando operações ponderadas de forma que não há considerações, por exemplo, com relação ao diâmetro. Notamos também a falta de registros na literatura sobre o estudo de operações de prefixo e sufixo ponderadas.

2.11 Problemas de Ordenação com Operações que não Cortam *Strips*

Um outro tópico de estudo consiste em considerar a diferença que existe quando se permite que as operações possam cortar *strips* e quando não se permite isso durante a ordenação. Considere, por exemplo, a permutação $\pi = (10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 12\ 11)$. Sabemos que $d_{\rho_p}(\pi) = 3$ e que as operações $\rho_p(11)$, $\rho_p(12)$ e $\rho_p(11)$ a ordenam. Note que a primeira reversão comentada gera $\pi' = \pi \cdot \rho_p(11) = (12\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11)$ e, para isso, corta a *strip* $\langle 12, 11\rangle$ que existia em π. No entanto, se não fosse permitido cortar *strips*, então a $d_{\rho_p}(\pi)$ seria 4 e as operações $\rho_p(12)$, $\rho_p(2)$, $\rho_p(12)$ e $\rho_p(10)$ seriam necessárias para ordená-la.

Para alguns problemas, a ferramenta GRAAu[5], de Galvão e Dias [27], já calculou todas as permutações de tamanho pequeno (com n até 12) para as quais a distância mudaria caso o corte de *strips* não fosse permitido. Por exemplo, para o SbPR, quando $n = 12$, existem 9 permutações desse tipo para as quais a distância aumentaria em uma unidade. Além disso, não existem permutações para as quais a distância aumentaria em mais de uma unidade.

Christie [16] indicou que cortar *strips* é um movimento desnecessário que apenas aumentaria a distância de transposição das permutações. No entanto, conforme os exemplos acima, isso não é necessariamente verdade quando reversões são consideradas. Além disso, é razoável acreditar, pela lei da parcimônia, que segmentos que ficarão unidos ao final da ordenação não devem se separar após serem unidos durante a ordenação. Por isso, desejamos verificar quais são as permutações para as quais cortar ou não *strips* faz diferença.

3 Objetivos

Considerando os problemas de Ordenação por Reversões de Prefixo e Transposições de Prefixo com e sem Sinal (SbPRPT e SbSigPRPT, de *Sorting by Signed Prefix Reversals and Prefix Transpositions*), Ordenação por Reversões de Prefixo e Reversões de Sufixo com e sem

[5]Disponível em `http://mirza.ic.unicamp.br:8080/bioinfo/results.jsf`.

Sinal (SbPRSR, de *Sorting by Prefix Reversals and Suffix Reversals*, e SbSigPRSigSR, de *Sorting by Signed Prefix Reversals and Signed Suffix Reversals*), Ordenação por Reversões de Prefixo, Transposições de Prefixo, Reversões de Sufixo e Transposições de Sufixo com e sem Sinal (SbPRPTSRST, de *Sorting by Prefix Reversals, Prefix Transpositions, Suffix Reversals e Suffix Transpositions*, SbSigPRPTSigSRST, de *Sorting by Signed Prefix Reversals, Prefix Transpositions, Signed Suffix Reversals e Suffix Transpositions*), e Ordenação por Transposições de Prefixo e Transposições de Sufixo, (SbPTST, de *Sorting by Prefix Transpositions e Suffix Transpositions*) e, para cada um deles, sempre que possível, considerando Operações Ponderadas pelo tamanho dos segmentos envolvidos e Operações que não Cortam *Strips*, pretendemos estudar os seguintes tópicos:

- obter melhores limitantes para a distância;
- obter limitantes para o diâmetro;
- definir classes de permutações para as quais é possível determinar a distância exata;
- obter algoritmos de aproximação com fator melhor do que os existentes nos problemas para os quais já se conhece algum e obter algoritmos de aproximação para os outros problemas.

4 Etapas Previstas e Cronograma de Execução

As Tabelas 1 e 2 apresentam a distribuição das atividades a seguir, que serão realizadas no trabalho:

(1) Obtenção dos créditos obrigatórios em disciplinas do programa de doutorado;

(2) Revisão bibliográfica;

(3) Escrita da proposta de doutorado;

(4) Exame de Qualificação Específico (EQE);

(5) Programa de Estágio Docente (PED);

(6) Investigação sobre algoritmos de aproximação;

(7) Definição das classes para as quais é possível determinar a distância exata;

(8) Investigação sobre limitantes para o valor da distância;

(9) Investigação sobre limitantes para o valor do diâmetro;

(10) Escrita da tese;

(11) Revisão da tese;

(12) Defesa da tese.

Os itens de 6 a 9 são subdivididos para os problemas (a) SbPRSR e SbSigPRSigSR, (b) SbPRPT e SbSigPRPT, (c) SbPRPTSRST e SbSigPRPTSigSRST, (d) SbPTST.

Tabela 1: Cronograma de execução das atividades previstas – Março/2012 até Fevereiro/2014.

Etapa	2012										2013												2014	
	M	A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D	J	F
1	*	*	*	*		*	*	*	*				*	*	*	*									
2		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*						
3											*	*	*	*	*	*	*						*	
4																								*
5													*	*	*	*								
6 (a)													*						*					
6 (b)														*						*				
6 (c)															*						*			
6 (d)																*						*		
10																							*	*

Tabela 2: Cronograma de execução das atividades previstas – Março/2014 até Fevereiro/2016.

Etapa	2014										2015												2016	
	M	A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D	J	F
2		*					*					*					*					*		
5	*	*	*	*																				
6 (a)																	*	*						
6 (b)																		*	*					
6 (c)																			*	*				
6 (d)																				*	*			
7 (a)		*	*																					
7 (b)			*	*																				
7 (c)				*	*																			
7 (d)					*	*																		
8 (a)							*	*																
8 (b)								*	*															
8 (c)									*	*														
8 (d)										*	*													
9 (a)												*	*											
9 (b)													*	*										
9 (c)														*	*									
9 (d)															*	*								
10						*	*				*	*				*	*				*	*		
11																							*	
12																								*

Ressaltamos que o cronograma apresentado pode ser adaptado à medida que os avanços ocorrerem, pois tais avanços poderão nos levar a resultados mais promissores do que outros, o que nos faria dedicar mais tempo em alguma(s) atividade(s), em detrimento de outras.

5 Metodologia

O trabalho a ser realizado é, em grande parte, teórico. O estudo de teoremas e provas dos trabalhos já existentes na literatura são a principal base para o estabelecimento dos limitantes e

das classes de permutações específicas. Sempre que possível e necessário, no entanto, programas serão desenvolvidos para comprovar a validade dos resultados obtidos.

6 Resultados Iniciais

Com relação aos problemas que nos propusemos a estudar, foram obtidos alguns resultados que apresentamos, em resumo, nesta seção.

Os algoritmos já obtidos para SbPRSR, SbPTST e SbPRPTSRST foram submetidos ao LATIN'2014 (*11th Latin American Theoretical INformatics Symposium*), que aconteceu de 31 de março a 4 de abril de 2014 em Montevidéu, Uruguai, em um trabalho intitulado "*Sorting Permutations by Prefix and Suffix Versions of Reversals and Transpositions*". Os algoritmos já obtidos para SbSigPRSigSR, SbSigPRPT e SbSigPRPTSigSRST foram submetidos ao AlCoB'2014 (*1st International Conference on Algorithms for Computational Biology*), que acontecerá de 1 a 3 de julho de 2014 em Tarragona, Espanha, em um trabalho intitulado "*On Sorting of Signed Permutations by Prefix and Suffix Reversals and Transpositions*". Para todos esses problemas, bem como para SbRT e SbSigRT, obtivemos resultados relacionados aos diâmetros, que também foram submetidos ao AlCoB'2014 em um artigo intitulado "*On the Diameter of Rearrangement Problems*". Todos os artigos foram aceitos.

As Subseções 6.1 a 6.9 apresentam tais resultados. Algoritmos e demonstrações omitidas são apresentados no Apêndice A. A Subseção 6.10 apresenta maneiras de melhorar na prática os resultados obtidos pelos algoritmos desenvolvidos. Tais resultados são recentes e ainda não foram submetidos em nenhum congresso. Os Apêndices C e D apresentam famílias e classes, respectivamente, que já encontramos para alguns dos problemas que nos propusemos a estudar.

6.1 Problema de Ordenação por Reversões e Transposições

Conforme comentado na Seção 2.4, não havia resultados com relação ao diâmetro do SbRT. O Teorema 33 apresenta limitantes inferior e superior para o diâmetro, sendo que o limitante inferior é baseado na distância da família dada a seguir, apresentada no Lema 32.

$$\pi_n^5 = \begin{cases} (2\ 4\ 6\ \ldots\ n{-}4\ n\ n{-}2\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ 1) & \text{se } n \text{ é par} \\ (2\ 4\ 6\ \ldots\ n{-}5\ n{-}1\ n{-}3\ n\ n{-}2\ n{-}4\ \ldots\ 3\ 1) & \text{se } n \text{ é ímpar} \end{cases} \tag{5}$$

Lema 32. *Para $n \geq 4$, $d_{\rho\tau}(\pi_n^5) = \lceil \frac{n}{2} \rceil$.*

Teorema 33. *Para $n \geq 4$, $D_{\rho\tau}(n) \geq \lceil \frac{n}{2} \rceil$ e para $n \geq 9$, $D_{\rho\tau}(n) \leq \lfloor \frac{2n-2}{3} \rfloor$.*

Sabemos que $D_{\rho\tau}(n) = \lceil \frac{n}{2} \rceil$ para $4 \leq n \leq 13$ e que $d_{\rho\tau}(\pi_n^5) = \lceil \frac{n}{2} \rceil$ para $4 \leq n \leq 13$. Por esse motivo, acreditamos na conjectura a seguir.

Conjectura 1. *Para $n \geq 4$, $D_{\rho\tau}(n) = d_{\rho\tau}(\pi_n^5) = \lceil \frac{n}{2} \rceil$.*

6.2 Problema de Ordenação por Reversões de Prefixo

Mudando a forma como a permutação é percorrida, isto é, fazendo a busca por uma aresta boa de prefixo ser da direita para a esquerda, criamos um novo algoritmo, o qual será chamado `2-PRg`. Com exceção desta parte, ele funciona exatamente como `2-PR`, apresentado na Subseção 2.6. Inclusive continua tendo complexidade de tempo de $O(n^2)$.

6.3 Problema de Ordenação por Reversões de Prefixo e Transposições de Prefixo

Conforme explicado na Seção 2.9 sobre o `3-PRPT`, se uma aresta boa de prefixo (π_i, π_j) é do tipo 3 ou 4, a adjacência entre π_i e π_j é criada usando uma transposição de prefixo e isso garante remover pelo menos um *breakpoint*. No entanto, sabemos que uma transposição de prefixo é capaz de remover no máximo 2 *breakpoints*. Considerando isso, desenvolvemos uma versão gulosa do algoritmo `3-PRPT`, que chamaremos de `3-PRPTg`, cujas características são:

- Ele percorre a permutação da direita para a esquerda na busca pelas arestas boas;
- Ele tenta encontrar arestas azuis em uma ordem diferente: tipo 4, tipo 3, tipo 1 e então tipo 2;
- Quando existe uma aresta do tipo 4, ele tenta encontrar a melhor aresta vermelha presa procurando por uma aresta (π_{k-1}, π_k), $i < k < j$, tal que $\pi_k = \pi_{j+1} \pm 1$. Dessa forma é possível remover dois *breakpoints* quando a transposição $\tau_p(k, j+1)$ for aplicada;
- Quando ele está tentando encontrar uma aresta do tipo 3, ele procura por um π_j tal que $\pi_{j-1} = \pi_1 \pm 1$. Dessa forma é possível remover dois *breakpoints* quando $\tau_p(i+1, j)$ for aplicada.

Note que `3-PRPTg` continua tendo complexidade de tempo de $O(n^2)$.

A distância da família a seguir, apresentada pelo Lema 34, fornece um limitante inferior para o diâmetro do SbPRPT. O Teorema 35 apresenta limitantes inferior e superior para o diâmetro desse problema.

$$\pi_n^6 = \begin{cases} (n{-}1\ n{-}2\ n\ n{-}4\ n{-}6\ \ldots\ 2\ n{-}3\ n{-}5\ \ldots\ 1) & \text{se } n \text{ é par} \\ (n\ n{-}3\ n{-}1\ n{-}5\ n{-}7\ \ldots\ 2\ n{-}2\ n{-}4\ \ldots\ 1) & \text{se } n \text{ é ímpar} \end{cases} \tag{6}$$

Lema 34. *Para* $n \geq 7$, $\left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi_n^6) \leq \left\lceil \frac{n}{2} \right\rceil + 1$.

Teorema 35. *Para* $n \geq 7$, $D_{\rho_p\tau_p}(n) \geq \left\lceil \frac{n}{2} \right\rceil - 1$ *e para* $n \geq 1$, $D_{\rho_p\tau_p}(n) \leq n - \log_{\frac{9}{2}} n$.

6.4 Problema de Ordenação por Reversões de Prefixo com Sinal e Transposições de Prefixo

Outro problema para o qual ainda não existiam resultados conhecidos é o de Ordenação por Reversões de Prefixo com Sinal e Transposições de Prefixo (SbSigPRPT, de *Sorting by Signed Prefix Reversals and Prefix Transpositions*).

Nesse problema podemos considerar tanto *breakpoints* de reversão de prefixo com sinal quanto *breakpoints* de transposição de prefixo. Como ambos são equivalentes, utilizaremos a notação de *breakpoints* de reversão de prefixo com sinal, por convenção. O Lema 36 apresenta um limitante inferior para a distância desse problema.

Lema 36. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}\tau_p}(\pi) \geq \frac{b_{\bar{\rho_p}}(\pi)}{2}$.*

Desenvolvemos um algoritmo para o SbSigPRPT, que chamamos de `2-SPRPT`. Ele também tem uma ideia gulosa: tenta remover dois *breakpoints* com uma transposição de prefixo e, se não for possível, tanta remover um *breakpoint* com uma transposição de prefixo ou uma reversão de prefixo. Descrevemos a seguir como remover dois ou um *breakpoints*, respectivamente, para esse problema.

Seja $\pi \neq \iota_n$ e suponha que $\tau_p(i,j)$ é uma transposição de prefixo que remove dois *breakpoints* de π. Como $\pi \cdot \tau_p(i,j) = (\pi_i \ \ldots \ \pi_{j-1} \ \pi_1 \ \ldots \ \pi_{i-1} \ \ldots \ \pi_n)$, deve-se ter que $\pi_{j-1} = \pi_1 - 1$ e $\pi_{i-1} = \pi_j - 1$. Para manter o fator de aproximação desejado, não podemos ter $\pi_i = 1$.

Seja $\pi = (\underline{k+1 \ k+2 \ \ldots \ k+(i-1)} \ \pi_i \ \ldots\ldots)$ com $i \geq 2$ e $\pi_i \neq k+i$. Para remover um *breakpoint* com uma transposição de prefixo $\tau_p(i,j)$, podemos considerar aumentar a primeira *strip* de π. Isso resulta em duas possibilidades:

(1) se seu próximo elemento $\pi_j = k+i = \pi_{i-1}+1$ existe em π; ou

(2) se seu elemento anterior $\pi_{j-1} = k = \pi_1 - 1$ existe em π.

Finalmente, remover um *breakpoint* com uma reversão de prefixo é feito de forma idêntica ao SbSigPR: se existe $\pi_{j+1} = -\pi_1 + 1$, então $\bar{\rho_p}(j)$, para $1 \leq j \leq n$, é suficiente.

Lema 37. *Seja $\pi \neq \iota_n$ e $\pi_1 \neq 1$. Sempre é possível remover um breakpoint de π.*

Note que é sempre possível remover um *breakpoint* de uma dessas formas descritas acima, porque elas sempre conseguem aumentar a primeira strip com seu elemento anterior ou com o próximo, seja ele positivo ou negativo. A menos que $\pi = \iota_n$ ou $\pi_1 = 1$, tais elementos sempre irão existir.

Os três casos principais descritos acima são considerados se e somente se $\pi_1 \neq 1$. Sempre que $\pi_1 = 1$, o algoritmo deve enviar a primeira *strip* para o fim da permutação, de onde ela será removida apenas quando o elemento n for enviado para lá, como o Lema 38 mostra. Por causa disso, π_1 será 1 de novo no máximo uma outra vez, o que irá permitir que o algoritmo continue

removendo um ou dois *breakpoints* até o fim da ordenação, como os Lemas 39 e 40 mostram. O Teorema 42 mostra como esse comportamento garante o fator de aproximação assintótico 2 para esse problema.

Lema 38. *Seja $\pi \neq \iota_n$ uma permutação com sinal da forma $\pi = (\ldots\ldots\ \pi_{n-i}\ \underline{1\ 2\ \ldots\ i})$, com $1 \leq i < n$. A última strip de π será removida do fim apenas quando o elemento n for enviado para lá.*

Lema 39. *Seja $\pi \neq \iota_n$ uma permutação com sinal da forma $\pi = (\ldots\ldots\ n\ \underline{1\ 2\ \ldots\ i})$, com $1 \leq i < n$. Então, os elementos n e 1 não serão separados até que n esteja ordenada. Logo, é sempre possível continuar removendo um ou dois breakpoints.*

Lema 40. *Durante a ordenação, π_1 será 1 no máximo duas vezes.*

Lema 41. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}\tau_p}(\pi) \leq b_{\bar{\rho_p}}(\pi) + 2$.*

Teorema 42. SbSigPRPT *é 2-aproximável assintoticamente.*

A distância da família a seguir, apresentada no Lema 43, fornece um limitante inferior para o diâmetro do SbSigPRPT. O Teorema 44 apresenta limitantes inferior e superior para o diâmetro desse problema.

$$\begin{array}{c}\pi_n^7 = \big(+\lfloor\frac{n}{2}\rfloor\ +\big(\lfloor\frac{n}{2}\rfloor-1\big)\ +\big(\lfloor\frac{n}{2}\rfloor-2\big)\ \ldots\ +2\ +1\\ -\big(\lfloor\frac{n}{2}\rfloor+1\big)\ -\big(\lfloor\frac{n}{2}\rfloor+2\big)\ -\big(\lfloor\frac{n}{2}\rfloor+3\big)\ \ldots\ -(n-1)\ -n\big)\end{array} \tag{7}$$

Lema 43. *Para $n \geq 2$, $\lceil\frac{n}{2}\rceil + 1 \leq d_{\bar{\rho_p}\tau_p}(\pi_n^7) \leq n + 1$.*

Teorema 44. *Para $n \geq 2$, $D_{\bar{\rho_p}\tau_p}(n) \geq \lceil\frac{n}{2}\rceil + 1$ e para $n \geq 16$, $D_{\bar{\rho_p}\tau_p}(n) \leq \frac{18n}{11} + O(1)$.*

Como $D_{\bar{\rho_p}\tau_p}(n) = d_{\bar{\rho_p}\tau_p}(\pi_n^7)$ para $2 \leq n \leq 9$, e quando $n = 7$ as duas únicas permutações para as quais $d_{\bar{\rho_p}\tau_p}(\pi) = D_{\bar{\rho_p}\tau_p}(7) = 8$ são $\pi = \pi_7^7$ e $\pi = (+4\ +3\ +2\ +1\ -5\ -6\ -7)$, acreditamos que a seguinte conjectura é verdadeira.

Conjectura 2. *Para $n \geq 2$, $D_{\bar{\rho_p}\tau_p}(n) = d_{\bar{\rho_p}\tau_p}(\pi_n^7)$.*

6.5 Problema de Ordenação por Reversões de Prefixo e Reversões de Sufixo

O Problema de Ordenação por Reversões de Prefixo e Reversões de Sufixo (SbPRSR, de *Sorting by Prefix Reversals and Suffix Reversals*) não possuía algoritmos conhecidos e nem limitantes para a distância ou diâmetro. O Lema 45 apresenta um limitante trivial para a distância.

Lema 45. *Seja π uma permutação qualquer. Então, $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$.*

Com base no trabalho de Fischer e Ginzinger [25], estabelecemos o que fazer com cada tipo de aresta azul quando consideramos não apenas reversões de prefixo mas também reversões de sufixo. O Lema 46, uma versão estendida do Lema 8, mostra isso. As arestas descritas nesse lema são chamadas de *arestas boas*: quando elas são tratadas por operações de prefixo são chamadas *arestas boas de prefixo* (ABP) e quando elas são tratadas por operações de sufixo, são chamadas *arestas boas de sufixo* (ABS).

Lema 46. *Seja π uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo duas reversões de prefixo ou de sufixo que pode remover um breakpoint de reversão de prefixo e reversão de sufixo existe se $G_b(\pi)$ contém pelo menos uma das oito arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 1 com $j \leq n$;

(2) (π_i, π_n) do tipo 2 com $i \geq 1$;

(3) (π_1, π_j) do tipo 3 com $j \leq n$;

(4) (π_i, π_n) do tipo 3 com $i \geq 1$;

(5) (π_i, π_j) do tipo 2 com $i \neq 0$ e $j \leq n$;

(6) (π_i, π_j) do tipo 1 com $j \neq n+1$ e $i \geq 1$;

(7) (π_i, π_j) do tipo 3 com $i > 1$ e $j \leq n$;

(8) (π_i, π_j) do tipo 3 com $j < n$ e $i \geq 1$.

Primeiramente, modificamos os algoritmos `2-PR` e `2-PRg`, criando `2-SR` e `2-SRg`, respectivamente, para lidar com o Problema de Ordenação por Reversões de Sufixo (SbSR, de *Sorting by Suffix Reversals*). Ambos `2-SR` e `2-SRg` lidam apenas com arestas azuis (π_i, π_j) cuja adjacência entre π_i e π_j é criada usando reversões de sufixo, a saber, itens 2, 4, 6 e 8 do Lema 46 sem as restrições sobre i, as quais também chamaremos de *arestas boas de sufixo*. As restrições sobre i existem para que *breakpoints* de reversão de sufixo sejam consideradas.

Lema 47. *Seja π uma permutação qualquer. Então, $d_{\rho_s}(\pi) \geq b_{\rho_s}(\pi)$.*

`2-SR` e `2-SRg` também tentam escolher uma aresta boa de sufixo na ordem que os quatro tipos de arestas boas de sufixo (sem as restrições sobre i) aparecem no Lema 46. Quando elas não podem ser encontradas, a permutação é da forma dada em (8), conforme o Lema 50 mostra. A diferença entre os dois algoritmos é o fato de que `2-SR` percorre a permutação da direita para a esquerda durante sua busca pelas arestas azuis enquanto `2-SRg` faz isso da esquerda para a direita (conceitualmente, a mesma diferença entre `2-PR` e `2-PRg`). Se não for possível encontrar uma aresta boa de sufixo, os algoritmos aplicam a sequência de reversões de sufixo mostrada no Lema 51.

Lema 48. *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de sufixo. Então π não contém nenhum singleton.*

Lema 49. *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de sufixo. Então, a primeira aresta de π é do tipo 2, a última é do tipo 1 e todas as outras são do tipo 4.*

Lema 50. *Seja $\pi \neq \iota_n$ uma permutação qualquer que não contém uma aresta boa de sufixo. Então ela é da forma*

$$(\underline{1\ 2\ \ldots\ t}\ \underbrace{p_1\ \ldots\ t+1}_{\ell_1}\ \underbrace{p_2\ \ldots\ p_1+1}_{\ell_2}\ \ldots\ldots\ \underbrace{n\ \ldots\ p_{b_{\rho_s}(\pi)-1}+1}_{\ell_{b_{\rho_s}(\pi)}}) \tag{8}$$

onde $t \geq 0$. Em outras palavras, π consiste em $b_{\rho_s}(\pi) \geq 2$ strips decrescentes de tamanho $\ell_i \geq 2$, para todo $1 \leq i \leq b_{\rho_s}(\pi)$, de tal forma que se tais strips fossem crescentes, a permutação seria igual à identidade.

Lema 51. *Seja $\pi \neq \iota_n$ uma permutação qualquer de tamanho n sem arestas boas de sufixo que é da forma dada em (8). A sequência de $2b_{\rho_s}(\pi)$ reversões de sufixo*

$$\rho_s(t+1) \cdot \rho_s(t+\ell_{b_{\rho_s}(\pi)}+1) \cdot \rho_s(t+1) \cdot \rho_s(t+\ell_{b_{\rho_s}(\pi)-1}+1) \cdot \ldots \cdot \rho_s(t+1) \cdot \rho_s(t+\ell_1+1) \tag{9}$$

quando aplicada a π, transforma-a na permutação identidade.

`2-SR` e `2-SRg` possuem complexidade de tempo $O(n^2)$, uma vez que para encontrar qualquer tipo de aresta gasta-se um tempo $O(n)$ no pior caso, aplicar a reversão ou reversões correspondentes também é $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é $O(n)$. O Lema 52 e o Teorema 53 mostram que ambos algoritmos são de aproximação com fator 2.

Lema 52. *Seja π uma permutação qualquer. Então, $d_{\rho_s}(\pi) \leq 2b_{\rho_s}(\pi)$.*

Teorema 53. SbSR *é 2-aproximável.*

Finalmente, criamos dois algoritmos de 2-aproximação para o SbPRSR, os quais chamaremos de `2-PRSR` e `2-PRSRg`. De certa forma, esses algoritmos são baseados em uma intercalação dos passos de `2-PR` com `2-SR` e `2-PRg` com `2-SRg`, respectivamente. As diferenças estão nos limitantes dos índices das arestas boas, que garantem o uso de *breakpoints* de reversão de prefixo e reversão de sufixo. Além disso, as diferenças entre os novos algoritmos são equivalentes às descritas para `2-PR` e `2-PRg` ou `2-SR` e `2-SRg`: enquanto `2-PRSR` busca por uma aresta boa de prefixo da direita para a esquerda e busca por uma aresta boa de sufixo da esquerda para a direita, `2-PRSRg` faz o contrário.

Ambos `2-PRSR` e `2-PRSRg` procuram por qualquer uma das oito arestas boas listadas no Lema 46. Quando uma permutação não contém uma aresta boa, ela é caracterizada pelos Lemas 54 e 55. O Lema 56 mostra como lidar com tais permutações. É importante notar que quando ambas operações de reversão de prefixo e reversão de sufixo são permitidas, o número de operações necessárias para lidar com permutações que não possuem arestas boas pode ser reduzido praticamente pela metade com relação a `2-PR`, `2-PRg`, `2-SR` e `2-SRg`.

Lema 54. *Seja $\pi \neq \iota_n$ uma permutação qualquer que não contém uma aresta boa. Então, π não contém nenhum singleton.*

Lema 55. *Seja $\pi \neq \iota_n$ uma permutação qualquer que não contém uma aresta boa. Então, π é de uma das três formas:*

$$\eta_n$$
$$\sigma^1 = (\underbrace{p_1 \ \dots \ 1}_{\ell_1} \ \underbrace{p_2 \ \dots \ p_1+1}_{\ell_2} \ \dots\dots \ \underbrace{n \ \dots \ p_{b_{\rho_p\rho_s}(\pi)}+1}_{\ell_{b_{\rho_p\rho_s}(\pi)+1}})$$
$$\sigma^2 = (\underbrace{p_{b_{\rho_p\rho_s}(\pi)}+1 \ \dots \ n}_{\ell_{b_{\rho_p\rho_s}(\pi)+1}} \ \underbrace{p_{b_{\rho_p\rho_s}(\pi)-1}+1 \ \dots \ p_{b_{\rho_p\rho_s}(\pi)}}_{\ell_{b_{\rho_p\rho_s}(\pi)}} \ \dots\dots \ \underbrace{1 \ \dots \ p_1}_{\ell_1})$$
(10)

onde $b_{\rho_p\rho_s}(\pi) \geq 1$ e $\ell_i \geq 2$ para todo $1 \leq i \leq b_{\rho_p\rho_s}(\pi)+1$. Em outras palavras, ou π é a permutação reversa, ou é formada por $b_{\rho_p\rho_s}(\pi)+1$ strips decrescentes de tal forma que se fossem crescentes a permutação seria igual à identidade, ou então π é formada por $b_{\rho_p\rho_s}(\pi)+1$ strips crescentes de tal forma que se fossem decrescentes a permutação seria igual à reversa.

Lema 56. *Seja π uma permutação sem sinal de n elementos sem arestas boas de uma das formas dadas no Lema 55. Se $\pi = \eta_n$, uma reversão $\rho_p(n)$ a ordena. Caso contrário, no máximo $b_{\rho_p\rho_s}(\pi)+2$ movimentos ordenam π.*

A questão que surge quando permitimos mais de uma operação (no caso, reversões de prefixo e reversões de sufixo) é sobre qual aplicar. Decidimos apenas intercalar as escolhas. Ambos algoritmos possuem complexidade de tempo $O(n^2)$, uma vez que para encontrar qualquer tipo de aresta gasta-se um tempo $O(n)$ no pior caso, aplicar a operação correspondente também é da ordem de $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é da ordem de $O(n)$. O Lema 57 e o Teorema 58 mostram que o fator assintótico de aproximação de ambos algoritmos é 2.

Lema 57. *Seja π uma permutação qualquer. Então, $d_{\rho_p\rho_s}(\pi) \leq 2b_{\rho_p\rho_s}(\pi)+1$.*

Teorema 58. SbPRSR *é 2-aproximável assintoticamente.*

A distância da família a seguir, apresentada pelo Lema 59, fornece um limitante inferior para o diâmetro do SbPRSR. Com base nisso, o Teorema 60 apresenta limitantes inferior e superior para o diâmetro desse problema.

$$\pi_n^{11} = \begin{cases} (n \ 1 \ n{-}2 \ n{-}4 \ n{-}6 \ \dots \ 4 \ 2 \ n{-}3 \ n{-}5 \ n{-}7 \ \dots \ 3 \ n{-}1) & \text{se } n \text{ é par} \\ (n \ 1 \ n{-}2 \ n{-}4 \ n{-}6 \ \dots \ 5 \ 3 \ n{-}3 \ n{-}5 \ n{-}7 \ \dots \ 2 \ n{-}1) & \text{se } n \text{ é ímpar} \end{cases}$$
(11)

Lema 59. *Para $n \geq 8$, $n-1 \leq d_{\rho_p\rho_s}(\pi_n^{11}) \leq n$.*

Teorema 60. *Para $n \geq 8$, $D_{\rho_p\rho_s}(n) \geq n-1$ e para $n \geq 1$, $D_{\rho_p\rho_s}(n) \leq \frac{18n}{11} + O(1)$.*

Sabe-se que $D_{\rho_p\rho_s}(n) = n$ para $7 \leq n \leq 12$. Além disso, é possível validar que $d_{\rho_p\rho_s}(\pi_n^{11}) = n$ para $8 \leq n \leq 12$. Isso nos leva a acreditar na conjectura abaixo. Vale mencionar que quando $n = 7$, as únicas permutações para as quais $d_{\rho_p\rho_s}(\pi) = D_{\rho_p\rho_s}(7) = 7$ são $\pi = (7\ 3\ 5\ 2\ 6\ 4\ 1)$ e $\pi = (7\ 4\ 2\ 6\ 3\ 5\ 1)$.

Conjectura 3. *Para $n \geq 7$, $D_{\rho_p\rho_s}(n) = n$. Para $n \geq 8$, $D_{\rho_p\rho_s}(n) = d_{\rho_p\rho_s}(\pi_n^{11}) = n$.*

6.6 Problema de Ordenação por Reversões de Prefixo com Sinal e Reversões de Sufixo com Sinal

Para o Problema de Ordenação por Reversões de Prefixo com Sinal e Reversões de Sufixo com Sinal (SbSigPRSigSR, de *Sorting by Signed Prefix Reversals and Signed Suffix Reversals*), também não existiam resultados conhecidos. O Lema 61 apresenta um limitante inferior para a distância.

Lema 61. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho}_p\bar{\rho}_s}(\pi) \geq b_{\bar{\rho}_p\bar{\rho}_s}(\pi)$.*

Desenvolvemos um algoritmo para esse problema baseado na remoção gulosa de *breakpoints*, o qual chamamos de `2-SPRSSR`. A ideia desse algoritmo é de certa forma similar à ideia do `2-SPR`: ele tenta remover um *breakpoint* com uma operação e se isso não for possível, tenta remover um *breakpoint* com duas operações; se isso também não for possível, a permutação tem uma forma especial e existe uma sequência para ordená-la. A principal diferença é que não consideramos o "maior elemento positivo fora de ordem" simplesmente porque não faz sentido considerar elementos fora de ordem, uma vez que uma reversão de sufixo sempre os removeria de seu lugar. Assim, `2-SPRSSR` considera cinco casos principais:

(1) Se existe um $\pi_j = -\pi_1 + 1$ em π, $2 \leq j \leq n$, então a reversão $\bar{\rho}_p(j-1)$ remove um *breakpoint* de reversão de prefixo com sinal e reversão de sufixo com sinal;

(2) Se existe um $\pi_i = -\pi_n - 1$ em π, $1 \leq i \leq n-1$, então a reversão $\bar{\rho}_s(i+1)$ remove um *breakpoint*;

(3) Se existem π_i e π_j tais que:

   (i) $\pi_j = -\pi_i - 1$, $1 \leq i < j \leq n$, então $\bar{\rho}_p(j) \cdot \bar{\rho}_p(j-i)$ remove um *breakpoint*;

   (ii) $\pi_j = \pi_i + 1$, $0 \leq i+1 < j \leq n$, então $\bar{\rho}_p(i) \cdot \bar{\rho}_p(j-1)$ remove um *breakpoint*.

(4) Se existem π_i e π_j tais que:

   (i) $\pi_i = -\pi_j + 1$, $1 \leq i < j \leq n$, então $\bar{\rho}_s(i) \cdot \bar{\rho}_s(n+1-(j-i))$ remove um *breakpoint*;

   (ii) $\pi_i = \pi_j - 1$, $0 \leq i+1 < j \leq n$, então $\bar{\rho}_s(j) \cdot \bar{\rho}_s(i+1)$ remove um *breakpoint*. Esse item é equivalente ao item 3.(ii), portanto, apenas um deles precisa ser considerado.

(5) Se não for possível remover um *breakpoint* como descrito nos itens anteriores, então π é de uma das três formas descritas no Lema 62 e no máximo $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+2$ reversões são usadas para ordenar π, como descrito no Lema 63.

O Lema 64 e o Teorema 65 mostram que o fator assintótico de aproximação desse algoritmo é 2.

Lema 62. *Seja $\pi \neq \iota_n$ uma permutação com sinal na qual não se pode remover um breakpoint com uma ou duas operações. Então, π é de uma das três formas:*

$$\begin{aligned}
&\bar{\eta_n}\\
\sigma^1 &= (\underbrace{p_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)}+1 \ \dots \ n}_{\ell_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1}} \ \underbrace{p_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)-1}+1 \ \dots \ p_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)}}_{\ell_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)}} \ \dots\dots \ \underbrace{1 \ \dots \ p_1}_{\ell_1})\\
\sigma^2 &= (\underbrace{-p_1 \ \dots \ -1}_{\ell_1} \ \underbrace{-p_2 \ \dots \ -(p_1+1)}_{\ell_2} \ \dots\dots \ \underbrace{-n \ \dots \ -(p_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)}+1)}_{\ell_{b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1}}).
\end{aligned} \tag{12}$$

onde $\ell_i \geq 1$ para todo $1 \leq i \leq b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1$ e $b_{\bar{\rho}_p\bar{\rho}_s}(\pi) \geq 1$. Em outras palavras, ou π é a permutação reversa com sinal, ou é formada por $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1$ strips positivas de tal forma que se elas fossem negativas a permutação seria igual à reversa com sinal, ou então π é formada por $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1$ strips negativas de tal forma que se fossem positivas a permutação seria igual à identidade.

Lema 63. *Seja π uma permutação com sinal de n elementos de uma das formas descritas no Lema 62. Se $\pi = \bar{\eta_n}$, então uma reversão $\bar{\rho}_p(n)$ a ordena. Caso contrário, no máximo $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+2$ operações ordenam π.*

Lema 64. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho}_p\bar{\rho}_s}(\pi) \leq 2b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1$.*

Teorema 65. SbSigPRSigSR *é 2-aproximável assintoticamente.*

A distância da família a seguir, apresentada no Lema 66, fornece um limitante inferior para a distância do SbSigPRSigSR. O Teorema 67 apresenta limitantes inferior e superior para esse problema.

$$\pi_n^{13} = (-1^n n \ -1^{n-1}(n-1) \ -1^{n-2}(n-2) \ \dots \ +2 \ -1) \tag{13}$$

Lema 66. *Para $n \geq 5$, $n \leq d_{\bar{\rho}_p\bar{\rho}_s}(\pi_n^{13}) \leq n + \lfloor \frac{n-1}{2} \rfloor$.*

Teorema 67. *Para $n \geq 5$, $D_{\bar{\rho}_p\bar{\rho}_s}(n) \geq n$ e para $n \geq 16$, $D_{\bar{\rho}_p\bar{\rho}_s}(n) \leq 2n-6$.*

Sabe-se que $D_{\bar{\rho}_p\bar{\rho}_s}(n) = n + \lfloor \frac{n-1}{2} \rfloor$ para $5 \leq n \leq 9$ e que $d_{\bar{\rho}_p\bar{\rho}_s}(\pi_n^{13}) = n + \lfloor \frac{n-1}{2} \rfloor$ para $5 \leq n \leq 9$. Isso nos leva à seguinte conjectura.

Conjectura 4. *Para $n \geq 5$, $D_{\bar{\rho}_p\bar{\rho}_s}(n) = d_{\bar{\rho}_p\bar{\rho}_s}(\pi_n^{13}) = n + \lfloor \frac{n-1}{2} \rfloor$.*

6.7 Problema de Ordenação por Transposições de Prefixo e Transposições de Sufixo

Para o Problema de Ordenação por Transposições de Prefixo e Transposições de Sufixo (SbPTST, de *Sorting by Prefix Transpositions and Suffix Transpositions*) também não existiam algoritmos conhecidos nem limitantes para a distância ou diâmetro. O Lema 68 apresenta um limitante inferior para a distância.

Lema 68. *Seja π uma permutação qualquer. Então, $d_{\tau_p\tau_s}(\pi) \geq \frac{b_{\tau_p\tau_s}(\pi)}{2}$.*

Primeiramente, modificamos os algoritmos `2-PT` e `2-PTg`, criando `2-ST` e `2-STg`, respectivamente, para lidar com o Problema de Ordenação por Transposições de Sufixo (SbST, de *Sorting by Suffix Transpositions*). O que `2-ST` faz, assim como `2-PT`, é sempre remover um *breakpoint* usando uma transposição de sufixo, mas dessa vez aumentando a última *strip* da permutação ao colocar seu primeiro elemento i depois de seu elemento anterior $i-1$, conforme mostra o Lema 69.

Lema 69. *Seja $\pi \neq \iota_n$ uma permutação qualquer. Sempre é possível obter uma transposição de sufixo τ_s tal que $\Delta b_{\tau_s}(\pi, \tau_s) \leq -1$.*

Equivalente ao que ocorre com SbPT, para uma permutação qualquer existe no máximo uma transposição de sufixo que remove dois *breakpoints* de transposição de sufixo de uma vez, conforme o Lema 70 mostra. Assim, `2-STg` primeiro tenta encontrar tal transposição e se isso não for possível ele remove apenas um, da mesma forma que `2-ST` faz. O Lema 73 e o Teorema 74 mostram que o fator de aproximação de ambos `2-ST` e `2-STg` é 2.

Lema 70. *Seja π uma permutação qualquer. Existe no máximo uma transposição de sufixo τ_s tal que $\Delta b_{\tau_s}(\pi, \tau_s) = -2$.*

Lema 71. *Seja π uma permutação qualquer. Então, $d_{\tau_s}(\pi) \geq \frac{b_{\tau_s}(\pi)}{2}$.*

Lema 72. *Seja $\pi \neq \iota_n$ uma permutação qualquer e $\tau_s(j,k)$ uma transposição de sufixo tal que $\pi \cdot \tau_s = \iota_n$. Então $\pi_j = n$ e $\Delta b_{\tau_s}(\pi, \tau_s) = -2$.*

Lema 73. *Seja π uma permutação qualquer. Então, $d_{\tau_s}(\pi) \leq b_{\tau_s}(\pi) - 1$.*

Teorema 74. SbST *é 2-aproximável.*

Finalmente, criamos dois algoritmos para o SbPTST, os quais serão chamados de `2-PTST` e `2-PTSTg`. A diferença entre eles é que o primeiro sempre remove um *breakpoint* por vez, tentando aumentar o tamanho da primeira ou da última *strip*, podendo escolher aleatoriamente se faz isso com uma transposição de prefixo ou uma transposição de sufixo uma vez que ambas as formas são sempre possíveis. Por outro lado, `2-PTSTg` primeiro tenta remover dois *breakpoints*

de transposição de prefixo e transposição de sufixo e se isso não for possível, remove apenas um, da mesma forma que `2-PTST`. Os Lemas 75 e 76 mostram como remover um ou dois *breakpoints*, respectivamente, quando estamos lidando com o SbPTST.

Semelhante ao que ocorre com `2-PRSR` e `2-PRSRg`, os algoritmos `2-PTST` e `2-PTSTg` são quase uma união dos algoritmos `2-PT` com `2-ST` e `2-PTg` com `2-STg`, respectivamente, tendo como diferença as restrições sobre os índices das operações, para garantir que *breakpoints* de transposição de prefixo e transposição de sufixo sejam utilizados.

Lema 75. *Seja $\pi \neq \iota_n$ uma permutação qualquer. Sempre é possível obter uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_p) \leq -1$ e uma transposição de sufixo τ_s tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_s) \leq -1$.*

Lema 76. *Seja π uma permutação qualquer. Existe no máximo uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_p) = -2$ e existe no máximo uma transposição de sufixo τ_s tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_s) = -2$.*

Ambos `2-PTST` e `2-PTSTg` têm complexidade de tempo $O(n^2)$, uma vez que encontrar a primeira ou a última *strip* de uma permutação leva tempo $O(n)$, aplicar uma transposição também leva tempo $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é $O(n)$. O Lema 77 e o Teorema 78 mostram que ambos algoritmos possuem fator de aproximação 2.

Lema 77. *Seja π uma permutação qualquer. Então, $d_{\tau_p\tau_s}(\pi) \leq b_{\tau_p\tau_s}(\pi)$.*

Teorema 78. SbPTST *é 2-aproximável.*

A distância da família a seguir, apresentada pelo Lema 79, fornece um limitante inferior para o diâmetro do SbPTST. O Teorema 80 apresenta limitantes inferior e superior para o diâmetro desse problema.

$$\pi_n^{14} = \eta_n = (n\ n{-}1\ n{-}2\ \ldots\ 2\ 1) \tag{14}$$

Lema 79. *Para $n \geq 3$, $\left\lceil \frac{n-1}{2} \right\rceil + 1 \leq d_{\tau_p\tau_s}(\pi_n^{14}) \leq n - \left\lfloor \frac{n}{4} \right\rfloor$.*

Teorema 80. *Para $n \geq 3$, $\left\lceil \frac{n-1}{2} \right\rceil + 1 \leq D_{\tau_p\tau_s}(n) \leq n - \log_{\frac{9}{2}} n$.*

Sabemos que $D_{\tau_p\tau_s}(n) = d_{\tau_p\tau_s}(\pi_n^{14} = \eta_n)$ para $1 \leq n \leq 12$. Isso nos leva a acreditar na seguinte conjectura.

Conjectura 5. *Para $n \geq 1$, $D_{\tau_p\tau_s}(n) = d_{\tau_p\tau_s}(\eta_n)$.*

6.8 Problema de Ordenação por Reversões de Prefixo, Transposições de Prefixo, Reversões de Sufixo e Transposições de Sufixo

O Problema de Ordenação por Reversões de Prefixo, Transposições de Prefixo, Reversões de Sufixo e Transposições de Sufixo (SbPRPTSRST, de *Sorting by Prefix Reversals, Prefix Transpositions, Suffix Reversals and Suffix Transpositions*) também não possuía algoritmos conhecidos

nem limitantes para a distância ou diâmetro. Os Lemas 81 e 82 apresentam limitantes inferiores para a distância.

Nesse problema, podemos considerar *breakpoints* de reversão de prefixo e reversão de sufixo ou *breakpoints* de transposição de prefixo e transposição de sufixo. Além disso, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \min\{d_{\rho_p\rho_s}(\pi), d_{\tau_p\tau_s}(\pi)\}$, pois as sequências de ordenação obtidas para SbPRSR e SbPTST são válidas para o SbPRPTSRST, mas não necessariamente ótimas.

Lema 81. *Seja π uma permutação qualquer. Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \frac{b_{\rho_p\rho_s}(\pi)}{2}$.*

Dos Lemas 81 e 57 podemos ver que os algoritmos `2-PRSR` e `2-PRSRg` são algoritmos assintóticos de 4-aproximação para o SbPRPTSRST.

Lema 82. *Seja π uma permutação qualquer. Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \frac{b_{\tau_p\tau_s}(\pi)}{n}$.*

Dos Lemas 82 e 77 podemos ver que os algoritmos `2-PTST` e `2-PTSTg` não são algoritmos aproximativos para o SbPRPTSRST. Apesar disso, eles devolvem uma resposta válida para esse problema. Pela falta de um limitante inferior independente de n com os *breakpoints* de transposição de prefixo e transposição de sufixo, neste problema consideramos apenas os *breakpoints* de reversão de prefixo e reversão de sufixo.

Com base no trabalho de Sharmin *et al.* [40], é possível estabelecer o que fazer com cada tipo de aresta azul quando consideramos não apenas reversões de prefixo e transposições de prefixo mas também reversões de sufixo e transposições de sufixo. O Lema 83, uma versão estendida do Lema 29, mostra as arestas que são chamadas de *arestas boas*: quando elas são tratadas por operações de prefixo são chamadas *arestas boas de prefixo* (ABP) e quando elas são tratadas por operações de sufixo, são chamadas *arestas boas de sufixo* (ABS).

Lema 83. *Seja π uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo três reversões de prefixo, transposições de prefixo, reversões de sufixo ou transposições de sufixo que remove pelo menos um breakpoint de reversão de prefixo e reversão de sufixo existe se $G_b(\pi)$ contém pelo menos uma das oito arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 4 com $\pi_1 \neq 1$ e $j \leq n$;

(2) (π_i, π_n) do tipo 4 com $\pi_n \neq n$ e $i \geq 1$;

(3) (π_1, π_j) do tipo 1 com $\pi_1 \neq 1$ e $j \leq n$;

(4) (π_i, π_n) do tipo 2 com $\pi_n \neq n$ e $i \geq 1$;

(5) (π_i, π_j) do tipo 3 com $\pi_1 = 1$, $i \geq 1$ e $j \leq n$;

(6) (π_i, π_j) do tipo 3 com $\pi_n = n$, $i \geq 1$ e $j \leq n$;

(7) (π_i, π_j) do tipo 2 com $\pi_1 = 1$, $i \geq 1$ e $j \leq n$, onde π_i é o último elemento da primeira strip de π;

(8) (π_i, π_j) do tipo 1 com $\pi_n = n$, $i \geq 1$ e $j \leq n$, onde π_j é o primeiro elemento da última strip de π.

Primeiramente, fizemos versões de sufixo de ambos `3-PRPT` e `3-PRPTg`, que chamamos de `3-SRST` e `3-SRSTg`, respectivamente, para lidar com o Problema de Ordenação por Reversões de

Sufixo e Transposições de Sufixo (SbSRST, de *Sorting by Suffix Reversals and Suffix Transpositions*). Ambos lidam com arestas azuis (π_i, π_j) cuja adjacência entre π_i e π_j é criada usando reversões de sufixo ou transposições de sufixo apenas. A saber, itens 2, 4, 6 e 8 do Lema 83 sem as restrições sobre i, as quais também chamaremos de *arestas boas de sufixo*. As restrições sobre i são desconsideradas para que *breakpoints* de reversão de sufixo sejam considerados.

Lema 84. *Seja π uma permutação qualquer. Então, $d_{\rho_s\tau_s}(\pi) \geq \frac{b_{\rho_s}(\pi)}{2}$.*

`3-SRST` percorre a permutação da direita para a esquerda para tentar encontrar a primeira aresta boa de sufixo e então, na ordem em que as quatro aparecem no Lema 83, ele decide o tipo de tal aresta azul e as operações que ele deve realizar. O `3-SRSTg`, por outro lado, percorre a permutação da esquerda para a direita tentando encontrar arestas azuis em uma ordem diferente: tipo 4, tipo 3, tipo 2 e então tipo 1. Assim como `3-PRPTg`, `3-SRSTg` tenta encontrar arestas dos tipos 4 e 3 que são capazes de remover 2 *breakpoints* de uma vez. Quando existe uma aresta do tipo 4, ele tenta encontrar a melhor aresta vermelha presa (π_{k-1}, π_k), $i < k < j$, tal que $\pi_k = \pi_{j+1} \pm 1$. Quando está procurando por uma aresta do tipo 3, ele tenta encontrar um π_i tal que $\pi_{i+1} = \pi_n \pm 1$.

Lema 85. *Seja π uma permutação qualquer. Então, $d_{\rho_s\tau_s}(\pi) \leq \frac{3b_{\rho_s}(\pi)}{2}$.*

Teorema 86. SbSRST *é 3-aproximável.*

Finalmente, criamos algoritmos para o SbPRPTSRST, que de certa forma são baseados em uma intercalação dos passos de `3-PRPT` com `3-SRST` e `3-PRPTg` com `3-SRSTg`. Tais algoritmos foram chamados `2-PRPTSRST` e `2-PRPTSRSTg`. As diferenças estão nos limitantes dos índices das arestas boas, que garantem o uso de *breakpoints* de reversão de prefixo e reversão de sufixo.

Os novos algoritmos `2-PRPTSRST` e `2-PRPTSRSTg` podem lidar com todas as arestas boas descritas no Lema 83. No entanto, eles não consideram as arestas descritas nos itens 7 e 8, pois quando as outras seis arestas não existem, π é de uma das formas mostradas pelo Lema 87 e os algoritmos fazem uma reversão de prefixo $\rho_p(n)$ ou uma transposição de prefixo que concatena a primeira *strip* com a última. Por esse motivo, eles nunca podem separar os elementos n e 1, a menos que a aresta vermelha entre eles seja a última da permutação (desconsiderando as arestas (π_0, π_1) e (π_n, π_{n+1})). O Lema 91 e o Teorema 92 mostram como esse comportamento garante o fator de aproximação assintótico 2 para ambos algoritmos.

A diferença entre os dois algoritmos é que `2-PRPTSRST` percorre a permutação da esquerda para a direita quando procura por arestas boas de prefixo e da direita para a esquerda quando procura por arestas boas de sufixo. Ele segue a ordem definida pelo Lema 83. Por sua vez, `2-PRPTSRSTg` faz o contrário, percorrendo a permutação da direita para a esquerda quando procura por arestas boas de prefixo e da esquerda para a direita quando procura por arestas boas de sufixo. Além disso, ele segue a mesma ordem de arestas que `3-PRPTg` e `3-SRSTg` seguem, de maneira intercalada, e também tenta encontrar arestas dos tipos 4 e 3 que são capazes de

remover 2 *breakpoints* de uma vez. Quando existe uma aresta do tipo 4, ele também tenta encontrar a melhor aresta vermelha presa procurando por uma aresta (π_{k-1}, π_k), $i < k < j$, tal que $\pi_k = \pi_{j+1} \pm 1$. Dessa vez, se a aresta é de prefixo, deve-se garantir que $j \leq n-1$ e se ela é de sufixo, deve-se garantir que $i \geq 2$. Com isso, é possível remover dois *breakpoints* de reversão de prefixo e reversão de sufixo quando as transposições $\tau_p(k, j+1)$ ou $\tau_s(k, j+1)$ forem aplicadas. A busca por arestas boas do tipo 3 é da mesma forma: se a aresta é de prefixo, ele procura por um π_j tal que $\pi_{j-1} = \pi_1 \pm 1$ e se a aresta é de sufixo, ele procura por um π_i tal que $\pi_{i+1} = \pi_n \pm 1$.

Lema 87. *Seja $\pi \neq \iota_n$ uma permutação na qual não existem as seis primeiras arestas boas do Lema 83, que garantem remover um breakpoint com um único movimento. Então π é de uma das três formas:*

$$\eta_n \tag{15}$$

ou

$$\sigma^1 = (\underline{1\ 2\ \ldots\ k} \ldots\ldots \underline{k{+}i\ \ldots\ k{+}1} \ldots\ldots \underline{j{-}1\ \ldots\ j{-}\ell} \ldots\ldots \underline{j\ j{+}1\ \ldots\ n}) \tag{16}$$

sendo que (i) $i \geq 2$ se $k \geq 1$ ou $i \geq 1$ se $k \geq 2$, (ii) $\ell \geq 2$ se $j \geq 1$ ou $\ell \geq 1$ se $j \geq 2$, e (iii) a ordem relativa entre as strips que contém $k+i$ e $j-\ell$ é irrelevante. Ou então

$$\sigma^2 = (\underline{n\ n-1\ \ldots\ j}\ \pi_{n-j+2}\ \ldots\ldots\ \pi_{n-k}\ \underline{k\ k-1\ \ldots\ 1}) \tag{17}$$

com $\pi_{n-j+2} \neq j-1$, $\pi_{n-k} \neq k+1$, $j < n$ e $k > 1$.

Lema 88. *Seja $\pi \neq \iota_n$ uma permutação de uma das formas dadas no Lema 87. Se $\pi = \eta_n$, uma reversão $\rho_p(n)$ a ordena. Caso contrário, uma transposição $\tau_p(i+1, n+1)$, onde π_i é o último elemento da primeira strip, concatena a primeira strip com a última sem alterar o número de breakpoints. Após ela, sempre é possível continuar removendo pelo menos um breakpoint com uma única operação.*

Lema 89. *Seja (π_i, π_j) uma aresta azul do tipo 4 com $i = 1$ ou com $j = n$ existente em uma permutação $\pi \neq \iota_n$ qualquer. Se a aresta vermelha entre os elementos 1 e n for a única aresta presa entre π_i e π_j, então na verdade $i = 1$, $j = n$ e a permutação é de uma das duas formas:*

$$\begin{aligned} \pi' &= (\underline{k\ k-1\ k-2\ \ldots\ 2\ 1}\ \underline{n\ n-1\ \ldots\ k+2\ k+1}) \\ \pi'' &= (\underline{k+1\ k+2\ \ldots\ n-1\ n}\ \underline{1\ 2\ \ldots\ k-2\ k-1\ k}) \end{aligned} \tag{18}$$

Além disso, ao agir em tal aresta os algoritmos estarão realizando sua última ou penúltima operação.

Lema 90. *A operação explicada no Lema 88 é realizada no máximo uma vez pelos algoritmos `2-PRPTSRST` e `2-PRPTSRSTg`.*

Lema 91. *Seja π uma permutação qualquer. Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq b_{\rho_p\rho_s}(\pi) + 2$.*

Teorema 92. SbPRPTSRST *é 2-aproximável assintoticamente.*

A complexidade de tempo de ambos algoritmos `2-PRPTSRST` e `2-PRPTSRSTg` também é $O(n^2)$, uma vez que para encontrar qualquer tipo de aresta gasta-se um tempo $O(n)$ no pior caso, aplicar a operação correspondente também é da ordem de $O(n)$, e a distância é proporcional ao número de *breakpoints*, que é $O(n)$.

A distância da família a seguir, apresentada pelo Lema 93, fornece um limitante inferior para o diâmetro de SbPRPTSRST. O Teorema 94 apresenta limitantes para o diâmetro do problema.

$$\pi_n^{19} = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 5\ 3\ 6\ 8\ 10\ \ldots\ n\ 2\ 4\ 1) & \text{se } n \text{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 5\ 3\ 6\ 8\ 10\ \ldots\ n{-}1\ 2\ 4\ 1) & \text{se } n \text{ é ímpar} \end{cases} \tag{19}$$

Lema 93. *Para* $n \geq 6$, $\left\lceil \frac{n}{2} \right\rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) \leq \left\lceil \frac{n}{2} \right\rceil + 1$ *se* n *é par e* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) = \left\lceil \frac{n}{2} \right\rceil + 1$ *se* n *é ímpar.*

Teorema 94. *Para* $n \geq 6$, $D_{\rho_p\tau_p\rho_s\tau_s}(n) \geq \left\lceil \frac{n}{2} \right\rceil$ *e para* $n \geq 1$, $D_{\rho_p\tau_p\rho_s\tau_s}(n) \leq n - \log_{\frac{9}{2}} n$.

Sabemos que $D_{\rho_p\tau_p\rho_s\tau_s}(n) = \left\lceil \frac{n}{2} \right\rceil + 1$ para $6 \leq n \leq 12$ e que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) = \left\lceil \frac{n}{2} \right\rceil + 1$ para $6 \leq n \leq 12$. Isso nos leva à acreditar na seguinte conjectura.

Conjectura 6. *Para* $n \geq 6$, $D_{\rho_p\tau_p\rho_s\tau_s}(n) = d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) = \left\lceil \frac{n}{2} \right\rceil + 1$.

6.9 Problema de Ordenação por Reversões de Prefixo com Sinal, Transposições de Prefixo, Reversões de Sufixo com Sinal e Transposições de Sufixo

Também desenvolvemos um algoritmo para o problema de Ordenação por Reversões de Prefixo com Sinal, Transposições de Prefixo, Reversões de Sufixo com Sinal e Transposições de Sufixo (SbSigPRPTSigSRST), o qual chamamos de `2-SPRPTSSRST`. Outros resultados para esse problema também não eram conhecidos.

No SbSigPRPTSigSRST, podemos considerar tanto *breakpoints* de reversão de prefixo com sinal e reversão de sufixo com sinal quanto *breakpoints* de transposição de prefixo e transposição de sufixo. Como ambos são equivalentes, utilizaremos a notação do primeiro tipo, por convenção. O Lema 95 apresenta um limitante inferior para a distância.

Lema 95. *Seja* π *uma permutação com sinal qualquer. Então,* $d_{\bar{\rho_p}\tau_p\bar{\rho_s}\tau_s}(\pi) \geq \frac{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}{2}$.

`2-SPRPTSSRST` segue a mesma ideia gulosa de `2-SPRPT` com a mesma restrição com relação à separação dos elementos n e 1, mas ele também não permite que -1 e $-n$ sejam separados quando eles estão juntos nessa ordem. Assim, os principais passos deste algoritmo são: (i) tentar remover dois *breakpoints* com uma transposição de prefixo ou de sufixo; (ii) tentar remover um

breakpoint com uma transposição de prefixo ou de sufixo; e (iii) tentar remover um *breakpoint* com uma reversão de prefixo ou de sufixo.

Para remover dois *breakpoints* de reversão de prefixo com sinal e reversão de sufixo com sinal com uma transposição de prefixo $\tau_p(i,j)$, também devemos encontrar $\pi_{j-1} = \pi_1 - 1$ e $\pi_{i-1} = \pi_j - 1$, mas dessa vez considerando que $2 \leq i < j \leq n$. Para remover dois *breakpoints* desse tipo com uma transposição de sufixo $\tau_s(i,j)$, como $\pi \cdot \tau_s(i,j) = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_j \ \ldots \ \pi_n \ \pi_i \ \ldots \ \pi_{j-1})$, deve-se ter que $\pi_i = \pi_n + 1$ e $\pi_j = \pi_{i-1} + 1$ também considerando que $2 \leq i < j \leq n$. Nem $[\pi_{i-1}, \pi_i]$ nem $[\pi_{j-1}, \pi_i]$ podem ser $[-1, -n]$ ou $[n, 1]$.

A remoção de um *breakpoint* de reversão de prefixo com sinal e reversão de sufixo com sinal com uma transposição de prefixo $\tau_p(i,j)$ é quase similar ao que foi explicado para SbSigPRPT: ela pode ser feita aumentando a primeira *strip*, o que resulta em duas possibilidades. Seja $\pi = (\underline{k+1 \ k+2 \ \ldots \ k+(i-1)} \ \pi_i \ \ldots\ldots)$ com $i \geq 2$ e $\pi_i \neq k + i$:

(1) se o próximo elemento dessa *strip* $\pi_j = k + i = \pi_{i-1} + 1$, com $j \leq n$, existe em π; ou

(2) se seu elemento anterior $\pi_{j-1} = k = \pi_1 - 1$ existe em π.

Nem $[\pi_{i-1}, \pi_i]$ nem $[\pi_{j-1}, \pi_i]$ podem ser $[-1, -n]$ ou $[n, 1]$.

Seja $\pi = (\ldots\ldots \ \pi_{j-1} \ \underline{k+1 \ k+2 \ \ldots \ k+x})$ com $x \geq 1$, $j \leq n$ e $\pi_{j-1} \neq k$. Também existem duas possibilidades para aumentar a última *strip* e remover um *breakpoint* com uma transposição de sufixo $\tau_s(i,j)$:

(1) se seu elemento anterior $\pi_{i-1} = \pi_j - 1 = k$, com $i \geq 2$, existe em π; ou

(2) se seu próximo elemento $\pi_i = \pi_n + 1 = k + x + 1$ existe em π.

Novamente, nem $[\pi_{i-1}, \pi_i]$ nem $[\pi_{j-1}, \pi_i]$ podem ser $[-1, -n]$ ou $[n, 1]$.

Finalmente, para remover um *breakpoint* com uma reversão de prefixo $\bar{\rho}_p(j)$, é suficiente encontrar um $\pi_{j+1} = -\pi_1 + 1$ onde $1 \leq j \leq n - 1$. Nesse caso, $[\pi_j, \pi_{j+1}]$ não pode ser $[-1, -n]$ nem $[n, 1]$. Para remover um *breakpoint* com uma reversão de sufixo $\bar{\rho}_s(i)$, deve-se encontrar um $\pi_{i-1} = -\pi_n - 1$, com $2 \leq i \leq n$. Nesse caso, $[\pi_{i-1}, \pi_i]$ não pode ser $[-1, -n]$ nem $[n, 1]$.

Quando nenhuma das opções acima está disponível, a permutação é de uma das formas descritas no Lema 96 e devemos fazer uma reversão para ordenar $\bar{\eta}_n$ ou uma transposição de prefixo para concatenar a primeira *strip* com a última. Essa última operação é a razão pela qual o algoritmo não pode separar os elementos n e 1 ou -1 e $-n$, o que vai garantir o fator de aproximação do algoritmo, conforme o Teorema 99 mostra.

Lema 96. *Seja $\pi \neq \iota_n$ uma permutação com sinal para a qual não é possível remover um ou dois breakpoints com uma operação. Então π é de uma das cinco formas:*

$$\bar{\eta}_n \tag{20}$$

ou

$$\mu^1 = (\underline{1\ 2\ \ldots\ k}\ \ldots\ -(k+1)\ \ldots\ -(i-1)\ \ldots\ \underline{i\ i+1\ \ldots\ n}) \tag{21}$$

isto é, $\pi_1 = 1$, $\pi_n = n$, *e os elementos que podem aumentar a primeira e a última strips são negativos (a posição relativa entre* $-(k+1)$ *e* $-(i-1)$ *é irrelevante). Ou*

$$\mu^2 = (\underline{-n\ -(n-1)\ldots-i}\ \ldots\ (i-1)\ \ldots\ (k+1)\ \ldots\ \underline{-k\ -(k-1)\ldots-1}) \tag{22}$$

isto é, $\pi_1 = -n$, $\pi_n = -1$, *e os elementos que podem aumentar a primeira e a última strips são positivos (a posição relativa entre* $i-1$ *e* $k+1$ *é irrelevante). Ou*

$$\mu^3 = (\underline{k+1\ k+2\ \ldots\ n}\ \underline{1\ 2\ \ldots\ k}) \tag{23}$$

ou então

$$\mu^4 = (\underline{-k\ -(k-1)\ \ldots\ -1}\ \underline{-n\ -(n-1)\ \ldots\ -(k+1)}). \tag{24}$$

Em todos os casos, $1 \leq k$ *e* $k+1 < i \leq n$.

Lema 97. *Seja* π *uma permutação com sinal conforme descrito no Lema 96 tal que* $\pi \neq \bar{\eta}_n$. *Então, uma transposição que concatena a primeira strip com a última não cria novos breakpoints e garante que as próximas operações sempre conseguirão remover pelo menos um breakpoint.*

Lema 98. *Seja* π *uma permutação qualquer. Então,* $d_{\bar{\rho}_p\tau_p\bar{\rho}_s\tau_s}(\pi) \leq b_{\bar{\rho}_p\bar{\rho}_s}(\pi) + 2$.

Teorema 99. SbSigPRPTSigSRST *é 2-aproximável assintoticamente.*

6.10 Melhorando os Resultados na Prática

Todos os algoritmos que foram descritos até agora têm um comportamento similar: quando se está tentando remover uma certa quantidade de *breakpoints*, busca-se pela primeira oportunidade de fazê-lo. Por exemplo, considere o `2-PRSR`. O primeiro passo é tentar remover um *breakpoint* com uma reversão de prefixo e, em seguida, com uma reversão de sufixo. Assim, mesmo se ambas as formas de remoção forem possíveis, o algoritmo sempre usará uma reversão de prefixo. O mesmo ocorre com o `2-PR` e as várias formas de remover um *breakpoint* com duas reversões. Todos os outros algoritmos apresentados possuem, de alguma forma, essa característica.

Sendo assim, desenvolvemos novos algoritmos, que em teoria possuem o mesmo fator de aproximação que os algoritmos anteriores, mas na prática terão a chance de encontrar resultados melhores. Esses novos algoritmos continuam sendo determinísticos. Além disso, eles utilizam os algoritmos de aproximação já descritos para auxiliá-los na decisão de qual operação realizar.

De forma geral, cada novo algoritmo procura por todas as operações que podem remover o maior número possível de *breakpoints*. Cada uma dessas operações é aplicada sobre a permutação π atual e, sobre cada permutação resultante π', o algoritmo de aproximação adequado é aplicado

para que se obtenha um limitante superior na distância de π'. Escolhe-se, então, a permutação π' cujo limitante superior da distância seja o menor para ser a próxima permutação da ordenação. Foram criados algoritmos para os problemas SbPR, SbPRSR, SbSigPR, SbSigPRSigSR, SbPT, SbPTST, SbPRPT, SbPRPTSRST, SbSigPRPT e SbSigPRPTSigSRST, e eles foram chamados, respectivamente, de `2-PRx`, `2-PRSRx`, `2-SPRx`, `2-SPRSSRx`, `2-PTx`, `2-PTSTx`, `3-PRPTx`, `2-PRPTSRSTx`, `2-SPRPTx` e `2-SPRPTSSRSTx`.

Vale notar que, se existir apenas uma operação que remova a quantidade máxima de *breakpoints*, não há necessidade de utilizar o resultado do algoritmo de aproximação. De qualquer forma, todos esses novos algoritmos possuem complexidade de tempo $O(n^4)$: a distância é $O(n)$, encontrar as operações desejadas é $O(n)$ e os algoritmos de aproximação utilizados são $O(n^2)$.

Todos os algoritmos apresentados nesta seção de resultados parciais foram implementados e os resultados obtidos encontram-se no Apêndice B.

Referências Bibliográficas

[1] David A. Bader, Bernard M. E. Moret, e Mi Yan. A Linear-Time Algorithm for Computing Inversion Distance Between Signed Permutations with an Experimental Study. *Journal of Computational Biology*, 8: 483–491, 2001.

[2] Martin Bader e Enno Ohlebusch. Sorting by Weighted Reversals, Transpositions, and Inverted Transpositions. In Alberto Apostolico, Concettina Guerra, Sorin Istrail, Pavel A. Pevzner, e Michael Waterman, editors, *Research in Computational Molecular Biology*, volume 3909 of *Lecture Notes in Computer Science*, pages 563–577. Springer Berlin Heidelberg, 2006.

[3] Vineet Bafna e Pavel A. Pevzner. Genome Rearrangements and Sorting by Reversals. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS'1993)*, pages 148–157, 1993.

[4] Vineet Bafna e Pavel A. Pevzner. Sorting by Transpositions. *SIAM Journal on Discrete Mathematics*, 11 (2):224–240, 1998.

[5] Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y. Pinter, Steven S. Skiena, e Firas Swidan. Improved Bounds on Sorting by Length-Weighted Reversals. *Journal of Computer and System Science*, 74 (5):744–774, 2008.

[6] Anne Bergeron. A Very Elementary Presentation of the Hannenhalli-Pevzner Theory. *Discrete Applied Mathematics*, 146(2):134–145, 2005.

[7] Piotr Berman e Marek Karpinski. On Some Tighter Inapproximability Results (Extended Abstract). In Jiří Wiedermann, Peter Emde Boas, e Mogens Nielsen, editors, *Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer Berlin Heidelberg, 1999.

[8] Piotr Berman, Sridhar Hannenhalli, e Marek Karpinski. 1.375-Approximation Algorithm for Sorting by Reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'2002)*, pages 200–210, London, UK, 2002. Springer-Verlag.

[9] Mathieu Blanchette, Takashi Kunisawa, e David Sankoff. Parametric Genome Rearrangement. *Gene*, 172 (1):GC11–GC17, 1996.

[10] Laurent Bulteau, Guillaume Fertin, e Irena Rusu. Pancake Flipping Is Hard. In *Mathematical Foundations of Computer Science 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 247–258. Springer Berlin Heidelberg, 2012.

[11] Laurent Bulteau, Guillaume Fertin, e Irena Rusu. Sorting by Transpositions is Difficult. *SIAM Journal on Computing*, 26(3):1148–1180, 2012.

[12] Alberto Caprara. Sorting Permutations by Reversals and Eulerian Cycle Decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[13] Xin Chen. On Sorting Unsigned Permutations by Double-Cut-and-Joins. *Journal of Combinatorial Optimization*, 25(3):339–351, 2013.

[14] Bhadrachalam Chitturi e Ivan Hal Sudborough. Bounding Prefix Transposition Distance for Strings and Permutations. *Theoretical Computer Science*, 421:15–24, 2012.

[15] Bhadrachalam Chitturi, William Fahle, Z. Meng, Linda Morales, Charles O. Shields, Ivan Hal Sudborough, e Walter Voit. An $(18/11)n$ Upper Bound for Sorting by Prefix Reversals. *Theoretical Computer Science*, 410(36):3372–3390, 2009.

[16] David Alan Christie. A 3/2-Approximation Algorithm for Sorting by Reversals. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'1998)*, pages 244–252, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[17] Josef Cibulka. On Average and Highest Number of Flips in Pancake Sorting. *Theoretical Computer Science*, 412(8-10):822–834, 2011.

[18] David S. Cohen e Manuel Blum. On the Problem of Sorting Burnt Pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995.

[19] Zanoni Dias e João Meidanis. Sorting by Prefix Transpositions. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE'2002)*, pages 65–76, London, UK, 2002. Springer-Verlag.

[20] Harry Dweighter. Problem E2569. *American Mathematical Monthly*, 82:1010, 1975.

[21] Isaac Elias e Tzvika Hartman. A 1.375-Approximation Algorithm for Sorting by Transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

[22] Niklas Eriksen. $(1+\epsilon)$-Approximation of Sorting by Reversals and Transpositions. *Theoretical Computer Science*, 289(1):517–529, 2002.

[23] Henrik Eriksson, Kimmo Eriksson, Johan Karlander, Lars Svensson, e Johan Wastlund. Sorting a Bridge Hand. *Discrete Mathematics*, 241(1-3):289–300, 2001.

[24] Guillaume Fertin, Anthony Labarre, Irena Rusu, Éric Tannier, e Stéphane Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. The MIT Press, 2009.

[25] Johannes Fischer e Simon W. Ginzinger. A 2-Approximation Algorithm for Sorting by Prefix Reversals. In *Proceedings of the 13th Annual European Conference on Algorithms (ESA'2005)*, pages 415–425, Berlin, Heidelberg, 2005. Springer-Verlag.

[26] Gustavo Rodrigues Galvão. Uma Ferramenta de Auditoria para Algoritmos de Rearranjo de Genomas. Master's thesis, University of Campinas, Institute of Computing, Campinas, São Paulo, Brazil, 2012. In Portuguese.

[27] Gustavo Rodrigues Galvão e Zanoni Dias. Computing Rearrangement Distance of Every Permutation in the Symmetric Group. In William C. Chu, W. Eric Wong, Mathew J. Palakal, e Chih-Cheng Hung, editors, *Proceedings of the 26th ACM Symposium on Applied Computing (SAC'2011)*, pages 106–107. ACM, 2011.

[28] William H. Gates e Christos H. Papadimitriou. Bounds for Sorting by Prefix Reversal. *Discrete Mathematics*, 27(1):47–57, 1979.

[29] Qian-Ping Gu, Shietung Peng, e Ivan Hal Sudborough. A 2-Approximation Algorithm for Genome Rearrangements by Reversals and Transpositions. *Theoretical Computer Science*, 210(2):327–339, 1999.

[30] Sridhar Hannenhalli e Pavel A. Pevzner. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *Journal of the ACM*, 46(1):1–27, 1999.

[31] Tzvika Hartman e Roded Sharan. A 1.5-Approximation Algorithm for Sorting by Transpositions and Transreversals. *Journal of Computer and System Sciences*, 70(3):300–320, 2005.

[32] Mohammad H. Heydari e Ivan Hal Sudborough. On the Diameter of the Pancake Network. *Journal of Algorithms*, 25(1):67–94, 1997.

[33] John Kececioglu e David Sankoff. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, 13:180–210, 1995.

[34] Anthony Labarre. Edit Distances and Factorisations of Even Permutations. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'2008)*, pages 635–646, Berlin, Heidelberg, 2008. Springer-Verlag.

[35] Guo-Hui Lin e Guoliang Xue. Signed Genome Rearrangement by Reversals and Transpositions: Models and Approximations. *Theoretical Computer Science*, 259(1-2):513–531, 2001.

[36] Guohui Lin e Tao Jiang. A Further Improved Approximation Algorithm for Breakpoint Graph Decomposition. *Journal of Combinatorial Optimization*, 8(2):183–194, 2004.

[37] Pavel A. Pevzner e Michael S. Waterman. Open Combinatorial Problems in Computational Molecular Biology. In *Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems (ISTCS'1995)*, pages 158–173, Washington, DC, USA, 1995. IEEE Computer Society.

[38] Ron Y. Pinter e Steven Skiena. Genomic Sorting with Length-Weighted Reversals. *Genome Informatics*, 13:2002, 2002.

[39] Atif Rahman, Swakkhar Shatabda, e Masud Hasan. An Approximation Algorithm for Sorting by Reversals and Transpositions. *Journal of Discrete Algorithms*, 6(3):449–457, 2008.

[40] Mahfuza Sharmin, Rukhsana Yeasmin, Masud Hasan, Atif Rahman, e Mohammad Sohel Rahman. Pancake Flipping with Two Spatulas. *Electronic Notes in Discrete Mathematics*, 36:231–238, 2010.

[41] Firas Swidan, Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, e Ron Y. Pinter. Sorting by Length-Weighted Reversals: Dealing with Signs and Circularity. In SuleymanCenk Sahinalp, S. Muthukrishnan, e Ugur Dogrusoz, editors, *Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin Heidelberg, 2004.

[42] Eric Tannier, Anne Bergeron, e Marie-France Sagot. Advances on Sorting by Reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.

[43] Maria Emilia Machado Telles Walter, Zanoni Dias, e João Meidanis. Reversal and Transposition Distance of Linear Chromosomes. In *Proceedings of the 5th International Symposium on String Processing and Information Retrieval (SPIRE'1998)*, pages 96–102, Santa Cruz, Bolivia, 1998. IEEE Computer Society.

A Demonstrações e Algoritmos Omitidos

A.1 Problema de Ordenação por Reversões

Lema 1. [3] *Seja π uma permutação sem sinal qualquer. Então, $d_\rho(\pi) \geq \frac{b_\rho(\pi)}{2}$.*

Demonstração. Uma reversão $\rho(i,j)$ separa os pares de elementos $[\pi_{i-1}, \pi_i]$ e $[\pi_j, \pi_{j+1}]$, podendo criar ou remover *breakpoints* nos dois pares. Logo, $\Delta b_\rho(\pi, \rho) \in \{-2, -1, 0, 1, 2\}$. □

Lema 2. [33] *Seja π uma permutação sem sinal qualquer. Então, $d_\rho(\pi) \leq b_\rho(\pi) - 1$.*

Demonstração. Kececioglu e Sankoff [33] apresentaram um algoritmo que ordena qualquer permutação em no máximo $b_\rho(\pi) - 1$ movimentos. □

A.2 Problema de Ordenação por Transposições

Lema 3. [4] *Seja π uma permutação sem sinal qualquer. Então, $d_\tau(\pi) \geq \frac{b_\tau(\pi)}{3}$.*

Demonstração. Uma transposição $\tau(i,j,k)$ separa os pares $[\pi_{i-1}, \pi_i]$, $[\pi_{j-1}, \pi_j]$ e $[\pi_{k-1}, \pi_k]$, podendo criar ou remover *breakpoints* em todos eles. Assim, $\Delta b_\tau(\pi, \tau) \in \{-3, -2, -1, 0, 1, 2, 3\}$. □

A.3 Problema de Ordenação por Reversões e Transposições

Lema 4. [43] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho\tau}(\pi) \geq \frac{b_\rho(\pi)}{3}$.*

Demonstração. Uma reversão $\rho(i,j)$ separa os pares $[\pi_{i-1}, \pi_i]$ e $[\pi_j, \pi_{j+1}]$, podendo criar ou remover *breakpoints* em qualquer um deles. Logo, $\Delta b_\rho(\pi, \rho) \in \{-2, -1, 0, 1, 2\}$. Uma transposição $\tau(i,j,k)$ separa $[\pi_{i-1}, \pi_i]$, $[\pi_{j-1}, \pi_j]$ e $[\pi_{k-1}, \pi_k]$, podendo criar ou remover *breakpoints* em qualquer um dos pares. Logo, $\Delta b_\rho(\pi, \tau) \in \{-3, -2, -1, 0, 1, 2, 3\}$. □

Lema 5. [43] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho\tau}(\pi) \leq b_\rho(\pi)$.*

Demonstração. É sempre possível remover pelo menos um *breakpoint* de uma permutação qualquer aumentando a primeira *strip* em cada operação. Seja $\pi_i = \ell$ o último elemento da primeira *strip*. Se $\pi_j = \ell+1$ é um *singleton* ou é o primeiro elemento de uma *strip*, então uma transposição $\tau(1, i+1, j)$ une ambos. Se $\pi_j = \ell + 1$ é o último elemento de uma *strip*, então uma reversão $\rho(i+1, j+1)$ une ambos. Note que um algoritmo que aplica apenas essas operações garante que a primeira *strip* da permutação é sempre crescente. Por esse motivo basta considerar apenas o elemento que é uma unidade maior do que o último elemento da primeira *strip*. □

Teorema 6. [43] SbRT *é 3-aproximável.*

Demonstração. Diretamente dos Lemas 4 e 5. □

Algoritmo 1 Ordenando π_n^5 com reversões e transposições

RT_FAMILIA_5 ($\pi = \pi_n^5$, $\mathtt{n} \geq 4$)

```
1  if  n mod 2 = 0  then
2       π ← π · ρ(π^{-1}_{n-2}, π^{-1}_{n-1})
3       for  i ← n − 4 downto 2 by −2  do
4            π ← π · τ(π^{-1}_{i}, π^{-1}_{n}, π^{-1}_{i-1})
5       π ← π · ρ(1, n)
6  else
7       π ← π · τ(π^{-1}_{n-3}, π^{-1}_{n}, π^{-1}_{n-4})
8       for  i ← n − 5 downto 2 by −2  do
9            π ← π · τ(π^{-1}_{i}, π^{-1}_{n-1}, π^{-1}_{i-1})
10      π ← π · τ(1, 3, n + 1) · ρ(1, n − 2)
```

Lema 32. *Para $n \geq 4$, $d_{\rho\tau}(\pi_n^5) = \left\lceil \frac{n}{2} \right\rceil$.*

Demonstração. Primeiro note que $d_{\rho\tau}(\pi_n^5) \leq \left\lceil \frac{n}{2} \right\rceil$ devido ao Algoritmo 1. Agora, note que quando n é par, não existe uma transposição que remove três *breakpoints* de uma vez de π_n^5. Isso acontece porque os números ímpares estão completamente separados dos números pares e pode-se demonstrar isso com uma simples contradição. No entanto, existem quatro formas de remover dois *breakpoints*:

(1) colocando qualquer par i, $2 \leq i \leq n-4$, entre $i+1$ e $i-1$;

(2) colocando qualquer ímpar i, $1 \leq i \leq n-5$, entre $i-1$ e $i+1$;

(3) invertendo o segmento $n-2, n-1$; ou

(4) colocando o segmento $2, 4, 6, \ldots, n-4, n$ no fim da permutação, entre 1 e $n+1$.

Note que ainda não é possível remover três *breakpoints*. A separação entre pares e ímpares continua sendo o motivo. Além disso, é sempre possível remover dois *breakpoints*, bastando colocar um número par (ímpar) entre seu sucessor e antecessor. Como remover três *breakpoints* não vai ser possível nunca porque a separação vai ser mantida e remover um apenas aumentaria o tamanho da sequência de ordenação, o algoritmo é ótimo.

Quando n é ímpar, é possível remover três *breakpoints* na primeira operação. Isso gera $\pi' = (2\ 4\ 6\ \ldots\ n-5\ n-1\ n\ n-2\ n-3\ n-4\ n-6\ \ldots\ 3\ 1)$. Agora, não é possível mais remover três. No entanto, existem três formas de remover dois *breakpoints*:

(1) colocando qualquer par i, $2 \leq i \leq n-5$, entre $i+1$ e $i-1$;

(2) colocando qualquer ímpar i, $1 \leq i \leq n-6$, entre $i-1$ e $i+1$; ou

(3) colocando o segmento $2, 4, 6, \ldots, n-5, n-1, n$ no fim da permutação, entre 1 e $n+1$.

Pelo mesmo motivo do caso par, não é possível remover três *breakpoints*, porque há uma separação dos pares e ímpares. Assim, também é sempre possível remover dois *breakpoints* ao mover um número par (ímpar) para entre seu sucessor e antecessor.

Se escolhermos sempre mover os $\lceil \frac{n}{2} \rceil - 3$ números pares para entre os ímpares, chegará um momento em que a permutação vai ser da forma $(n{-}1\ n\ n{-}2\ n{-}3\ n{-}4\ \ldots\ 3\ 2\ 1)$, um movimento que remove um *breakpoint* seguido por outro que remove dois *breakpoints* serão necessários para ordenar π, e um total de $\lceil \frac{n}{2} \rceil$ movimentos foram utilizados. Se escolhermos sempre mover os $\lceil \frac{n}{2} \rceil - 3$ números ímpares para entre os pares, chegará um momento em que a permutação vai ser da forma $(1\ 2\ 3\ \ldots\ n-6\ n-5\ n-1\ n\ n-2\ n-3\ n-4)$, um movimento que remove zero *breakpoints* seguido por outro que remove três *breakpoints* serão necessários para ordenar π, e, novamente, um total de $\lceil \frac{n}{2} \rceil$ movimentos foram utilizados.

Note que se não escolhermos mover apenas os pares (ímpares) para entre os ímpares (pares), seriam necessários mais movimentos para ordenar a permutação. □

Teorema 33. *Para $n \geq 4$, $D_{\rho\tau}(n) \geq \lceil \frac{n}{2} \rceil$ e para $n \geq 9$, $D_{\rho\tau}(n) \leq \lfloor \frac{2n-2}{3} \rfloor$.*

Demonstração. O limitante inferior é verdadeiro devido à família π_n^5, como o Lema 32 mostra. O limitante superior é verdadeiro porque $D_{\rho\tau}(n) \leq \min\{D_\rho(n), D_\tau(n)\}$, uma vez que $d_{\rho\tau}(\pi) \leq \min\{d_\rho(\pi), d_\tau(\pi)\}$ para qualquer π. Isso por sua vez é verdade porque qualquer sequência de ordenação para SbR ou para SbT é válida também para SbRT, mas não necessariamente ótima. □

A.4 Problema de Ordenação por Reversões de Prefixo

Lema 7. [25] *Seja $\pi \in S_n$ uma permutação qualquer. Então, $d_{\rho_p}(\pi) \geq b_{\rho_p}(\pi)$.*

Demonstração. Uma reversão de prefixo $\rho_p(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* apenas no segundo par, uma vez que (π_0, π_1) nunca é considerado um *breakpoint* de reversão de prefixo. Logo, $\Delta b_{\rho_p}(\pi, \rho_p) \in \{-1, 0, 1\}$. □

Lema 8. [25] *Seja $\pi \neq \iota_n$ uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo duas reversões de prefixo que pode remover um breakpoint existe se $G_b(\pi)$ contém pelo menos uma das quatro arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 1;

(2) (π_1, π_j) do tipo 3;

(3) (π_i, π_j) do tipo 2 com $i \neq 0$;

(4) (π_i, π_j) do tipo 3 com $i > 1$.

Demonstração. Se (π_i, π_j) é uma aresta azul, então $\pi_j = \pi_i \pm 1$. Para criar uma adjacência entre π_i e π_j sem criar novos *breakpoints* basta fazer, para cada tipo de aresta:

(1) Uma reversão de prefixo $\rho_p(j-1)$, que une π_1 e π_j;

(2) Uma reversão de prefixo $\rho_p(j-1)$, que une π_1 e π_j;

(3) Uma reversão de prefixo $\rho_p(j)$, que gera $\pi' = (\pi_j \ \ldots \ \pi_i \ \ldots \ \pi_1 \ \pi_{j+1} \ \ldots \ \pi_n)$, seguida por outra reversão de prefixo $\rho_p(j-i)$ que une π_j e π_i. Note que π' contém a aresta azul $(\pi'_1 = \pi_j, \pi'_{j-i+1} = \pi_i)$, sobre a qual a segunda reversão age;

(4) Uma reversão de prefixo $\rho_p(i)$, que gera $\pi' = (\pi_i \ \ldots \ \pi_1 \ \pi_{i+1} \ \ldots \ \pi_j \ \ldots \ \pi_n)$, seguida por outra reversão de prefixo $\rho_p(j-1)$ que une π_i e π_j. Note que π' contém a aresta azul $(\pi'_1 = \pi_i, \pi'_j = \pi_j)$, sobre a qual a segunda reversão age.

Note que quando $i = 1$, a primeira reversão do último item não precisa ser realizada. Fischer e Ginzinger [25] não especificaram isto, de forma que o trabalho deles não contém o segundo caso que foi descrito neste lema. □

Lema 9. [25] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de prefixo. Então, π não contém nenhum singleton.*

Demonstração. Suponha que π possui um *singleton* k. Então uma aresta vermelha termina em k e outra aresta vermelha começa em k. Além disso, duas arestas azuis também começam ou terminam em k. Se $k+1$ e $k-1$ estivessem à direita de k, então haveriam arestas boas do tipo 2 ou 3 em π, o que é uma contradição. Considere, sem perda de generalidade, $k+1$. Como já foi dito, ele deve estar à esquerda de k. Se uma aresta vermelha começa em $k+1$, então haveria uma aresta boa do tipo 3 em π. Então a única aresta vermelha que envolve $k+1$ acaba ali. Logo, deve haver uma *strip* crescente com início em $k+1$ e término em $k+\ell$, para algum $\ell \geq 1$, isto é, π é da forma $(\ldots\ldots \blacksquare \underline{k+1 \ldots k+\ell} \blacksquare \ldots\ldots \blacksquare k \blacksquare \ldots\ldots)$ e uma aresta vermelha começa em $k+\ell$. Se $k+\ell = n$, então haveria uma aresta do tipo 3 em π, uma vez que uma aresta vermelha termina em $n+1$. Caso contrário, pelo mesmo motivo anterior, as arestas azuis que saem de $k+\ell$ devem estar para a esquerda. Novamente, $k+\ell+1$ tem uma aresta vermelha que acaba nele. Seguindo esse raciocínio, eventualmente atingiremos n, o que nos levaria a uma contradição. □

Lema 10. [25] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de prefixo. Então, a primeira aresta de π é do tipo 2, a última é do tipo 1 e todas as outras são do tipo 4.*

Demonstração. Seja π_i o último elemento da primeira *strip* de π. Devemos ter que $\pi_i = 1$, caso contrário haveria um $\pi_j = \pi_i - 1$ com $i+1 < j$, o que é uma contradição, uma vez que uma aresta boa do tipo 2 ou 3 existiria. Como (π_0, π_1) é sempre uma aresta vermelha e $\pi_1 \neq 1$ porque não existem *singletons* de acordo com Lema 9, a primeira aresta azul, que começa em π_0, só pode ser do tipo 2 ou 3. Como não existem arestas boas, ela deve ser do tipo 2.

Note que as outras arestas só podem ser do tipo 1 ou 4 (caso contrário elas seriam boas). Note também que, como π não contém *singletons*, sempre existem 0 ou 2 arestas (uma vermelha e uma azul) incidentes a cada vértice de $G_b(\pi)$. Portanto, $G_b(\pi)$ forma um único ciclo. Seguindo

esse ciclo a partir do fim da primeira aresta azul f, procuramos pela primeira aresta azul f' do tipo 1. Seja π_j o início da aresta vermelha que fica à direita de f'. A aresta azul que incide em π_j (deve haver uma) termina ali, pois se começasse haveria uma aresta boa do tipo 2 ou 3 em π. Agora ou temos que $f' = f$, ou então o argumento pode continuar indutivamente até que eventualmente atinjamos f. Logo, f' deve ser a última aresta azul de π: uma aresta do tipo 1 deve existir, caso contrário não haveria um ciclo, e outro ciclo em π não pode existir, pois ele deveria começar com uma aresta do tipo 2, o que seria uma contradição. □

Lema 11. [25] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de prefixo. Então π é da forma*

$$(\underbrace{p_1 \ \dots \ 1}_{\ell_1} \ \underbrace{p_2 \ \dots \ p_1{+}1}_{\ell_2} \ \cdots\cdots \ \underbrace{t \ \dots \ p_{b_{\rho_p}(\pi)-1}{+}1}_{\ell_{b_{\rho_p}(\pi)}} \ \underline{t{+}1 \ t{+}2 \ \dots \ n} \ ) \tag{1}$$

onde $t \leq n$. Em outras palavras, π consiste em $b_{\rho_p}(\pi) \geq 2$ strips decrescentes de tamanho $\ell_i \geq 2$, para todo $1 \leq i \leq b_{\rho_p}(\pi)$, de tal forma que se tais strips fossem crescentes, a permutação seria igual à identidade.

Demonstração. Se $b_{\rho_p}(\pi) = 1$, então ou π tem duas *strips* e a última termina em n, ou $\pi = \eta_n$. Em ambos os casos, uma aresta boa de prefixo (π_1, π_j) do tipo 1 existe. Por isso, $b_{\rho_p}(\pi) \geq 2$. Além disso, $\ell_i \geq 2$ para todo $1 \leq i \leq b_{\rho_p}(\pi)$ porque π não tem *singleton*, conforme o Lema 9 mostra.

Quando $b_{\rho_p}(\pi) = 2$, ou π possui duas *strips* ou então possui três *strips* sendo que a última termina em n. Não é difícil perceber que, de todos os 10 possíveis formatos com essas características, as permutações que não possuem arestas boas de prefixo são apenas aquelas nas formas $(\underline{k \ k{-}1 \ \dots \ 1} \blacksquare \underline{n \ n{-}1 \ \dots \ k{+}1} \blacksquare)$ e $(\underline{k \ k{-}1 \ \dots \ 1} \blacksquare \underline{k{+}\ell \ k{+}\ell{-}1 \ \dots \ k{+}1} \blacksquare \underline{k{+}\ell{+}1 \ k{+}\ell{+}2 \ \dots \ n})$, isto é, da mesma forma dada em (1).

A partir daqui vamos escrever $b = b_{\rho_p}(\pi)$ por simplificação. Suponha que toda permutação sem arestas boas de prefixo com $b - 1 \geq 2$ *breakpoints* é da forma dada em (1).

Seja π uma permutação com $b > 2$ *breakpoints* sem arestas boas de prefixo. Seja π_w o último elemento da primeira *strip* de π. Devemos ter que $\pi_w = 1$, pois, pelo Lema 10, a primeira aresta é do tipo 2 e começa em π_0.

Seja π' uma permutação construída a partir de π tal que $\pi'_j = \pi_{j+w} - \pi_1$ para todo $1 \leq j \leq n - w$. Note que se π' tivesse arestas boas do tipo 2 ou 3, é fácil ver que π também as teria. Se π' tivesse uma aresta boa do tipo 1 (π'_1, π'_j), para algum $j > 2$, então em π ela seria (π_{w+1}, π_{w+j}) do tipo 1 (não boa). Mas pelo Lema 10 sabemos que π só pode ter uma aresta do tipo 1, a última. Como π_{w+1} é o primeiro elemento da segunda *strip* de π, então (π_{w+1}, π_{w+j}) é a terceira aresta azul de π (a primeira começa em π_0 e a segunda começa em π_1). Portanto, ela só pode ser a última de π se $b = 2$.

Dessa forma, π' não possui arestas boas. Além disso, ela tem $b - 1$ *breakpoints* de reversão

de prefixo. Logo, pela hipótese de indução, π' é da forma dada em (1). Como π' é π renomeada sem a primeira *strip*, que tem o elemento 1, segue que π também é da forma dada em (1). □

Lema 12. [25] *Seja π uma permutação sem sinal de tamanho n que é da forma dada em (1). Para essa permutação, a sequência de $2b_{\rho_p}(\pi)$ reversões de prefixo*

$$\rho_p(t) \cdot \rho_p(t-\ell_1) \cdot \rho_p(t) \cdot \rho_p(t-\ell_2) \cdot \ldots \cdot \rho_p(t) \cdot \rho_p(t-\ell_{b_{\rho_p}(\pi)}) \tag{2}$$

quando aplicada a π, transforma-a na permutação identidade.

Demonstração. É fácil perceber que qualquer sequência de reversões de prefixo que ordena ($\underline{p_1 \ \ldots \ 1}$ $\underline{p_2 \ \ldots \ p_1{+}1}$ ..... $\underline{t \ \ldots \ p_{b_{\rho_p}(\pi)-1}{+}1}$ $\underline{t{+}1 \ t{+}2 \ \ldots \ n}$) também irá ordenar ($\underline{p_1 \ \ldots \ 1}$ $\underline{p_2 \ \ldots \ p_1{+}1}$ ..... $\underline{t \ \ldots \ p_{b_{\rho_p}(\pi)-1}{+}1}$). Portanto, escreveremos π da última maneira, por simplificação. Também escreveremos $b = b_{\rho_p}(\pi)$.

Seja π^k, com $1 \leq k \leq b-1$, a permutação que obtemos depois de aplicar as primeiras $2k$ reversões de prefixo da sequência dada em (2), isto é, depois de aplicar $\rho_p(t) \cdot \rho_p(t-\ell_1) \cdot \ldots \cdot \rho_p(t) \cdot \rho_p(t-\ell_k)$. Vamos mostrar por indução em k que

$$\pi^k = (\underbrace{p_{k+1} \ \ldots \ p_k{+}1}_{\ell_{k+1}} \ \underbrace{p_{k+2} \ \ldots \ p_{k+1}{+}1}_{\ell_{k+2}} \ \ldots\ldots \ \underbrace{t \ \ldots \ p_{b-1}{+}1}_{\ell_b} \ \underbrace{1 \ 2 \ \ldots \ p_k}_{\ell_1+\ell_2+\ldots+\ell_k}).$$

Quando $k = 1$, as duas primeiras operações de (2) são $\rho_p(t)$ e $\rho_p(t-\ell_1)$ e temos que $\pi' = \pi \cdot \rho_p(t) = (\underline{p_{b-1}{+}1 \ \ldots \ t}$ ..... $\underline{p_2{+}1 \ \ldots \ p_3 \ p_1{+}1 \ \ldots p_2 \ 1 \ \ldots \ p_1})$ e $\pi'' = \pi' \cdot \rho_p(t-\ell_1) = (\underline{p_2 \ \ldots p_1{+}1} \ \underline{p_3 \ \ldots \ p_2{+}1}$ ..... $\underline{t \ \ldots \ p_{b-1}{+}1} \ \underline{1 \ \ldots \ p_1})$. É fácil ver que π'' é da mesma forma que π^1.

Agora assuma que π^{k-1} é da forma dada, isto é,

$$\pi^{k-1} = (\underbrace{p_k \ \ldots \ p_{k-1}{+}1}_{\ell_k} \ \underbrace{p_{k+1} \ \ldots \ p_k{+}1}_{\ell_{k+1}} \ \ldots\ldots \ \underbrace{t \ \ldots \ p_{b-1}{+}1}_{\ell_b} \ \underbrace{1 \ 2 \ \ldots \ p_{k-1}}_{\ell_1+\ell_2+\ldots+\ell_{k-1}}).$$

É fácil ver que $\pi^k = \pi^{k-1} \cdot \rho_p(t) \cdot \rho_p(t-\ell_k)$. Portanto, o resultado segue. Ao fim, $\pi^{b-1} = (\underline{t \ \ldots \ p_{b-1}{+}1} \ \underline{1 \ 2 \ \ldots \ p_{b-1}})$ e $\rho_p(t) \cdot \rho_p(t-\ell_b)$ a ordena.

□

`2-PR` e `2-PRg` são apresentados no Algoritmo 2, uma vez que a única diferença entre eles é a forma com que eles percorrem a permutação.

Lema 13. [25] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p}(\pi) \leq 2b_{\rho_p}(\pi)$.*

Demonstração. Os algoritmos `2-PR` e `2-PRg` aplicam a sequência de reversões de prefixo dada por (2) quando não conseguem mais encontrar arestas boas de prefixo, as quais eles conseguem lidar com no máximo dois movimentos. Assim, no pior caso eles usam duas reversões de prefixo para remover um *breakpoint*. □

Algoritmo 2 Algoritmo de 2-aproximação para SbPR

2-PR/2-PRg(π, $\mathtt{n}$)

```
1  d ← 0
2  while  π ≠ ι_n  do
3       // Tenta remover um breakpoint com uma reversão
4       if  G_b(π) tem uma aresta boa de prefixo (π_1, π_j) do tipo 1  then
5            π ← π · ρ_p(j − 1)
6            d ← d + 1
7       else if G_b(π) tem uma aresta boa de prefixo (π_1, π_j) do tipo 3  then
8            π ← π · ρ_p(j − 1)
9            d ← d + 1
10      // Tenta remover um breakpoint com duas reversões
11      else if G_b(π) tem uma aresta boa de prefixo (π_i, π_j) do tipo 2  then
12           π ← π · ρ_p(j) · ρ_p(j − i)
13           d ← d + 2
14      else if G_b(π) tem uma aresta boa de prefixo (π_i, π_j) do tipo 3  and i > 1  then
15           π ← π · ρ_p(i) · ρ_p(j − 1)
16           d ← d + 2
17      // Forma especial
18      else
19           Seja t o maior elemento positivo fora de posição em π
20           Seja ℓ_i o tamanho da i-ésima strip de π
21           d ← d + 2b_{ρ_p}(π)
22           π ← π · ρ_p(t) · ρ_p(t − ℓ_1) · ρ_p(t) · ρ_p(t − ℓ_2) · ... · ρ_p(t) · ρ_p(t − ℓ_{b_{ρ_p}(π)})
23 return  d
```

Teorema 14. SBPR *é 2-aproximável.*

Demonstração. Diretamente dos Lemas 7 e 13. □

A.5 Problema de Ordenação por Reversões de Prefixo com Sinal

Lema 15. [18] *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}}(\pi) \geq b_{\bar{\rho_p}}(\pi)$.*

Demonstração. Uma reversão de prefixo com sinal $\bar{\rho_p}(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* apenas no segundo par, uma vez que (π_0, π_1) nunca é considerado um *breakpoint* de reversão de prefixo com sinal. Assim, temos que $\Delta b_{\bar{\rho_p}}(\pi) \in \{-1, 0, 1\}$. □

Lema 16. [26] *Seja π uma permutação com sinal sem elementos positivos fora de ordem para a qual não existem elementos π_i e π_j tais que $\pi_i = -(k+1)$ e $\pi_j = -k$ para algum $k \geq 1$ com $i+1 < j$. Então π é da forma*

$$\pi = (\underbrace{-p_1 \ldots -1}_{\ell_1}\ \underbrace{-p_2 \ldots -(p_1{+}1)}_{\ell_2}\ \ldots\ldots\ \underbrace{-t \ldots -(p_{b_{\bar{\rho_p}}(\pi)-1}{+}1)}_{\ell_{b_{\bar{\rho_p}}(\pi)}}\ \underline{t{+}1\ t{+}2\ \ldots\ n}) \qquad (3)$$

onde $t \leq n$. Em outras palavras, π consiste em $b_{\bar{\rho_p}}(\pi) \geq 2$ strips negativas de tamanho $\ell_i \geq 1$, para todo $1 \leq i \leq b_{\bar{\rho_p}}(\pi)$, de tal forma que se tais strips fossem positivas, a permutação seria igual à identidade.

Demonstração. Note que, se $b_{\bar{\rho_p}}(\pi) = 1$, então ou π é formada por uma *strip*, isto é, $\pi = \bar{\eta_n}$, ou π é formada por duas *strips* onde a última termina em $+n$. Em ambos os casos, existe um $\pi_j = -\pi_1 + 1$ e uma reversão de prefixo que remove um *breakpoint* pode ser aplicada. Logo, $b_{\bar{\rho_p}}(\pi) \geq 2$.

Vamos provar por indução no número b de *strips* negativas de π que ela deve ter o formato declarado. Quando $b = 2$, ou π possui duas *strips* negativas ou então possui três *strips* sendo que as duas primeiras são negativas e a última é positiva e termina em $+n$. Não é difícil perceber que, de todos os 8 possíveis formatos, as permutações que não possuem elementos positivos para as quais não se consegue remover um *breakpoint* são apenas aquelas nas formas $(\underline{-k\ -(k-1)\ \ldots\ -1}\ .\ \underline{-n\ -(n-1)\ \ldots\ -(k+1)}\ .)$ e $(\underline{-k\ -(k-1)\ \ldots\ -1}\ .\ \underline{-(k+\ell)\ -(k+\ell-1)\ \ldots\ -(k+1)}\ .\ \underline{k+\ell+1\ k+\ell+2\ \ldots\ n})$ isto é, da mesma forma dada em (3).

Assuma então que toda permutação com $b-1$ *strips* negativas sem elementos positivos fora de ordem para a qual não existem elementos $\pi_i = -(k+1)$ e $\pi_j = -k$ com $i+1 < j$ é da forma dada em (3).

Para o passo indutivo, considere uma permutação π com b *strips* negativas envolvendo os elementos de $-t$ a -1, $t \leq n$, sem elementos positivos fora de ordem para a qual não existem elementos $\pi_i = -(k+1)$ e $\pi_j = -k$ com $i+1 < j$. Se a primeira *strip* de π termina em π_w,

$w \geq 1$, então deve-se ter que $\pi_w = -1$. Caso contrário, deveríamos ter $\pi_j = \pi_w + 1$ para algum $j > w + 1$, uma contradição.

Seja π' uma permutação construída a partir de π tal que $\pi'_j = \pi_{j+w} - \pi_1$ para $1 \leq j \leq t - w$. É fácil ver que π' tem $b - 1$ *strips* negativas. Além disso, suponha que π' tem elementos $\pi'_i = -(k + 1)$ e $\pi'_j = -k$ com $i + 1 < j$ para algum k. Então π também deveria ter elementos $\pi_{i+w} = -(k+1)+\pi_1$ e $\pi_{j+w} = -k+\pi_1$, o que é uma contradição. Portanto, π' é da forma dada em (3). Como π' é π renomeada sem a primeira *strip*, que possui o elemento -1, segue que π também é da forma dada em (3). □

Lema 17. [26] *Seja π uma permutação com sinal da forma descrita em (3). A seguinte sequência de $2b_{\bar{\rho}_p}(\pi)$ reversões de prefixo ordena π:*

$$\bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_1) \cdot \bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_2) \cdot \ldots \cdot \bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_{b_{\bar{\rho}_p}(\pi)}). \tag{4}$$

Demonstração. É fácil perceber que uma sequência de reversões de prefixo que ordena $(\underline{-p_1 \ \ldots \ -1} \ \underline{-p_2 \ \ldots \ -(p_1{+}1)} \ldots\ldots \underline{-t \ \ldots \ -(p_{b_{\bar{\rho}_p}(\pi)-1}{+}1)} \ \underline{t{+}1 \ t{+}2 \ \ldots \ n})$ também irá ordenar $(\underline{-p_1 \ \ldots \ -1} \ \underline{-p_2 \ \ldots \ -(p_1{+}1)} \ \ldots\ldots \ \underline{-t \ \ldots \ -(p_{b_{\bar{\rho}_p}(\pi)-1}{+}1)})$. Portanto, usaremos a última durante essa prova por simplificação. Além disso, de agora em diante, $b = b_{\bar{\rho}_p}(\pi)$.

Seja π^k, $1 \leq k \leq b - 1$, a permutação obtida depois de aplicar as primeiras $2k$ reversões de prefixo da sequência dada em (4), isto é, depois de aplicar $\bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_1) \cdot \ldots \cdot \bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_k)$. Vamos mostrar por indução em k que

$$\pi^k = (\underbrace{-p_{k+1} \ldots -(p_k{+}1)}_{\ell_{k+1}} \ \underbrace{-p_{k+2} \ldots -(p_{k+1}{+}1)}_{\ell_{k+2}} \ldots\ldots \underbrace{-t \ldots -(p_{b-1}{+}1)}_{\ell_b} \ \underbrace{1 \ 2 \ \ldots \ p_k}_{\ell_1+\ell_2+\ldots\ell_k}).$$

Quando $k = 1$, as duas primeiras reversões de (4) são $\bar{\rho}_p(t)$ e $\bar{\rho}_p(t - \ell_1)$ e temos que $\pi' = \pi \cdot \bar{\rho}_p(t) = (\underline{p_{b-1}{+}1 \ \ldots \ t} \ \ldots\ldots \ \underline{p_2{+}1 \ \ldots \ p_3} \ \underline{p_1{+}1 \ \ldots \ p_2} \ \underline{1 \ \ldots \ p_1})$ e também que $\pi'' = \pi' \cdot \bar{\rho}_p(t - \ell_1) = (\underline{-p_2 \ \ldots \ -(p_1{+}1)} \ \underline{-p_3 \ \ldots \ -(p_2{+}1)} \ \ldots\ldots \ \underline{-t \ \ldots \ -(p_{b-1}{+}1)} \ \underline{1 \ \ldots \ p_1})$. É fácil ver que π'' é da mesma forma que π^1.

Agora, assuma que π^{k-1} é da forma dada acima, isto é,

$$\pi^{k-1} = (\underbrace{-p_k \ldots -(p_{k-1}{+}1)}_{\ell_k} \ \underbrace{-p_{k+1} \ldots -(p_k{+}1)}_{\ell_{k+1}} \ldots\ldots \underbrace{-t \ldots -(p_{b-1}{+}1)}_{\ell_b} \ \underbrace{1 \ 2 \ \ldots \ p_k}_{\ell_1+\ell_2+\ldots\ell_{k-1}}).$$

É fácil ver que $\pi^k = \pi^{k-1} \cdot \bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_k)$. Portanto, o resultado segue. Ao fim, $\pi^{b-1} = (\underline{-t \ \ldots \ -(p_{b-1}{+}1)} \ \underline{1 \ 2 \ \ldots \ p_{b-1}})$ e $\bar{\rho}_p(t) \cdot \bar{\rho}_p(t - \ell_{b-1})$ a ordena. □

Lema 18. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho}_p}(\pi) \leq 2b_{\bar{\rho}_p}(\pi)$.*

Demonstração. No pior caso, serão utilizadas duas reversões de prefixo com sinal para remover um *breakpoint*, mesmo quando a permutação está na forma dada em (3). □

Algoritmo 3 Algoritmo de 2-aproximação para SbSigPR

2-SPR(π, $\mathtt{n}$)

```
 1  d ← 0
 2  while  π ≠ ι_n  do
 3       // Tenta remover um breakpoint com uma operação
 4       if  existe π_j = −π_1 + 1  then
 5            π ← π · ρ̄_p(j − 1)
 6            d ← d + 1
 7       // Tenta remover um breakpoint com duas operações
 8       else if π tem pelo menos um elemento positivo fora de ordem  then
 9            Seja π_i o maior elemento positivo fora de ordem
10            if  existe π_j = −(π_i + 1)  then
11                 if  i < j  then
12                      π ← π · ρ̄_p(j) · ρ̄_p(j − i)
13                 else
14                      π ← π · ρ̄_p(i) · ρ̄_p(i − j)
15            else
16                 π ← π · ρ̄_p(i) · ρ̄_p(k)
17            d ← d + 2
18       // Nesse caso, existem apenas elementos negativos fora de ordem
19       else if existe π_i = −(k + 1)  and existe π_j = −k  and i < j − 1  then
20            π ← π · ρ̄_p(i) · ρ̄_p(j − 1)
21            d ← d + 2
22       else
23            Seja t o número de elementos fora de ordem em π
24            Seja ℓ_i o tamanho da i-ésima strip de π
25            d ← d + 2b_ρ̄p(π)
26            π ← π · ρ̄_p(t) · ρ̄_p(t − ℓ_1) · ρ̄_p(t) · ρ̄_p(t − ℓ_2) · ... · ρ̄_p(t) · ρ̄_p(t − ℓ_{b_ρ̄p(π)})
27  return  d
```

Teorema 19. SbSigPR *é 2-aproximável.*

Demonstração. Diretamente dos Lemas 15 e 18. □

A.6 Problema de Ordenação por Transposições de Prefixo

Lema 20. [19] *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer. Sempre é possível obter uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p}(\pi, \tau_p) \leq -1$.*

Demonstração. Seja π_i o último elemento da primeira *strip* de π. Deve existir outra *strip* em π que começa com algum π_j tal que $\pi_j = \pi_i + 1$ e $i < j$. Claramente, existe um *breakpoint* entre π_{j-1} e π_j e a transposição de prefixo $\tau_p(i+1, j)$ o remove. □

Lema 21. [19] *Seja $\pi \neq \iota_n$ uma permutação qualquer. Existe no máximo uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p}(\pi, \tau_p) = -2$.*

Demonstração. Suponha que $\tau_p(i, j)$ remove dois *breakpoints* de uma vez de π. Nesse caso, $\pi \cdot \tau_p(i, j) = (\pi_i \ \ldots \ \pi_{j-1} \ \pi_1 \ \ldots \ \pi_{i-1} \ \pi_j \ \ldots \ \pi_n)$, onde $\pi_{i-1} = \pi_j - 1 \neq \pi_i - 1$ e $\pi_{j-1} = \pi_1 - 1 \neq \pi_j - 1$. É fácil ver que π_1 determina unicamente j e j determina unicamente i. □

Lema 22. [19] *Seja π uma permutação sem sinal qualquer. Então, $d_\tau(\pi) \geq \frac{b_\tau(\pi)}{2}$.*

Demonstração. Uma transposição de prefixo $\tau_p(i, j)$ separa $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos pares, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de transposição de prefixo. Logo, $\Delta b_{\tau_p}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. □

Lema 23. [19] *Seja π uma permutação qualquer e $\tau_p(i, j)$ uma transposição de prefixo tal que $\pi \cdot \tau_p = \iota_n$. Então, $\pi_i = 1$ e $\Delta b_{\tau_p}(\pi, \tau_p) = -2$.*

Demonstração. É fácil ver que o único formato de uma permutação que está à uma transposição de prefixo de distância da permutação identidade é

$$(\underline{k{+}1 \ k{+}2 \ \ldots \ k{+}\ell} \ \underline{1 \ 2 \ \ldots \ k} \ \underline{k{+}\ell{+}1 \ k{+}\ell{+}2 \ \ldots \ n})$$

com $k \geq 1$ e $\ell \geq 1$. □

Lema 24. [19] *Seja π uma permutação sem sinal qualquer. Então, $d_{\tau_p}(\pi) \leq b_{\tau_p}(\pi) - 1$.*

Demonstração. No pior caso, sempre é possível remover um *breakpoint* de transposição de prefixo com uma transposição de prefixo. Além disso, a última operação da ordenação sempre remove dois *breakpoints* de uma vez, conforme o Lema 23. □

Teorema 25. SbPT *é 2-aproximável.*

Demonstração. Diretamente dos Lemas 22 e 24. □

Algoritmo 4 Algoritmo de 2-aproximação para SbPT

2-PT(π, n)

```
1  d ← 0
2  while  π ≠ ι_n  do
3        Seja i a posição do último elemento da primeira strip de π
4        j ← π^{-1}(π_i + 1)
5        π ← π · τ_p(i + 1, j)
6        d ← d + 1
7  return  d
```

Algoritmo 5 Algoritmo de 2-aproximação para SbPT, versão gulosa

2-PTg(π, n)

```
 1  d ← 0
 2  while  π ≠ ι_n  do
 3        // Tenta remover dois breakpoints
 4        j ← π^{-1}(π_1 − 1) + 1
 5        i ← π^{-1}(π_j − 1) + 1
 6        if  1 < i  and i < j  then
 7              π ← π · τ_p(i, j)
 8        // Remove apenas um breakpoint
 9        else
10              Seja i a posição do último elemento da primeira strip de π
11              j ← π^{-1}(π_i + 1)
12              π ← π · τ_p(i + 1, j)
13        d ← d + 1
14  return  d
```

A.7 Problema de Ordenação por Reversões de Prefixo e Transposições de Prefixo

Lema 26. [40] *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$.*

Demonstração. Uma reversão de prefixo $\rho_p(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* apenas no último, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de reversão de prefixo. Assim, $\Delta b_{\rho_p}(\pi, \rho_p) \in \{-1, 0, 1\}$. Uma transposição de prefixo $\tau_p(i, j)$ separa os pares de elementos $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de reversão de prefixo. Assim, $\Delta b_{\rho_p}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. □

Lema 27. *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\tau_p}(\pi)}{n} \right\rceil$.*

Demonstração. Uma transposição de prefixo $\tau_p(i, j)$ separa os pares de elementos $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de transposição de prefixo. Assim, $\Delta b_{\tau_p}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. Por outro lado, não é possível fazer uma análise similar para $\Delta b_{\tau_p}(\pi, \rho_p)$, isto é, a variação no número de *breakpoints* de transposição de prefixo após uma reversão de prefixo. Isso ocorre porque uma reversão $\rho_p(i)$ pode alterar a configuração de *breakpoints* de transposição de prefixo em todo o intervalo $\pi_1, \ldots, \pi_i$, o que, no pior caso, faz com que $\Delta b_{\tau_p}(\pi, \rho_p) \in \{-n, \ldots, n\}$. □

Lema 28. [40] *Seja (π_i, π_j) uma aresta azul do tipo 4 de uma permutação π qualquer. Então existe pelo menos uma aresta vermelha (π_{k-1}, π_k) para algum $i < k < j$. Tal aresta é chamada de* ***presa****.*

Demonstração. Se não existe tal aresta vermelha, então $\pi_i, \pi_{i+1}, \ldots, \pi_j$ é uma *strip*, o que é uma contradição pois (π_i, π_j) não seria uma aresta azul. □

Lema 29. [40] *Seja $\pi \neq \iota_n$ uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo três reversões de prefixo e transposições de prefixo que remove até dois breakpoints existe se $G_b(\pi)$ contém pelo menos uma das quatro arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 4 com $\pi_1 \neq 1$;

(2) (π_1, π_j) do tipo 1 com $\pi_1 \neq 1$;

(3) (π_i, π_j) do tipo 3 com $\pi_1 = 1$;

(4) (π_i, π_j) do tipo 2 com $\pi_1 = 1$ onde π_i é o último elemento da primeira strip de π.

Demonstração. Se existe uma aresta azul (π_i, π_j), então $\pi_j = \pi_i \pm 1$. Para criar uma adjacência entre π_i e π_j sem criar novos *breakpoints* deve-se fazer, respectivamente, para cada tipo de aresta:

(1) Uma transposição de prefixo $\tau_p(k, j+1)$ onde (π_{k-1}, π_k) é uma aresta vermelha presa, $i < k < j$, que une π_1 e π_j;

(2) Uma reversão de prefixo $\rho_p(j-1)$, que une π_1 e π_j;

(3) Uma transposição de prefixo $\tau_p(i+1, j)$, que une π_i e π_j;

(4) Uma reversão de prefixo $\rho_p(j)$, que gera $\pi' = (\pi_j \ \ldots \ \pi_i \ \ldots \ \pi_1 \ \pi_{j+1} \ \ldots \ \pi_n)$ sem remover nenhum *breakpoint*, seguida por uma reversão de prefixo $\rho_p(j-i)$, que gera $\pi'' = (\pi_{i+1} \ \ldots \ \pi_j \ \pi_i \ \ldots \ \pi_1 \ \pi_{j+1} \ \ldots)$ ao agir sobre a aresta azul $(\pi'_1 = \pi_j, \pi'_{j-i+1} = \pi_i)$, seguida por uma operação para lidar com uma aresta do tipo 4 ou do tipo 1, uma vez que a aresta vermelha $(\pi''_0 = \pi_0, \pi''_1 = \pi_{i+1})$ existe. Em resumo, usa-se 3 operações para remover 2 *breakpoints*.

□

Algoritmo 6 Algoritmo de 3-aproximação para SbPRPT

```
3-PRPT(π, n)
 1  d ← 0
 2  while π ≠ ι_n do
 3      if π_1 ≠ 1 then
 4          if G_b(π) tem uma aresta (π_1, π_j) do tipo 4 then
 5              Seja (k − 1, k) uma aresta vermelha presa entre π_1 e π_j
 6              π ← π · τ_p(k, j + 1)
 7              d ← d + 1
 8          else if G_b(π) tem uma aresta (π_1, π_j) do tipo 1 then
 9              π ← π · ρ_p(j − 1)
10              d ← d + 1
11      else if G_b(π) tem uma aresta (π_i, π_j) do tipo 3 then
12          π ← π · ρ_p(i + 1, πj)
13          d ← d + 1
14      else
15          Seja (π_i, π_j) uma aresta do tipo 2
16          π ← π · ρ_p(j) · ρ_p(j − i)
17          d ← d + 2
18  return d
```

Lema 30. *Seja π uma permutação sem sinal qualquer. Então, $d_{\rho_p\tau_p}(\pi) \leq \frac{3b_{\rho_p}(\pi)}{2}$.*

Demonstração. No pior caso, serão aplicadas três operações para remover dois *breakpoints* de reversão de prefixo. □

Teorema 31. SbPRPT *é 3-aproximável.*

Demonstração. Diretamente dos Lemas 26 e 30. □

Lema 34. *Para $n \geq 7$, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n^6) \leq \lceil \frac{n}{2} \rceil + 1$.*

Algoritmo 7 Ordenando π_n^6 com reversões de prefixo e transposições de prefixo

PRPT_FAMILIA_6 ($\pi = \pi_n^6$, $\mathtt{n} \geq 7$)

1 **if** $\mathtt{n} \mod 2 = 0$ **then**
2 $\quad \pi \leftarrow \pi \cdot \rho_p(2) \cdot \tau_p(5, \pi_{\mathtt{n}-3}^{-1} + 1)$
3 **else**
4 $\quad \pi \leftarrow \pi \cdot \tau_p(3, \pi_{\mathtt{n}-4}^{-1}) \cdot \tau_p(2, \pi_{\mathtt{n}}^{-1})$
5 **while** $\pi_1 \neq 2$ **do**
6 $\quad \pi \leftarrow \pi \cdot \tau_p(2, \pi_{\pi_1 - 1}^{-1})$
7 $\pi \leftarrow \pi \cdot \tau_p(\pi_{\mathtt{n}}^{-1} + 1, \mathtt{n} + 1) \cdot \rho_p(\pi_2^{-1}) \cdot \rho_p(2)$

Demonstração. O limitante inferior é verdadeiro porque $b_{\rho_p}(\pi_n^6) = n - 1$ quando n é par, $b_{\rho_p}(\pi_n^6) = n$ quando n é ímpar e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 7. □

Teorema 35. *Para $n \geq 7$, $D_{\rho_p\tau_p}(n) \geq \lceil \frac{n}{2} \rceil$ e para $n \geq 1$, $D_{\rho_p\tau_p}(n) \leq n - \log_{\frac{9}{2}} n$.*

Demonstração. O limitante inferior é verdadeiro devido à família π_n^6, como o Lema 34 mostra. O limitante superior é verdadeiro porque $D_{\rho_p\tau_p}(n) \leq \min\{D_{\rho_p}(n), D_{\tau_p}(n)\}$, uma vez que $d_{\rho_p\tau_p}(\pi) \leq \min\{d_{\rho_p}(\pi), d_{\tau_p}(\pi)\}$ para todo π. □

A.8 Problema de Ordenação por Reversões de Prefixo com Sinal e Transposições de Prefixos

Lema 36. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho}_p\tau_p}(\pi) \geq \frac{b_{\bar{\rho}_p}(\pi)}{2}$.*

Demonstração. Uma reversão de prefixo com sinal $\bar{\rho}_p(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* apenas no último par, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de reversão de prefixo com sinal. Assim, $\Delta b_{\bar{\rho}_p}(\pi, \bar{\rho}_p) \in \{-1, 0, 1\}$. Uma transposição de prefixo $\tau_p(i, j)$ separa os pares de elementos $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos. Assim, $\Delta b_{\bar{\rho}_p}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. □

Lema 37. *Seja $\pi \neq \iota_n$ e $\pi_1 \neq 1$. Sempre é possível remover um breakpoint de π.*

Demonstração. Considere o primeiro elemento $\pi_1 = k$ da permutação e seja π_i o último elemento da primeira *strip* de π. Devemos ter que ou $k-1$ ou $-(k-1)$ existe em π em alguma posição j. Se $k-1$ existir, ele deve ser um *singleton* ou então ele deve estar no final da *strip* que o contém. Em ambos os casos, uma transposição de prefixo $\tau_p(i+1, j+1)$ une $k-1$ e k e remove um *breakpoint*. Se $-(k-1)$ existir, ele deve ser um *singleton* ou então ele deve estar no começo da *strip* que o contém. Em ambos os casos, uma reversão de prefixo $\bar{\rho}_p(j-1)$ une $-k$ e $-(k-1)$ e remove um *breakpoint*. □

Lema 38. *Seja $\pi \neq \iota_n$ uma permutação com sinal da forma $\pi = (\ldots\ldots\ \pi_{n-i}\ \underline{1\ 2\ \ldots\ i})$, com $1 \leq i < n$. A última strip de π será removida do fim apenas quando o elemento n for enviado para lá.*

Demonstração. Como $\pi_1 \neq 1$, o algoritmo ainda é capaz de fazer um de seus três casos principais. Iremos descrever agora o que acontece em cada um desses casos.

Se uma transposição $\tau_p(i,j)$ remove dois *breakpoints*, devemos ter que $\pi_i = \pi_j - 1$. Para remover a última *strip* do final com essa transposição, devemos ter $j = n+1$. Mas então $\pi_{i-1} = n$, que é é enviado para o fim.

Seja π_i o último elemento da primeira *strip* de π e $\tau_p(i,j)$ uma transposição que remove um *breakpoint*. Se ela aumenta a primeira *strip* com seu próximo elemento, então $\pi_j = \pi_{i-1} + 1$. Para remover a última *strip* do fim, deve-se ter $j = n+1$, de forma que $\pi_{i-1} = n$ será enviado para o fim. Se ela aumenta a primeira *strip* com seu elemento anterior, então $\pi_{j-1} = \pi_1 - 1$. Para remover a última *strip* do fim, π_{j-1} deve ser o último elemento da última *strip*, isto é, $j = n+1$. No entanto, neste caso a última *strip* estaria apenas aumentando de tamanho.

Finalmente, se uma reversão $\bar{\rho}_p(j)$ remove um *breakpoint*, deve-se ter $\pi_{j+1} = -\pi_1 + 1$. Para remover a última *strip* do fim com essa reversão, deve-se ter $j = n$. Dessa forma, $\pi_1 = -n$ e n é enviado para o fim da permutação. □

Lema 39. *Seja $\pi \neq \iota_n$ uma permutação com sinal da forma $\pi = (\ldots\ldots\ n\ \underline{1\ 2\ \ldots\ i})$, com $1 \leq i < n$. Então, os elementos n e 1 não serão separados até que n esteja ordenada. Logo, é sempre possível continuar removendo um ou dois breakpoints.*

Demonstração. Temos que $\pi_1 \neq 1$, então o algoritmo primeiro tentará remover dois *breakpoints* com uma transposição $\tau_p(i,j)$. Como explicitamente negamos que $\pi_i = 1$ nessa operação, considere que $\pi_j = 1$. Pelo formato de π, temos que $\pi_{j-1} = n$, mas se a transposição realmente remove dois *breakpoints*, teríamos que ter $\pi_1 = n+1$, o que é impossível.

Em seguida o algoritmo vai tentar aumentar a primeira *strip*, que termina em π_{i-1}, com uma transposição $\tau_p(i,j)$. Se tivéssemos $\pi_i = 1$, significaria que a primeira *strip* termina em n e, consequentemente, π é formada por apenas duas *strips*. Dessa forma, a transposição dada estaria ordenando π e removendo dois *breakpoints* de uma vez. Se, por outro lado, tivéssemos $\pi_j = 1$, existem duas possibilidades. Se a transposição vai unir a primeira *strip* com seu próximo elemento, então π_j é tal próximo elemento, o que nos levaria a ter $\pi_{i-1} = 0$. Se a transposição vai unir a primeira *strip* com seu elemento anterior, então deveríamos ter $\pi_1 = n+1$, uma vez que pelo formato de π, $\pi_{j-1} = n$. Claramente, ambos os casos são impossíveis.

Por fim, o algoritmo tentaria remover um *breakpoint* com uma reversão de prefixo $\bar{\rho}_p(j)$. Se $\pi_{j+1} = 1$, então deveríamos ter $\pi_1 = 0$, o que também é impossível.

Logo, vemos que não há possibilidades do algoritmo separar os elementos n e 1, a menos que ele esteja a um passo de ordenar a permutação. Exceto por esse caso, π_1 vai ser sempre diferente de 1 e os três casos principais do algoritmo poderão sempre ser utilizados. □

Lema 40. *Durante a ordenação, π_1 será 1 no máximo duas vezes.*

Demonstração. Quando $\pi_1 = 1$ e $\pi_n \neq n$, a primeira *strip* é enviada para o fim da permutação, o que não muda o número de *breakpoints*. Como o Lema 38 mostra, ela será removida de lá

quando n for para o fim. Como os passos principais do algoritmo estão sempre tentando remover *breakpoints*, n não será removido do fim até que $\pi_1 = 1$ novamente. Nesse caso, o número de *breakpoints* aumenta em uma unidade. No entanto, quando isso acontece, os elementos n e 1 são unidos e o algoritmo não os separa mais, conforme mostrado pelo Lema 39. □

O Algoritmo 8 apresenta `2-SPRPT`, que também tem complexidade de tempo $O(n^2)$.

Algoritmo 8 Algoritmo assintótico de 2-aproximação para SbSigPRPT

```
2-SPRPT(π, n)
 1  d ← 0
 2  while π ≠ ι_n do
 3      // Enviar a primeira strip para o fim
 4      if π_1 = 1 then
 5          Seja i a posição do último elemento da primeira strip de π
 6          π ← π · τ_p(i + 1, n + 1)
 7      // Tenta remover dois breakpoints com uma transposição de prefixo
 8      else if existe π_{j-1} = π_1 − 1 and existe π_{i-1} = π_j − 1 and 2 ≤ i < j and π_i ≠ 1 then
 9          π ← π · τ_p(i, j)
10      // Tenta remover um breakpoint
11      else
12          Seja i a posição do último elemento da primeira strip de π
13          if existe π_j = π_i + 1 then
14              π ← π · τ_p(i + 1, j)
15          else if existe π_j = π_1 − 1 and j > 1 then
16              π ← π · τ_p(i + 1, j + 1)
17          else
18              π_j = −π_1 + 1
19              π ← π · ρ̄_p(j − 1)
20      d ← d + 1
21  return d
```

Lema 41. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho}_p\tau_p}(\pi) \leq b_{\bar{\rho}_p}(\pi) + 2$.*

Demonstração. Enquanto $\pi_1 \neq 1$, o algoritmo consegue remover um ou dois *breakpoints*. No pior caso, $\pi_1 = 1$ duas vezes e duas operações extras são necessárias para mover a primeira *strip*, como o Lema 40 mostra (uma delas cria um *breakpoint* mas coloca n e 1 juntos, o que vai garantir que a última operação da ordenação remova dois *breakpoints*, como o Lema 39 mostra). □

Teorema 42. SbSigPRPT *é 2-aproximável assintoticamente.*

Demonstração. Dos Lemas 36 e 41, temos que o fator de aproximação teórico é $\frac{b_{\bar{\rho}_p}(\pi)+2}{\frac{b_{\bar{\rho}_p}(\pi)}{2}}$. Para $b_{\bar{\rho}_p}(\pi)$ suficientemente grande, temos

$$\lim_{b_{\bar{\rho}_p}(\pi)\to\infty} \frac{b_{\bar{\rho}_p}(\pi) + 2}{\frac{b_{\bar{\rho}_p}(\pi)}{2}} = \lim_{b_{\bar{\rho}_p}(\pi)\to\infty} \frac{2b_{\bar{\rho}_p}(\pi) + 4}{b_{\bar{\rho}_p}(\pi)} = 2 + \lim_{b_{\bar{\rho}_p}(\pi)\to\infty} \frac{4}{b_{\bar{\rho}_p}(\pi)} = 2 + \epsilon.$$

□

Algoritmo 9 Ordenando π_n^7 com reversões de prefixo com sinal e transposições de prefixo

SigPRPT_Familia_7 ($\pi = \pi_n^7$, $\mathtt{n} \geq 2$)
1 $\pi \leftarrow \pi \cdot \tau_p(2, \lfloor \frac{\mathtt{n}}{2} \rfloor + 1)$
2 **for** $\mathtt{j} \leftarrow \lfloor \frac{\mathtt{n}}{2} \rfloor - 1$ **downto** 2 **do**
3 $\quad \pi \leftarrow \pi \cdot \tau_p(2, \pi_{\mathtt{j}+1}^{-1})$
4 $\pi \leftarrow \pi \cdot \bar{\rho_p}(\mathtt{n})$
5 **for** $\mathtt{j} \leftarrow \mathtt{n}$ **downto** $\lfloor \frac{\mathtt{n}}{2} \rfloor + 1$ **do**
6 $\quad \pi \leftarrow \pi \cdot \tau_p(2, \pi_{\mathtt{j}+1}^{-1})$
7 $\pi \leftarrow \pi \cdot \bar{\rho_p}(\lfloor \frac{\mathtt{n}}{2} \rfloor)$

Lema 43. *Para $n \geq 2$, $\lceil \frac{n}{2} \rceil + 1 \leq d_{\bar{\rho_p}\tau_p}(\pi_n^7) \leq n + 1$.*

Demonstração. Primeiro note que $\lceil \frac{n}{2} \rceil$ é um limitante inferior válido, uma vez que $b_{\bar{\rho_p}}(\pi_n^7) = n$ e $d_{\bar{\rho_p}\tau_p}(\pi) \geq \left\lceil \frac{b_{\bar{\rho_p}}(\pi)}{2} \right\rceil$ para qualquer π. No entanto, esse limitante não é justo. Se fosse, quando n é par uma ordenação deveria ter apenas operações que removem dois *breakpoints*. Mas é fácil ver que isso não é possível logo na primeira operação. Quando n é ímpar, uma ordenação deveria ter $\lceil \frac{n}{2} \rceil - 1$ operações que removem dois *breakpoints* de uma vez e uma operação que remove um *breakpoint*. Remover dois *breakpoints* na primeira operação não é possível e a única forma de remover um é colocando $\lfloor \frac{n}{2} \rfloor$ depois de $\lfloor \frac{n}{2} \rfloor - 1$. Contudo, depois dessa operação não seria possível remover dois *breakpoints* também. Logo, a distância é pelo menos $\lceil \frac{n}{2} \rceil + 1$. O limitante superior é dado pelo Algoritmo 9. □

Teorema 44. *Para $n \geq 2$, $D_{\bar{\rho_p}\tau_p}(n) \geq \lceil \frac{n}{2} \rceil + 1$ e para $n \geq 16$, $D_{\bar{\rho_p}\tau_p}(n) \leq \frac{18n}{11} + O(1)$.*

Demonstração. O limitante inferior é verdadeiro devido à família π_n^7, como o Lema 43 mostra. O limitante superior é verdadeiro porque $D_{\bar{\rho_p}\tau_p}(n) \leq D_{\bar{\rho_p}}(n)$, uma vez que $d_{\bar{\rho_p}\tau_p}(\pi) \leq d_{\bar{\rho_p}}(\pi)$ para qualquer π. Note que uma sequência de ordenação para SbPT não é válida para SbSigPRPT sempre, porque transposições de prefixo sozinhas não conseguem lidar com sinais. □

A.9 Problema de Ordenação por Reversões de Prefixo e Reversões de Sufixo

Lema 45. *Seja π uma permutação qualquer. Então, $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$.*

Demonstração. Uma reversão de prefixo $\rho_p(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* apenas no último par, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de reversão de prefixo e reversão de sufixo. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \rho_p) \in \{-1, 0, 1\}$. Da mesma forma, uma reversão de sufixo $\rho_s(j)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas no primeiro, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de reversão de prefixo e reversão de sufixo. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \rho_s) \in \{-1, 0, 1\}$. □

Lema 46. *Seja π uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo duas reversões de prefixo ou de sufixo que pode remover um breakpoint de reversão de prefixo e reversão de sufixo existe se $G_b(\pi)$ contém pelo menos uma das oito arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 1 com $j \leq n$;

(2) (π_i, π_n) do tipo 2 com $i \geq 1$;

(3) (π_1, π_j) do tipo 3 com $j \leq n$;

(4) (π_i, π_n) do tipo 3 com $i \geq 1$;

(5) (π_i, π_j) do tipo 2 com $i \neq 0$ e $j \leq n$;

(6) (π_i, π_j) do tipo 1 com $j \neq n+1$ e $i \geq 1$;

(7) (π_i, π_j) do tipo 3 com $i > 1$ e $j \leq n$;

(8) (π_i, π_j) do tipo 3 com $j < n$ e $i \geq 1$.

Demonstração. Se (π_i, π_j) é uma aresta azul, então $\pi_j = \pi_i \pm 1$. Para criar uma adjacência entre π_i e π_j sem criar novos *breakpoints* basta fazer, para cada tipo de aresta:

(1) Uma reversão de prefixo $\rho_p(j-1)$, que une π_1 e π_j;

(2) Uma reversão de sufixo $\rho_s(i+1)$, que une π_i e π_n;

(3) Uma reversão de prefixo $\rho_p(j-1)$, que une π_1 e π_j;

(4) Uma reversão de sufixo $\rho_s(i+1)$, que une π_i e π_n;

(5) Uma reversão de prefixo $\rho_p(j)$, que gera $\pi' = (\pi_j \ \ldots \ \pi_i \ \ldots \ \pi_1 \ \pi_{j+1} \ \ldots \ \pi_n)$, seguida por outra reversão de prefixo $\rho_p(j-i)$ que une π_j e π_i. Note que π' contém a aresta azul $(\pi'_1 = \pi_j, \pi'_{j-i+1} = \pi_i)$, sobre a qual a segunda reversão age;

(6) Uma reversão de sufixo $\rho_s(i)$, que gera $\pi' = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_n \ \ldots \ \pi_j \ \ldots \ \pi_i)$, seguida por outra reversão de sufixo $\rho_s(n+1-(j-i))$ que une π_i e π_j. Note que π' contém a aresta azul $(\pi'_{n-(j-i)} = \pi_j, \pi'_n = \pi_i)$, sobre a qual a segunda reversão age;

(7) Uma reversão de prefixo $\rho_p(i)$, que gera $\pi' = (\pi_i \ \ldots \ \pi_1 \ \pi_{i+1} \ \ldots \ \pi_j \ \ldots \ \pi_n)$, seguida por outra reversão de prefixo $\rho_p(j-1)$ que une π_i e π_j. Note que π' contém a aresta azul $(\pi'_1 = \pi_i, \pi'_j = \pi_j)$, sobre a qual a segunda reversão age;

(8) Uma reversão de sufixo $\rho_s(j)$, que gera $\pi' = (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_{j-1} \ \pi_n \ \ldots \ \pi_j)$, seguida por outra reversão de sufixo $\rho_s(i+1)$ que une π_i e π_j. Note que π' contém a aresta azul $(\pi'_i = \pi_i, \pi'_n = \pi_j)$, sobre a qual a segunda reversão age.

□

Lema 47. *Seja π uma permutação qualquer. Então, $d_{\rho_s}(\pi) \geq b_{\rho_s}(\pi)$.*

Demonstração. Uma reversão de sufixo $\rho_s(j)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas no primeiro, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de reversão de sufixo. Assim, $\Delta b_{\rho_s}(\pi, \rho_s) \in \{-1, 0, 1\}$. □

Lema 48. *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de sufixo. Então, π não contém nenhum singleton.*

Demonstração. Suponha que π possui um *singleton* $\pi_i = k$. Então uma aresta vermelha termina em k e outra aresta vermelha começa em k. Além disso, duas arestas azuis também começam ou terminam em k. Se $k+1$ e $k-1$ estivessem à esquerda de k, então haveriam arestas boas do tipo 1 ou 3 em π, o que é uma contradição. Considere, sem perda de generalidade, $k+1$. Como já foi dito, ele deve estar à direita de k. Se uma aresta vermelha termina em $k+1$, então haveria uma aresta boa do tipo 3 em π. Então a única aresta vermelha que envolve $k+1$ começa ali. Logo, deve haver uma *strip* decrescente com início em $k+\ell$ e término em $k+1$, para algum $\ell \geq 1$, isto é, π é da forma $(\ldots\ldots\, k \,\ldots\ldots\, \underline{k+\ell \ldots k+1} \,\ldots\ldots)$ e uma aresta vermelha termina em $k+\ell$. Se $k+\ell = 1$, então haveria uma aresta do tipo 3 em π, uma vez que uma aresta vermelha começa em 0. Caso contrário, pelo mesmo motivo anterior, as arestas azuis que saem de $k+\ell$ devem estar para a direita. Novamente, $k+\ell+1$ tem uma aresta vermelha que começa nele. Seguindo esse raciocínio, eventualmente atingiremos 1, o que nos levaria a uma contradição. □

Lema 49. *Seja $\pi \neq \iota_n$ uma permutação sem sinal qualquer que não contém uma aresta boa de sufixo. Então, a primeira aresta de π é do tipo 2, a última é do tipo 1 e todas as outras são do tipo 4.*

Demonstração. Seja π_j o primeiro elemento da última *strip* de π. Devemos ter que $\pi_j = n$, caso contrário haveria um $\pi_i = \pi_j + 1$ com $i+1 < j$, o que é uma contradição, uma vez que uma aresta boa do tipo 1 ou 3 existiria. Como (π_n, π_{n+1}) é sempre uma aresta vermelha e $\pi_n \neq n$ porque não existem *singletons* de acordo com Lema 48, a última aresta azul, que termina em π_n, só pode ser do tipo 1 ou 3. Como não existem arestas boas, ela deve ser do tipo 1.

Note que as outras arestas só podem ser do tipo 2 ou 4 (caso contrário elas seriam boas). Note também que, como π não contém *singletons*, sempre existem 0 ou 2 arestas (uma vermelha e uma azul) incidentes a cada vértice de $G_b(\pi)$. Portanto, $G_b(\pi)$ forma um único ciclo. Seguindo esse ciclo a partir do começo da última aresta f, procuramos pela primeira aresta azul f' do tipo 2 que aparecer. Seja π_i o final da aresta vermelha que fica à esquerda de f'. A aresta azul que incide em π_j (deve haver uma) começa ali, pois se terminasse haveria uma aresta boa do tipo 1 ou 3 em π. Agora ou temos que $f' = f$, ou então o argumento pode continuar indutivamente até que eventualmente atinjamos f. Logo, f' deve ser a primeira aresta azul de π: uma aresta do tipo 2 deve existir, caso contrário não haveria um ciclo, e outro ciclo em π não pode existir, pois ele deveria terminar com uma aresta do tipo 1, o que seria uma contradição. □

Lema 50. *Seja $\pi \neq \iota_n$ uma permutação qualquer que não contém uma aresta boa de sufixo. Então ela é da forma*

$$(\underline{1\ 2\ \ldots\ t}\ \underbrace{p_1\ \ldots\ t+1}_{\ell_1}\ \underbrace{p_2\ \ldots\ p_1+1}_{\ell_2}\ \ldots\ldots\ \underbrace{n\ \ldots\ p_{b_{\rho_s}(\pi)-1}+1}_{\ell_{b_{\rho_s}(\pi)}}) \tag{8}$$

onde $t \geq 0$. Em outras palavras, π consiste em $b_{\rho_s}(\pi) \geq 2$ strips decrescentes de tamanho $\ell_i \geq 2$, para todo $1 \leq i \leq b_{\rho_s}(\pi)$, de tal forma que se tais strips fossem crescentes, a permutação seria igual à identidade.

Demonstração. Se $b_{\rho_p}(\pi) = 1$, então ou π tem duas *strips* e a primeira começa em 1, ou $\pi = \eta_n$. Em ambos os casos, uma aresta boa do tipo 2 existe. Por isso, $b_{\rho_p}(\pi) \geq 2$. Além disso, $\ell_i \geq 2$ para todo $1 \leq i \leq b_{\rho_p}(\pi)$ porque π não tem *singleton*, conforme o Lema 48 mostra.

Quando $b_{\rho_p}(\pi) = 2$, ou π possui duas *strips* ou então possui três *strips* sendo que a primeira começa em 1. Não é difícil perceber que, de todos os 10 possíveis formatos, as permutações que não possuem arestas boas são apenas aquelas nas formas ($\_\ \underline{k\ k{-}1\ \ldots\ 1}\ \_\ \underline{n\ n{-}1\ \ldots\ k{+}1}$) e ($\underline{1\ 2\ \ldots k}\ \_\ \underline{k{+}\ell\ k{+}\ell{-}1\ \ldots\ k{+}1}\ \_\ \underline{n\ n{-}1\ \ldots\ k{+}\ell{+}1}$), isto é, da mesma forma dada em (8).

Vamos escrever $b = b_{\rho_s}(\pi)$ por simplificação. Suponha que toda permutação sem arestas boas de sufixo com $b - 1 \geq 2$ *breakpoints* é da forma dada em (8).

Seja π uma permutação com $b > 2$ *breakpoints* sem arestas boas de sufixo. Seja π_w o primeiro elemento da última *strip* de π. Devemos ter que $\pi_w = n$, pois, pelo Lema 49, a última aresta é do tipo 1 e termina em π_{n+1}.

Seja π' uma permutação construída a partir de π tal que $\pi'_j = \pi_j$ para todo $1 \leq j \leq w - 1$. Note que, se π' tivesse arestas boas do tipo 1 ou 3, é fácil ver que π também as teria. Se π' tivesse uma aresta boa do tipo 2 (π'_i, π'_{w-1}), para algum $i{+}1 < w$, então em π ela seria (π_i, π_{w-1}) do tipo 2 (não boa). Mas, pelo Lema 49, sabemos que π só pode ter uma aresta do tipo 2, a primeira. Como π_{w-1} é o último elemento da penúltima *strip* de π, então (π_i, π_{w-1}) é a antepenúltima aresta azul de π (a última termina em π_{n+1} e a penúltima termina em π_n). Portanto, ela só pode ser a primeira de π se $b = 2$.

Dessa forma, π' não possui arestas boas. Além disso, π' tem $b - 1$ *breakpoints* de reversão de sufixo. Logo, pela hipótese de indução, π' é da forma dada em (8). Como π' é π renomeada sem a última *strip*, que tem o elemento n, segue que π também é da forma dada em (8). □

Lema 51. *Seja $\pi \neq \iota_n$ uma permutação qualquer de tamanho n sem arestas boas de sufixo que é da forma dada em (8). A sequência de $2b_{\rho_s}(\pi)$ reversões de sufixo*

$$\rho_s(t+1) \cdot \rho_s(t + \ell_{b_{\rho_s}(\pi)} + 1) \cdot \rho_s(t+1) \cdot \rho_s(t + \ell_{b_{\rho_s}(\pi)-1} + 1) \cdot \ldots \cdot \rho_s(t+1) \cdot \rho_s(t + \ell_1 + 1) \quad (9)$$

quando aplicada a π, transforma-a na permutação identidade.

Demonstração. É fácil perceber que qualquer sequência de reversões de sufixo que ordena ($\underline{1\ 2\ \ldots\ t}\ \underline{p_1\ \ldots\ t{+}1}\ \underline{p_2\ \ldots\ p_1{+}1} \ldots\ldots \underline{n\ \ldots\ p_{b_{\rho_s}(\pi)-1}{+}1}$) também irá ordenar ($\underline{p_1\ \ldots\ t{+}1}\ \underline{p_2\ \ldots\ p_1{+}1}$ $\ldots\ldots$ $\underline{n\ \ldots\ p_{b_{\rho_s}(\pi)-1}{+}1}$). Portanto, escreveremos π da última maneira, por simplificação. Também escreveremos $b = b_{\rho_s}(\pi)$, por simplificação.

Seja π^k, com $1 \leq k \leq b - 1$, a permutação que obtemos depois de aplicar as primeiras $2k$ reversões de sufixo da sequência dada em (9), isto é, depois de aplicar $\rho_s(t+1) \cdot \rho_s(t + \ell_b + 1) \cdot$

$\ldots \cdot \rho_s(t+1) \cdot \rho_s(t+\ell_{b-(k-1)}+1)$. Vamos mostrar por indução em k que

$$\pi^k = (\underbrace{p_{b-k}{+}1\ p_{b-k}{+}2 \ldots\ n}_{\ell_b+\ell_{b-1}+\ldots+\ell_{b-(k-1)}}\ \underbrace{p_1\ \ldots\ t{+}1}_{\ell_1}\ \ldots\ldots\ \underbrace{p_{b-(k+1)}\ \ldots\ p_{b-(k+2)}{+}1}_{\ell_{b-(k+1)}}\ \underbrace{p_{b-k}\ \ldots\ p_{b-(k+1)}{+}1)}_{\ell_{b-k}}.$$

Quando $k = 1$, as duas primeiras operações de (9) são $\rho_s(t+1)$ e $\rho_s(t+\ell_b+1)$ e temos que $\pi' = \pi \cdot \rho_s(t+1) = (\underline{p_{b-1}{+}1\ \ldots\ n}\ \underline{p_{b-2}{+}1\ \ldots\ p_{b-1}}\ \underline{p_{b-3}{+}1\ \ldots\ p_{b-2}}\ \ldots\ldots\ \underline{t{+}1\ \ldots\ p_1})$ e $\pi'' = \pi' \cdot \rho_s(t+\ell_b+1) = (\underline{p_{b-1}{+}1\ \ldots\ n}\ \underline{p_1\ \ldots\ t{+}1}\ \ldots\ldots\ \underline{p_{b-2}\ \ldots\ p_{b-3}{+}1}\ \underline{p_{b-1}\ \ldots\ p_{b-2}{+}1})$. É fácil ver que π'' é da mesma forma que π^1.

Agora assuma que π^{k-1} é da forma dada, isto é,

$$\pi^{k-1} = (\underbrace{p_{b-(k-1)}{+}1\ p_{b-(k-1)}{+}2 \ldots\ n}_{\ell_b+\ell_{b-1}+\ldots+\ell_{b-(k-2)}}\ \underbrace{p_1\ \ldots\ t{+}1}_{\ell_1}\ \ldots\ldots\ \underbrace{p_{b-k}\ \ldots\ p_{b-(k+1)}{+}1}_{\ell_{b-k}}\ \underbrace{p_{b-(k-1)}\ \ldots\ p_{b-k}{+}1)}_{\ell_{b-(k-1)}}.$$

É fácil ver que $\pi^k = \pi^{k-1} \cdot \rho_s(t+1) \cdot \rho_s(t+\ell_{b-(k-1)}+1)$. Portanto, o resultado segue. Ao fim, $\pi^{b-1} = (\underline{p_1{+}1\ p_1{+}2\ \ldots\ n}\ \underline{p_1\ \ldots\ t{+}1})$ e $\rho_s(t+1) \cdot \rho_s(t+\ell_1+1)$ a ordena.

□

`2-SR` e `2-SRg` são apresentados no Algoritmo 10, uma vez que a única diferença entre eles é a forma que percorrem a permutação.

Lema 52. *Seja π uma permutação qualquer. Então, $d_{\rho_s}(\pi) \leq 2b_{\rho_s}(\pi)$.*

Demonstração. Os algoritmos `2-SR` e `2-SRg` aplicam a sequência de reversões de sufixo dada por (9) quando não conseguem mais encontrar arestas boas de sufixo, as quais eles conseguem lidar com no máximo dois movimentos. Assim, no pior caso eles usam duas reversões de sufixo para remover um *breakpoint*. □

Teorema 53. SbSR *é 2-aproximável.*

Demonstração. Diretamente dos Lemas 47 e 52. □

Lema 54. *Seja $\pi \neq \iota_n$ uma permutação qualquer que não contém uma aresta boa. Então, π não contém nenhum singleton.*

Demonstração. Suponha que π possui um *singleton* $\pi_i = k$. Então uma aresta vermelha termina em k e outra aresta vermelha começa em k. Além disso, duas arestas azuis também começam ou terminam em k. Se $k+1$ e $k-1$ estivessem à direita de k, então haveriam arestas boas de prefixo do tipo 2 ou 3 em π, o que é uma contradição. Se $k+1$ e $k-1$ estivessem à esquerda de k, então haveriam arestas boas de sufixo do tipo 1 ou 3 em π, o que também é uma contradição. □

Algoritmo 10 Algoritmo de 2-aproximação para SbSR

2-SR/2-SRg(π, $\mathtt{n}$)

```
1  d ← 0
2  while π ≠ ι_n do
3      // Tenta remover um breakpoint com uma reversão
4      if G_b(π) tem uma aresta boa de sufixo (π_i, π_n) do tipo 2 then
5          π ← π · ρ_s(i + 1)
6          d ← d + 1
7      else if G_b(π) tem uma aresta boa de sufixo (π_i, π_n) do tipo 3 then
8          π ← π · ρ_s(i + 1)
9          d ← d + 1
10     // Tenta remover um breakpoint com duas reversões
11     else if G_b(π) tem uma aresta boa de sufixo (π_i, π_j) do tipo 1 then
12         π ← π · ρ_s(i) · ρ_s(n + 1 − (j − i))
13         d ← d + 2
14     else if G_b(π) tem uma aresta boa de sufixo (π_i, π_j) do tipo 3 and j < n then
15         π ← π · ρ_s(j) · ρ_s(i + 1)
16         d ← d + 2
17     // Forma especial
18     else
19         Seja t + 1 o menor elemento fora de posição em π
20         Seja ℓ_i o tamanho da i-ésima strip de π, desconsiderando a primeira
           se esta começar em 1
21         d ← d + 2b_{ρ_s}(π)
22         π ← π · ρ_s(t + 1) · ρ_s(t + ℓ_{b_{ρ_s}(π)} + 1) · ρ_s(t + 1) · ρ_s(t + ℓ_{b_{ρ_s}(π)−1} + 1) · ...
           ... · ρ_s(t + 1) · ρ_s(t + ℓ_1 + 1)
23 return d
```

Lema 55. *Seja $\pi \neq \iota_n$ uma permutação qualquer que não contém uma aresta boa. Então, π é de uma das três formas:*

$$\eta_n$$
$$\sigma^1 = (\underbrace{p_1 \ \dots \ 1}_{\ell_1} \ \underbrace{p_2 \ \dots \ p_1+1}_{\ell_2} \ \dots\dots \ \underbrace{n \ \dots \ p_{b_{\rho_p\rho_s}(\pi)}+1}_{\ell_{b_{\rho_p\rho_s}(\pi)+1}})$$
$$\sigma^2 = (\underbrace{p_{b_{\rho_p\rho_s}(\pi)}+1 \ \dots \ n}_{\ell_{b_{\rho_p\rho_s}(\pi)+1}} \ \underbrace{p_{b_{\rho_p\rho_s}(\pi)-1}+1 \ \dots \ p_{b_{\rho_p\rho_s}(\pi)}}_{\ell_{b_{\rho_p\rho_s}(\pi)}} \ \dots\dots \ \underbrace{1 \ \dots \ p_1}_{\ell_1})$$

(25)

onde $b_{\rho_p\rho_s}(\pi) \geq 1$ e $\ell_i \geq 2$ para todo $1 \leq i \leq b_{\rho_p\rho_s}(\pi)+1$. Em outras palavras, ou π é a permutação reversa, ou é formada por $b_{\rho_p\rho_s}(\pi)+1$ strips decrescentes de tal forma que se fossem crescentes a permutação seria igual à identidade, ou então π é formada por $b_{\rho_p\rho_s}(\pi)+1$ strips crescentes de tal forma que se fossem decrescentes a permutação seria igual à reversa.

Demonstração. Se $\pi = \eta_n$, então π tem apenas duas arestas vermelhas $(0, n)$ e $(1, n+1)$ e duas arestas azuis $(0, 1)$ e $(n, n+1)$, que não são consideradas arestas boas devido às restrições sobre os índices que o Lema 46 mostra.

Se $b_{\rho_p\rho_s}(\pi) = 1$, então π é formada por duas *strips* apenas, porque (π_0, π_1) e (π_n, π_{n+1}) nunca são considerados *breakpoints* de reversão de prefixo e reversão de sufixo. De todas as 6 formatos possíveis, as únicas permutações que não possuem arestas boas são aquelas nas formas $(\underline{k \ k{-}1 \ \dots \ 1} \ \underline{n \ n{-}1 \ \dots \ k{+}1})$ e $(\underline{k{+}1 \ k{+}2 \ \dots \ n} \ \underline{1 \ 2 \ \dots \ k})$, isto é, da forma σ^1 e σ^2, respectivamente.

Agora suponha que toda permutação $\pi \neq \eta_n$ sem arestas boas com $b-1$ *strips* ou é da forma de σ^1 ou é da forma de σ^2.

Seja π uma permutação com b *strips* sem arestas boas e seja π_w o último elemento da primeira *strip* de π. Então ou $\pi_w = 1$, pois caso contrário haveria algum $\pi_i = \pi_w - 1$ com $i > w$ e uma aresta boa de prefixo do tipo 2 ou 3 existiria, ou então $\pi_w = n$, pois caso contrário haveria algum $\pi_i = \pi_w + 1$ com $i > w$ e uma aresta boa de prefixo do tipo 2 ou 3 existiria. Note que 0 e $n+1$ não formam arestas boas com 1 e n, respectivamente.

Suponha então que $\pi_w = 1$. Seja π' a permutação construída a partir de π tal que $\pi'_j = \pi_{j+w} - \pi_1$ para todo $1 \leq j \leq n-w$. Note que π' tem $b-1$ *strips*. Além disso, é fácil ver que se π' tivesse uma aresta boa, então π também as teria. Logo, pela hipótese de indução, π' é da forma de σ^1 ou de σ^2. Como π' é π renomeado sem a primeira *strip*, que tem o elemento 1, se $\pi' = \sigma^2$, então $\pi = (\underline{p_1 \ \dots \ 1} \ \underline{p_b+1 \ \dots \ n} \ \dots\dots \ \underline{p_1+1 \ \dots \ p_2})$. Mas nesse caso, uma aresta boa de prefixo do tipo 1 existiria, o que não é possível. Logo, $\pi' = \sigma^1$ e $\pi = \sigma^1$ também.

Suponha agora que $\pi_w = n$. Seja π' a permutação construída a partir de π tal que $\pi'_j = \pi_j$ para todo $1 \leq j \leq n-w$. Note que π' tem $b-1$ *strips*. Além disso, é fácil ver que se π' tivesse uma aresta boa, então π também as teria. Logo, ou π' é da forma de σ^1 ou de σ^2. Como π' é π sem a primeira *strip*, que tem o elemento n, se $\pi' = \sigma^1$, então $\pi = (\underline{p_b+1 \ \dots \ n} \ \underline{p_1 \ \dots \ 1}$

$\ldots\ldots\ \underline{p_b\ \ldots\ p_{b-1}+1})$. Mas nesse caso, uma aresta boa de prefixo do tipo 1 existiria, o que não é possível. Logo, $\pi' = \sigma^2$ e $\pi = \sigma^2$ também. □

Lema 56. *Seja π uma permutação sem sinal de n elementos sem arestas boas de uma das formas dadas no Lema 55. Se $\pi = \eta_n$, uma reversão $\rho_p(n)$ a ordena. Caso contrário, no máximo $b_{\rho_p\rho_s}(\pi) + 2$ movimentos ordenam π.*

Demonstração. Vamos denotar $b_{\rho_p\rho_s}(\pi)$ como b, por simplificação.

Se $\pi = \sigma^1$ e b é um número ímpar, então as $b+1$ reversões

$$\rho_s(\ell_1+1)\cdot\rho_p(n-\ell_2)\cdot\rho_s(\ell_3+1)\cdot\rho_p(n-\ell_4)\cdot\ldots\ldots\cdot\rho_s(\ell_b+1)\cdot\rho_p(n-\ell_{b+1}) \tag{26}$$

transformam π na permutação identidade, como iremos demonstrar a seguir.

Seja π^k, $1 \le k \le \frac{b-1}{2}$, a permutação obtida depois que as primeiras $2k$ reversões da sequência dada acima são aplicadas, isto é, depois que $\rho_s(\ell_1+1)\cdot\rho_p(n-\ell_2)\cdot\ldots\cdot\rho_s(\ell_{2k-1}+1)\cdot\rho_p(n-\ell_{2k})$ é aplicada. Vamos mostrar por indução em k que

$$\pi^k = (\underbrace{p_{2k+1}\ \ldots\ p_{2k}+1}_{\ell_{2k+1}}\ \underbrace{p_{2k+2}\ \ldots\ p_{2k+1}+1}_{\ell_{2k+2}}\ \ldots\ldots\ \underbrace{n\ \ldots\ p_b+1}_{\ell_{b+1}}$$
$$\underbrace{1\ 2\ \ldots\ p_{2k-2}+1\ \ldots\ p_{2k-1}\ p_{2k-1}+1\ \ldots\ p_{2k}}_{\ell_1+\ell_2+\ldots\ell_{2k}})\ .$$

Quando $k=1$, as $2k$ primeiras reversões de (26) são $\rho_s(\ell_1+1)\cdot\rho_p(n-\ell_2)$, e temos que $\pi' = \pi\cdot\rho_s(\ell_1+1) = (\underline{p_1\ \ldots\ 1\ p_b+1\ \ldots\ n}\ \ldots\ldots\ \underline{p_3+1\ \ldots\ p_4\ p_2+1\ \ldots\ p_3\ p_1+1\ \ldots\ p_2})$ e $\pi'' = \pi'\cdot\rho_p(n-\ell_2) = (\underline{p_3\ \ldots\ p_2+1\ p_4\ \ldots\ p_3+1}\ \ldots\ldots\ \underline{n\ \ldots\ p_b+1\ 1\ \ldots\ p_1\ p_1+1\ \ldots\ p_2})$. É fácil ver que π'' é da forma π^1.

Agora assuma que π^{k-1} é da forma dada acima, isto é

$$\pi^{k-1} = (\underbrace{p_{2k-1}\ \ldots\ p_{2k-2}+1}_{\ell_{2k-1}}\ \underbrace{p_{2k}\ \ldots\ p_{2k-1}+1}_{\ell_{2k}}\ \ldots\ldots\ \underbrace{n\ \ldots\ p_b+1}_{\ell_{b+1}}$$
$$\underbrace{1\ 2\ \ldots\ p_{2k-4}+1\ \ldots\ p_{2k-3}\ p_{2k-3}+1\ \ldots\ p_{2k-2}}_{\ell_1+\ell_2+\ldots\ell_{2k-2}})\ .$$

Como $\pi^k = \pi^{k-1}\cdot\rho_s(\ell_{2k-1}+1)\cdot\rho_p(n-\ell_{2k})$, o resultado segue. Ao fim, $\pi^{(b-1)/2} = (\underline{p_b\ \ldots\ p_{b-1}+1\ n\ \ldots\ p_b+1\ 1\ 2\ \ldots\ p_{b-1}})$ e $\rho_s(\ell_b+1)\cdot\rho_p(n-\ell_{b+1})$ a ordena.

Se $\pi = \sigma^2$ e b é um número ímpar, deve-se aplicar $\rho_p(n)$ para transformá-la em σ^1 e então aplicar as $b+1$ reversões dadas em (26).

Se $\pi = \sigma^2$ e b é um número par, então as $b+1$ reversões

$$\rho_p(n-\ell_1)\cdot\rho_s(\ell_2+1)\cdot\rho_p(n-\ell_3)\cdot\rho_s(\ell_4+1)\cdot\ldots\ldots\cdot\rho_p(n-\ell_{b-1})\cdot\rho_s(\ell_b+1)\cdot\rho_p(n-\ell_{b+1}). \tag{27}$$

transformam π na permutação identidade, como iremos demonstrar a seguir.

Seja π^k, $1 \leq k \leq \frac{b}{2}$, a permutação obtida depois que as primeiras $2k$ reversões da sequência dada acima são aplicadas, isto é, depois que $\rho_p(n-\ell_1)\cdot\rho_s(\ell_2+1)\cdot\ldots\cdot\rho_p(n-\ell_{2k-1})\cdot\rho_s(\ell_{2k}+1)$ é aplicada. Vamos mostrar por indução em k que

$$\pi^k = (\underbrace{p_{2k}\ \ldots\ p_{2k-1}{+}1\ p_{2k-1}\ \ldots\ p_{2k-2}{+}1\ \ldots\ldots\ p_1\ \ldots\ 1}_{\ell_1+\ell_2+\ldots+\ell_{2k}}\ \underbrace{p_b{+}1\ \ldots\ n}_{\ell_{b+1}}\ \ldots$$
$$\ldots\underbrace{p_{2k+1}{+}1\ \ldots\ p_{2k+2}}_{\ell_{2k+2}}\underbrace{p_{2k}{+}1\ \ldots\ p_{2k+1}}_{\ell_{2k+1}})\ .$$

Quando $k=1$, as $2k$ primeiras reversões de (27) são $\rho_p(n-\ell_1)\cdot\rho_s(\ell_2+1)$ e temos que $\pi' = \pi\cdot\rho_p(n-\ell_1) = (\underline{p_2\ \ldots\ p_1+1}\ \underline{p_3\ \ldots\ p_2+1}\ \underline{p_4\ \ldots\ p_3+1}\ \ldots\ldots\ \underline{n\ \ldots\ p_b+1})$ e $\pi'' = \pi'\cdot\rho_s(\ell_2+1) = (\underline{p_2\ \ldots\ p_1+1\ p_1\ \ldots\ 1}\ \underline{p_b+1\ \ldots\ n}\ \ldots\ldots\ \underline{p_3+1\ \ldots\ p_4}\ \underline{p_2+1\ \ldots\ p_3})$. É fácil ver que π'' é da forma π^1.

Agora assuma que π^{k-1} é da forma dada acima, isto é

$$\pi^{k-1} = (\underbrace{p_{2k-2}\ \ldots\ p_{2k-3}{+}1\ p_{2k-3}\ \ldots\ p_{2k-4}{+}1\ \ldots\ldots\ p_1\ \ldots\ 1}_{\ell_1+\ell_2+\ldots+\ell_{2k-2}}$$
$$\underbrace{p_b{+}1\ \ldots\ n}_{\ell_{b+1}}\ \ldots\ldots\underbrace{p_{2k-1}{+}1\ \ldots\ p_{2k}}_{\ell_{2k}}\underbrace{p_{2k-2}{+}1\ \ldots\ p_{2k-1}}_{\ell_{2k-1}})\ .$$

Como $\pi^k = \pi^{k-1}\cdot\rho_p(n-\ell_{2k-1})\cdot\rho_s(\ell_{2k}+1)$, o resultado segue. Ao fim, $\pi^{(b-1)/2} = (\underline{p_b\ p_b-1\ \ldots\ 1}\ \underline{p_b+1\ \ldots\ n})$ e $\rho_p(n-\ell_{b+1})$ a ordena.

Se $\pi=\sigma^1$ e b é um número par, deve-se aplicar $\rho_p(n)$ para transformá-la em σ^2 e então aplicar as $b+1$ reversões dadas em (27). □

O Algoritmo 11, representa ambos `2-PRSR` e `2-PRSRg`, uma vez que a única diferença entre eles é como eles percorrem a permutação.

Lema 57. *Seja π uma permutação qualquer. Então, $d_{\rho_p\rho_s}(\pi) \leq 2b_{\rho_p\rho_s}(\pi)+1$.*

Demonstração. No pior caso, duas reversões de prefixo ou de sufixo são necessárias para eliminar um *breakpoint*. Além disso, a permutação identidade ou a permutação reversa podem ser atingidas. No último caso, uma operação extra ainda é necessária para terminar a ordenação. □

Teorema 58. SbPRSR *é 2-aproximável assintoticamente.*

Demonstração. Dos Lemas 45 e 57, temos que o fator de aproximação teórico é $\frac{2b_{\rho_p\rho_s}(\pi)+1}{b_{\rho_p\rho_s}(\pi)}$. Para $b_{\rho_p\rho_s}(\pi)$ suficientemente grande, temos:

$$\lim_{b_{\rho_p\rho_s}(\pi)\to\infty}\frac{2b_{\rho_p\rho_s}(\pi)+1}{b_{\rho_p\rho_s}(\pi)} = 2+\lim_{b_{\rho_p\rho_s}(\pi)\to\infty}\frac{1}{b_{\rho_p\rho_s}(\pi)} = 2+\epsilon.$$

□

Algoritmo 11 Algoritmo assintótico de 2-aproximação para SbPRSR

2-PRSR/2-PRSRg(π, n)

```
 1  d ← 0
 2  while  π ≠ ι_n  do
 3      // Tenta remover um breakpoint com uma operação
 4      if  G(π) contém uma ABP (π_1, π_j) do tipo 1  and j ≤ n  then
 5          π ← π · ρ_p(j − 1)
 6          d ← d + 1
 7      else if G(π) contém uma ABS (π_i, π_n) do tipo 2  and i ≥ 1  then
 8          π ← π · ρ_s(i + 1)
 9          d ← d + 1
10      else if G(π) contém uma ABP (π_1, π_j) do tipo 3  and j ≤ n  then
11          π ← π · ρ_p(j − 1)
12          d ← d + 1
13      else if G(π) contém uma ABS (π_i, π_n) do tipo 3  and i ≥ 1  then
14          π ← π · ρ_s(i + 1)
15          d ← d + 1
16      // Tenta remover um breakpoint com duas operações
17      else if G(π) contém uma ABP (π_i, π_j) do tipo 2  and i ≠ 0  and j ≤ n  then
18          π ← π · ρ_p(j) · ρ_p(j − i)
19          d ← d + 2
20      else if G(π) contém uma ABS (π_i, π_j) do tipo 1  and j ≠ n + 1  and i ≥ 1  then
21          π ← π · ρ_s(i) · ρ_s(n + 1 − (j − i))
22          d ← d + 2
23      else if G(π) contém uma ABP (π_i, π_j) do tipo 3  and i > 1  and j ≤ n  then
24          π ← π · ρ_p(i) · ρ_p(j − 1)
25          d ← d + 2
26      else if G(π) contém uma ABS (π_i, π_j) do tipo 3  and j < n  and i ≥ 1  then
27          π ← π · ρ_s(j) · ρ_s(i + 1)
28          d ← d + 2
29      else if π = η_n  then
30          π ← π · ρ_p(n)
31          d ← d + 1
32      // Formas especiais
33      else
34          b ← b_{ρ_p ρ_s}(π)
35          if  b mod 2 ≡ 1  then
36              if  π = (p_1 ... 1 p_2 ... p_1 + 1 ..... n ... p_b + 1)  then
37                  π ← π · ρ_p(n)
38                  d ← d + 1
39              Seja ℓ_i o tamanho da i-ésima strip de π
40              π ← π · ρ_s(ℓ_1 + 1) · ρ_p(n − ℓ_2) · ..... · ρ_s(ℓ_b + 1) · ρ_p(n − ℓ_{b+1})
41          else
42              if  π = (p_b + 1 ... n p_{b−1} ... p_b ..... 1 ... p_1)  then
43                  π ← π · ρ_p(n)
44                  d ← d + 1
45              Seja ℓ_{b+2−i} o tamanho da i-ésima strip de π
46              π ← π · ρ_p(n − ℓ_1) · ρ_s(ℓ_2 + 1) · ..... · ρ_p(n − ℓ_{b−1}) · ρ_s(ℓ_b + 1) · ρ_p(n − ℓ_{b+1})
47          d ← d + b + 1
48  return  d
```

Algoritmo 12 Ordenando π_n^{11} com reversões de prefixo e reversões de sufixo

PRSR_Familia_11 ($\pi = \pi_n^{11}$, $\mathtt{n} \geq 8$)

1 $\pi \leftarrow \pi \cdot \rho_p(\mathtt{n}-1) \cdot \rho_p(\mathtt{n}-3) \cdot \rho_s(2) \cdot \rho_s(\pi_{\pi_n+1}^{-1}+1) \cdot \rho_p(\pi_{\pi_1-1}^{-1}-1) \cdot \rho_s(\pi_{\mathtt{n}}^{-1})$
2 **while** $\pi_1 \neq 1$ **do**
3 $\quad \pi \leftarrow \pi \cdot \rho_p(\pi_{\pi_1+1}^{-1}-1)$

Lema 59. *Para $n \geq 8$, $n-1 \leq d_{\rho_p\rho_s}(\pi_n^{11}) \leq n$.*

Demonstração. O limitante inferior é verdadeiro porque $b_{\rho_p\rho_s}(\pi_n^{11}) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para todo π. O limitante superior é verdadeiro por causa do Algoritmo 12. □

Teorema 60. *Para $n \geq 8$, $D_{\rho_p\rho_s}(n) \geq n-1$ e para $n \geq 1$, $D_{\rho_p\rho_s}(n) \leq \frac{18n}{11} + O(1)$.*

Demonstração. O limitante inferior é verdadeiro devido à família π_n^{11}, como o Lema 59 mostra. O limitante superior é verdadeiro porque $D_{\rho_p\rho_s}(n) \leq D_{\rho_p}(n)$, uma vez que $d_{\rho_p\rho_s}(\pi) \leq d_{\rho_p}(\pi)$ para todo π. Isso é verdade porque qualquer sequência de ordenação para SbPR é válida para SbPRSR, mas não necessariamente ótima. □

A.10 Problema de Ordenação por Reversões de Prefixo com Sinal e Reversões de Sufixo com Sinal

Lema 61. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}\bar{\rho_s}}(\pi) \geq b_{\bar{\rho_p}\bar{\rho_s}}(\pi)$.*

Demonstração. Uma reversão de prefixo com sinal $\bar{\rho_p}(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* apenas no último par, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de reversão de prefixo com sinal e reversão de sufixo com sinal. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \bar{\rho_p}) \in \{-1, 0, 1\}$. De forma equivalente, uma reversão de sufixo $\bar{\rho_s}(j)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas no primeiro par, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint*. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \bar{\rho_s}) \in \{-1, 0, 1\}$. □

Lema 62. *Seja $\pi \neq \iota_n$ uma permutação com sinal na qual não se pode remover um breakpoint com uma ou duas operações. Então, π é de uma das três formas:*

$$\begin{aligned}
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \bar{\eta_n} \\
\sigma^1 &= (\underbrace{p_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}+1 \ \ldots \ n}_{\ell_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1}} \ \underbrace{p_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)-1}+1 \ \ldots \ p_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}}_{\ell_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}} \ \ldots\ldots \ \underbrace{1 \ \ldots \ p_1}_{\ell_1}) \\
\sigma^2 &= (\underbrace{-p_1 \ \ldots \ -1}_{\ell_1} \ \underbrace{-p_2 \ \ldots \ -(p_1+1)}_{\ell_2} \ \ldots\ldots \ \underbrace{-n \ \ldots \ -(p_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}+1)}_{\ell_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1}}).
\end{aligned} \tag{28}$$

onde $\ell_i \geq 1$ para todo $1 \leq i \leq b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1$ e $b_{\bar{\rho_p}\bar{\rho_s}}(\pi) \geq 1$. Em outras palavras, ou π é a permutação reversa com sinal, ou é formada por $b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1$ strips positivas de tal forma que

se elas fossem negativas a permutação seria igual à reversa com sinal, ou então π é formada por $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+1$ strips negativas de tal forma que se fossem positivas a permutação seria igual à identidade.

Demonstração. É fácil perceber que quando $\pi = \bar{\eta}_n$ os quatro primeiros casos principais do algoritmo não conseguem transformá-la na permutação identidade, já que a reversão $\bar{\rho}_p(n)$ é necessária e ela não seria considerada como uma reversão que remove um *breakpoint* de reversão de prefixo com sinal e reversão de sufixo com sinal. De fato, ela não remove um *breakpoint*, porque $b_{\bar{\rho}_p\bar{\rho}_s}(\bar{\eta}_n)$ já é igual a zero, por definição.

Seja $\pi_i = k$ qualquer elemento de π. Se $\pi_j = -(k+1)$ existe e $i < j$, então $\bar{\rho}_p(j) \cdot \bar{\rho}_p(j-i)$ remove um *breakpoint*. Se $j < i$, então $\bar{\rho}_p(i) \cdot \bar{\rho}_p(i-j)$ remove um *breakpoint*. De forma similar, se $\pi_j = -(k-1)$ existe em π e $i < j$, então $\bar{\rho}_s(i) \cdot \bar{\rho}_s(n+1-(j-i))$ remove um *breakpoint*. Se $j < i$, então $\bar{\rho}_s(j) \cdot \bar{\rho}_s(n+1-(i-j))$ remove um *breakpoint*. Portanto, os elementos de π devem ser todos de mesmo sinal.

Suponha agora que π tem apenas elementos positivos. Se $b_{\bar{\rho}_p\bar{\rho}_s}(\pi) = 1$, então π é formada por duas *strips* apenas, porque (π_0, π_1) e (π_n, π_{n+1}) nunca são considerados *breakpoints* de reversão de prefixo com sinal e reversão de sufixo com sinal. É fácil ver que ela deve ser da forma de σ^1 ou σ^2.

Assuma que toda permutação com $b-1$ *strips* de mesmo sinal para a qual não é possível remover um *breakpoint* com uma ou duas operações é da forma de σ^1 ou é da forma de σ^2.

Seja π uma permutação com b *strips* de mesmo sinal para a qual não é possível remover um *breakpoint* com uma ou duas operações e seja π_w o último elemento da primeira *strip* de π. Então ou $\pi_w = n$ ou $\pi_w = -1$, pois caso contrário haveria algum $\pi_i = \pi_w + 1$ com $i > w+1$ e $\bar{\rho}_p(w) \cdot \bar{\rho}_p(i-w)$ removeria um *breakpoint*.

Suponha então que $\pi_w = n$. Seja π' a permutação construída a partir de π tal que $\pi'_j = \pi_{j+w}$ para todo $1 \leq j \leq n-w$. Note que π' tem $b-1$ *strips*. Além disso, é fácil ver que se fosse possível remover um *breakpoint* em π', então também seria possível em π. Portanto, pela hipótese de indução, π' ou é da forma de σ^1 ou de σ^2. Como π' é π renomeado sem a primeira *strip*, que tem o elemento n, se $\pi = \sigma^2$, então π consistiria em elementos positivos e negativos, o que não é possível. Logo, $\pi' = \sigma^1$ e $\pi = \sigma^1$ também.

Suponha agora que $\pi_w = -1$. Seja π' a permutação construída a partir de π tal que $\pi'_j = \pi_{j+w} + \pi_1$ para todo $1 \leq j \leq n-w$. Note que π' tem $b-1$ *strips*. Além disso, é fácil ver que se fosse possível remover um *breakpoint* em π', então também seria possível em π. Portanto, π' ou é da forma de σ^1 ou de σ^2. Como π' é π renomeado sem a primeira *strip*, que tem o elemento -1, se $\pi = \sigma^1$, então π consistiria em elementos positivos e negativos, o que não é possível. Logo, $\pi' = \sigma^2$ e $\pi = \sigma^2$ também. □

Lema 63. *Seja π uma permutação com sinal de n elementos de uma das formas descritas no Lema 62. Se $\pi = \bar{\eta}_n$, então uma reversão $\bar{\rho}_p(n)$ a ordena. Caso contrário, no máximo $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)+2$ operações ordenam π.*

Demonstração. Vamos denotar $b_{\bar{\rho}_p\bar{\rho}_s}(\pi)$ como b, por simplificação.

Se $\pi = \sigma^1$ e b é um número par, então as $b+1$ reversões

$$\bar{\rho}_p(n-\ell_1)\cdot\bar{\rho}_s(\ell_2+1)\cdot\bar{\rho}_p(n-\ell_3)\cdot\bar{\rho}_s(\ell_4+1)\cdot\ldots\cdot\bar{\rho}_p(n-\ell_{b-1})\cdot\bar{\rho}_s(\ell_b+1)\cdot\bar{\rho}_p(n-\ell_{b+1}) \quad (29)$$

ordenam π, como iremos demonstrar a seguir.

Seja π^k, $1 \leq k \leq \frac{b}{2}$, a permutação que obtemos depois de aplicar as $2k$ primeiras reversões da sequência dada acima, isto é, depois de aplicar $\bar{\rho}_p(n-\ell_1)\cdot\bar{\rho}_s(\ell_2+1)\cdot\ldots\cdot\bar{\rho}_p(n-\ell_{2k-1})\cdot\bar{\rho}_s(\ell_{2k}+1)$. Vamos demonstrar por indução sobre k que

$$\pi^k = (\underbrace{-p_{2k}\ \ldots\ -(p_{2k-1}+1)\ -p_{2k-1}\ \ldots -(p_{2k-2}+1)\ \ldots\ldots\ -p_1\ \ldots\ -1}_{\ell_{2k}+\ell_{2k-1}+\ldots+\ell_1}$$
$$\underbrace{p_b+1\ \ldots\ n}_{\ell_{b+1}}\ \ldots\ldots\ \underbrace{p_{2k+1}+1\ \ldots\ p_{2k+2}}_{\ell_{2k+2}}\ \underbrace{p_{2k}+1\ \ldots\ p_{2k+1}}_{\ell_{2k+1}})\ .$$

Quando $k=1$, as $2k$ primeiras reversões de (29) são $\bar{\rho}_p(n-\ell_1)\cdot\bar{\rho}_s(\ell_2+1)$ e temos que $\pi' = \pi\cdot\bar{\rho}_p(n-\ell_1) = (\underline{-p_2\ \ldots\ -(p_1+1)}\ \underline{-p_3\ \ldots\ -(p_2+1)}\ \underline{-p_4\ \ldots\ -(p_3+1)}\ \ldots\ldots\ \underline{-n\ \ldots\ -(p_b+1)}\,\underline{1\ \ldots\ p_1})$ e $\pi'' = \pi'\cdot\bar{\rho}_s(\ell_2+1) = (\underline{-p_2\ \ldots\ -(p_1+1)}\ \underline{-p_1\ \ldots\ -1}\,\underline{p_b+1\ \ldots\ n}\ \ldots\ldots\ \underline{p_4\ \ldots\ p_3+1}\ \underline{p_3\ \ldots\ p_2+1})$. É fácil ver que π'' é da forma π^1.

Agora assuma que π^{k-1} é da forma dada acima, isto é

$$\pi^{k-1} = (\underbrace{-p_{2k-2}\ \ldots\ -(p_{2k-3}+1)\ -p_{2k-3}\ \ldots -(p_{2k-4}+1)\ \ldots\ldots\ -p_1\ldots -1}_{\ell_{2k-2}+\ell_{2k-3}+\ldots+\ell_1}$$
$$\underbrace{p_b+1\ \ldots\ n}_{\ell_{b+1}}\ \ldots\ldots\ \underbrace{p_{2k-1}+1\ \ldots\ p_{2k}}_{\ell_{2k}}\ \underbrace{p_{2k-2}+1\ \ldots\ p_{2k-1}}_{\ell_{2k-1}})\ .$$

É fácil ver que $\pi^k = \pi^{k-1}\cdot\bar{\rho}_p(n-\ell_{2k-1})\cdot\bar{\rho}_s(\ell_{2k}+1)$. Portanto, o resultado segue. Ao fim, quando $k=\frac{b}{2}$, $\pi^{b/2} = (\underline{-p_b\ -(p_b-1)\ \ldots\ -1}\ \underline{p_b+1\ \ldots\ n})$ e a última reversão da sequência, $\bar{\rho}_p(n-\ell_{b+1})$, ordena $\pi^{b/2}$.

Se $\pi = \sigma^2$ e b é um número par, deve-se aplicar $\bar{\rho}_p(n)$ para transformá-la em σ^1 e então aplicar as $b+1$ reversões dadas em (29).

Se $\pi = \sigma^2$ e b é um número ímpar, então as $b+1$ reversões

$$\bar{\rho}_s(\ell_1+1)\cdot\bar{\rho}_p(n-\ell_2)\cdot\bar{\rho}_s(\ell_3+1)\cdot\bar{\rho}_p(n-\ell_4)\cdot\ldots\cdot\bar{\rho}_s(\ell_b+1)\cdot\bar{\rho}_p(n-\ell_{b+1}) \quad (30)$$

ordenam π, como mostraremos a seguir.

Seja π^k, $1 \leq k \leq \frac{b-1}{2}$, a permutação que obtemos depois de aplicar as $2k$ primeiras reversões da sequência dada acima, isto é, depois de aplicar $\bar{\rho}_s(\ell_1+1)\cdot\bar{\rho}_p(n-\ell_2)\cdot\ldots\cdot\bar{\rho}_s(\ell_{2k-1}+1)\cdot\bar{\rho}_p(n-\ell_{2k})$.

Vamos demonstrar por indução sobre k que

$$\pi^k = (\underbrace{-p_{2k+1} \ \dots \ -(p_{2k}+1)}_{\ell_{2k+1}} \ \underbrace{-p_{2k+2} \ \dots \ -(p_{2k+1}+1)}_{\ell_{2k+2}} \ \dots$$
$$\dots \ \underbrace{-n \ \dots \ -(p_b+1)}_{\ell_{b+1}} \ \underbrace{1 \ 2 \ \dots \ p_{2k}}_{\ell_1+\ell_2+\dots+\ell_{2k}})$$

Quando $k = 1$, as $2k$ primeiras reversões de (30) são $\bar{\rho_s}(\ell_1+1) \cdot \bar{\rho_p}(n-\ell_2)$ e temos que $\pi' = \pi \cdot \bar{\rho_s}(\ell_1+1) = (\underline{-p_1 \ \dots \ -1} \ \underline{p_b+1 \ \dots \ n} \ \dots\dots \ \underline{p_3+1 \ \dots \ p_4} \ \underline{p_2+1 \ \dots \ p_3} \ \underline{p_1+1 \ \dots \ p_2})$ e $\pi'' = \pi' \cdot \bar{\rho_p}(n-\ell_2) = (\underline{-p_3 \ \dots \ -(p_2+1)} \ \underline{-p_4 \ \dots \ -(p_3+1)} \ \dots\dots \ \underline{-n \ \dots \ -(p_b+1)} \ \underline{1 \ \dots \ p_1 \ p_1+1 \ \dots \ p_2})$. É fácil ver que π'' é da forma π^1.

Agora assuma que π^{k-1} é da forma dada acima, isto é

$$\pi^{k-1} = (\underbrace{-p_{2k-1} \ \dots \ -(p_{2k-2}+1)}_{\ell_{2k-1}} \ \underbrace{-p_{2k} \ \dots \ -(p_{2k-1}+1)}_{\ell_{2k}} \ \dots$$
$$\dots \ \underbrace{-n \ \dots \ -(p_b+1)}_{\ell_{b+1}} \ \underbrace{1 \ 2 \ \dots \ p_{2k-2}}_{\ell_1+\ell_2+\dots+\ell_{2k-2}})$$

É fácil ver que $\pi^k = \pi^{k-1} \cdot \bar{\rho_s}(\ell_{2k-1}+1) \cdot \bar{\rho_p}(n-\ell_{2k})$. Portanto, o resultado segue. Ao fim, quando $k = \frac{b-1}{2}$, $\pi^{(b-1)/2} = (\underline{-p_b \ \dots \ -(p_{b-1}+1)} \ \underline{-n \ \dots \ -(p_b+1)} \ \underline{1 \ 2 \ \dots \ p_{b-1}})$ e as duas últimas reversões da sequência, $\bar{\rho_s}(\ell_b+1) \cdot \bar{\rho_p}(n-\ell_{b+1})$, a ordenam.

Se $\pi = \sigma^1$ e b é ímpar, deve-se aplicar a reversão $\bar{\rho_p}(n)$ para transformá-la em σ^2 e então aplicar as reversões dadas em (30). □

O Algoritmo 13 mostra `2-SPRSSR`, que possui complexidade de tempo $O(n^2)$, uma vez que para encontrar os elementos requeridos leva tempo $O(n)$, aplicar as operações necessárias também leva tempo $O(n)$ e a distância é proporcional ao número de *breakpoints*, que é $O(n)$.

Lema 64. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}\bar{\rho_s}}(\pi) \leq 2b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1$.*

Demonstração. No pior caso, duas operações são necessárias para remover um *breakpoint*. Além disso, a permutação identidade ou a permutação reversa podem ser atingidas. No último caso, uma operação extra que não remove *breakpoints* ainda é necessária para terminar a ordenação. □

Teorema 65. SbSigPRSigSR *é 2-aproximável assintoticamente.*

Demonstração. Dos Lemas 61 e 64, temos que o fator de aproximação teórico é $\frac{2b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1}{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}$. Para $b_{\bar{\rho_p}\bar{\rho_s}}(\pi)$ suficientemente grande, temos

$$\lim_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)\to\infty} \frac{2b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+1}{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)} = 2 + \lim_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)\to\infty} \frac{1}{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)} = 2+\epsilon.$$

□

Algoritmo 13 Algoritmo assintótico de 2-aproximação para SbSigPRSigSR

2-SPRSSR(π, $\mathtt{n}$)

```
 1  d ← 0
 2  while  π ≠ ι_n  do
 3      // Tenta remover um breakpoint com uma reversão de prefixo ou de sufixo
 4      if  existe π_j = −π_1 + 1  and 2 ≤ j ≤ n  then
 5          π ← π · ρ̄_p(j − 1)
 6          d ← d + 1
 7      else if existe π_i = −π_n − 1  and 1 ≤ i ≤ n − 1  then
 8          π ← π · ρ̄_s(i + 1)
 9          d ← d + 1
10      // Tenta remover um breakpoint com duas reversões de prefixo
11      else if existe π_j = −π_i − 1 tal que 1 ≤ i < j ≤ n  then
12          π ← π · ρ̄_p(j) · ρ̄_p(j − i)
13          d ← d + 2
14      else if existe π_j = π_i + 1 tal que 0 ≤ i + 1 < j ≤ n  then
15          π ← π · ρ̄_p(i) · ρ̄_s(j − 1)
16          d ← d + 2
17      // Tenta remover um breakpoint com duas reversões de sufixo
18      else if existe π_i = −π_j + 1 tal que 1 ≤ i < j ≤ n  then
19          π ← π · ρ̄_s(i) · ρ̄_s(n + 1 − (j − i))
20          d ← d + 2
21      // Formas especiais
22      else if π = η̄_n  then
23          π ← π · ρ̄_p(n)
24          d ← d + 1
25      else
26          b ← b_{ρ̄_p ρ̄_s}(π)
27          if  b  mod 2 ≡ 1  then
28              if  π = (p_b + 1 ... n p_{b−1} + 1 ... p_b ..... 1 ... p_1)  then
29                  π ← π · ρ̄_p(n)
30                  d ← d + 1
31              Seja ℓ_{b+2−i} o tamanho da i-ésima strip de π
32              π ← π · ρ̄_p(n − ℓ_1) · ρ̄_s(ℓ_2 + 1) · ρ̄_p(n − ℓ_3) · ρ̄_s(ℓ_4 + 1) ...
                ... · ρ̄_p(n − ℓ_{b−1}) · ρ̄_s(ℓ_b + 1) · ρ̄_p(n − ℓ_{b+1})
33          else
34              if  π = (−p_1 ... − 1  − p_2 ... − (p_1 + 1) ..... −n ... − (p_b + 1))  then
35                  π ← π · ρ̄_p(n)
36                  d ← d + 1
37              Seja ℓ_i o tamanho da i-ésima strip de π
38              π ← π · ρ̄_s(ℓ_1 + 1) · ρ̄_p(n − ℓ_2) · ρ̄_s(ℓ_3 + 1) · ρ̄_p(n − ℓ_4) · ...
                ... · ρ̄_s(ℓ_b + 1) · ρ̄_p(n − ℓ_{b+1})
39          d ← d + b + 1
40  return  d
```

Algoritmo 14 Ordenando π_n^{13} com reversões de prefixo com sinal e reversões de sufixo com sinal

SigPRSigSR_Familia_13 ($\pi = \pi_n^{13}$, $\mathtt{n} \geq 5$)

```
1  if  n mod 2 = 0  then
2        π ← π · ρ̄_p(1)
3  for  i ← 3 − (n mod 2) to n by 2  do
4        π ← π · ρ̄_p(i) · ρ̄_p(1)
5  π ← π · ρ̄_p(n)
6  for  i ← n − 1 downto 2 by −2  do
7        π ← π · ρ̄_s(i)
```

Lema 66. *Para $n \geq 5$, $n \leq d_{\bar{\rho}_p\bar{\rho}_s}(\pi_n^{13}) \leq n + \left\lfloor \frac{n-1}{2} \right\rfloor$.*

Demonstração. Primeiro note que $n-1$ é um limitante inferior válido uma vez que $b_{\bar{\rho}_p\bar{\rho}_s}(\pi_n^{13}) = n-1$ e $d_{\bar{\rho}_p\bar{\rho}_s}(\pi) \geq b_{\bar{\rho}_p\bar{\rho}_s}(\pi)$ para qualquer π. No entanto, tal limitante não é justo, porque se fosse, seria necessário remover um *breakpoint* sempre. Se n é par, então $\pi_n^{13} = (+n\ -(n-1)\ +(n-2)\ -(n-3)\ \ldots\ +2\ -1)$ e é fácil perceber que existe apenas uma possibilidade de remover um *breakpoint* de uma vez, o que levará a $\pi' = (-n\ -(n-1)\ +(n-2)\ -(n-3)\ \ldots\ +2\ -1)$. Por outro lado, não é possível remover um *breakpoint* com uma operação de π'. Se n é ímpar, então $\pi_n^{13} = (-n\ +(n-1)\ -(n-2)\ +(n-3)\ \ldots\ +2\ -1)$, da qual não é possível remover um *breakpoint* com uma operação. Portanto, a distância é pelo menos n. O limitante superior é dado pelo Algoritmo 14. □

Teorema 67. *Para $n \geq 5$, $D_{\bar{\rho}_p\bar{\rho}_s}(n) \geq n$ e para $n \geq 16$, $D_{\bar{\rho}_p\bar{\rho}_s}(n) \leq 2n-6$.*

Demonstração. O limitante inferior é verdadeiro devido à família π_n^{13}, conforme o Lema 66 mostra. O limitante superior é verdadeiro porque $D_{\bar{\rho}_p\bar{\rho}_s}(n) \leq D_{\bar{\rho}_p}(n)$, uma vez que $d_{\bar{\rho}_p\bar{\rho}_s}(\pi) \leq d_{\bar{\rho}_p}(\pi)$ para qualquer π. □

A.11 Problema de Ordenação por Transposições de Prefixo e Transposições de Sufixo

Lema 68. *Seja π uma permutação qualquer. Então, $d_{\tau_p\tau_s}(\pi) \geq \frac{b_{\tau_p\tau_s}(\pi)}{2}$.*

Demonstração. Uma transposição de prefixo $\tau_p(i,j)$ separa os pares de elementos $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos pares, uma vez que (π_0, π_1) nunca é considerado *breakpoint* de transposição de prefixo e transposição de sufixo. Assim, $\Delta b_{\tau_p\tau_s}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. De forma equivalente, uma transposição de sufixo $\tau_s(j,k)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$, $[\pi_{k-1}, \pi_k]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas nos dois primeiros pares, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de transposição de prefixo e transposição de sufixo. Assim, $\Delta b_{\tau_p\tau_s}(\pi, \tau_s) \in \{-2, -1, 0, 1, 2\}$. □

Lema 69. *Seja $\pi \neq \iota_n$ uma permutação qualquer. Sempre é possível obter uma transposição de sufixo τ_s tal que $\Delta b_{\tau_s}(\pi, \tau_s) \leq -1$.*

Demonstração. Seja π_j o primeiro elemento da última *strip* de π. Deve existir outra *strip* de π que começa com o elemento π_i, tal que $\pi_i = \pi_j - 1$ e $i < j$. Claramente, existe um *breakpoint* de transposição de sufixo entre π_i e π_{i+1} e a transposição de sufixo $\tau_s(i+1, j)$ o remove. □

Lema 70. *Seja π uma permutação qualquer. Existe no máximo uma transposição de sufixo τ_s tal que $\Delta b_{\tau_s}(\pi, \tau_s) = -2$.*

Demonstração. Suponha que $\tau_s(i, j)$ remove dois *breakpoints* de uma vez de π. Nesse caso, $\pi \cdot \tau_s(i, j) = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_j \ \ldots \ \pi_n \ \pi_i \ \ldots \ \pi_{j-1})$, onde $\pi_i = \pi_n + 1 \neq \pi_{i-1} + 1$ e $\pi_j = \pi_{i-1} + 1 \neq \pi_{j-1} + 1$. É fácil ver que π_n determina unicamente i e i determina unicamente j. □

Lema 71. *Seja π uma permutação qualquer. Então, $d_{\tau_s}(\pi) \geq \frac{b_{\tau_s}(\pi)}{2}$.*

Demonstração. Uma transposição de sufixo $\tau_s(j, k)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$, $[\pi_{k-1}, \pi_k]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas nos dois primeiros pares, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de transposição de sufixo. Assim, $\Delta b_{\tau_s}(\pi, \tau_s) \in \{-2, -1, 0, 1, 2\}$. □

Lema 72. *Seja $\pi \neq \iota_n$ uma permutação qualquer e $\tau_s(j, k)$ uma transposição de sufixo tal que $\pi \cdot \tau_s = \iota_n$. Então $\pi_j = n$ e $\Delta b_{\tau_s}(\pi, \tau_s) = -2$.*

Demonstração. Não é difícil perceber que uma permutação que está à uma transposição de sufixo de distância da permutação identidade só pode estar no formato

$$(\underline{1\ 2\ \ldots\ k}\ \underline{k+\ell+1\ k+\ell+2\ \ldots\ n}\ \underline{k+1\ k+2\ \ldots\ k+\ell})$$

com $k \geq 1$ e $\ell \geq 1$. □

Algoritmo 15 Algoritmo de 2-aproximação para SbST

```
2-ST(π, n)
1  d ← 0
2  while  π ≠ ι_n  do
3        Seja j a posição do primeiro elemento da última strip de π
4        i ← π^{-1}(π_j − 1)
5        π ← π · τ_s(i + 1, j)
6        d ← d + 1
7  return  d
```

Lema 73. *Seja π uma permutação qualquer. Então, $d_{\tau_s}(\pi) \leq b_{\tau_s}(\pi) - 1$.*

Demonstração. No pior caso, sempre é possível remover um *breakpoint* de transposição de sufixo com uma transposição de sufixo. Além disso, a última operação da ordenação sempre remove dois *breakpoints* de uma vez, como mostra o Lema 72. □

Algoritmo 16 Algoritmo de 2-aproximação para SbST, versão gulosa

2-STg(π, n)

```
1  d ← 0
2  while  π ≠ ι_n  do
3      // Tenta remover dois breakpoints
4      i ← π^{-1}(π_n + 1)
5      j ← π^{-1}(π_{i-1} + 1)
6      if  i < j  and j < n  then
7          π ← π · τ_s(i, j)
8      // Remove apenas um breakpoint
9      else
10         Seja j a posição do primeiro elemento da última strip de π
11         i ← π^{-1}(π_j − 1)
12         π ← π · τ_s(i + 1, j)
13     d ← d + 1
14 return  d
```

Teorema 74. SbST *é 2-aproximável.*

Demonstração. Diretamente dos Lemas 71 e 73. □

Lema 75. *Seja $\pi \neq \iota_n$ uma permutação qualquer. Sempre é possível obter uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_p) \leq -1$ e uma transposição de sufixo τ_s tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_s) \leq -1$.*

Demonstração. Seja π_i o último elemento da primeira *strip* de π. Deve existir outra *strip* em π que começa com algum π_j tal que $\pi_j = \pi_i + 1$ e $i < j$, isto é, o próximo elemento da primeira *strip*. Se $\pi_i \neq n$, então $\tau_p(i+1, j)$ remove um *breakpoint* de transposição de prefixo e transposição de sufixo. No entanto, se $\pi_i = n$ então $j = n+1$ e $\tau_p(i+1, j)$ não removeria um *breakpoint* desse tipo. Nesse caso, também deve existir uma *strip* que termina com um π_j tal que $\pi_j = \pi_1 - 1$, o elemento anterior da primeira *strip*, e a transposição $\tau_p(i+1, j+1)$ remove um *breakpoint*.

Seja π_j o primeiro elemento da última *strip* de π. Deve existir outra *strip* de π que começa com o elemento π_i tal que $\pi_i = \pi_j - 1$ e $i < j$, isto é, o elemento anterior da primeira *strip*. Se $\pi_j \neq 1$, então $\tau_s(i+1, j)$ remove um *breakpoint*. No entanto, se $\pi_j = 1$ então $i = 0$ e $\tau_s(i+1, j)$ não removeria um *breakpoint* como desejado. Nesse caso, também deve existir uma *strip* que termina com um π_i tal que $\pi_i = \pi_n + 1$, o próximo elemento da última *strip*, e a transposição $\tau_s(i, j)$ remove um *breakpoint*. □

Lema 76. *Seja π uma permutação qualquer. Existe no máximo uma transposição de prefixo τ_p tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_p) = -2$ e existe no máximo uma transposição de sufixo τ_s tal que $\Delta b_{\tau_p\tau_s}(\pi, \tau_s) = -2$.*

Demonstração. Suponha que $\tau_p(i, j)$ remove dois *breakpoints* de uma vez de π. Nesse caso, $\pi \cdot \tau_p(i, j) = (\pi_i \ \ldots \ \pi_{j-1} \ \pi_1 \ \ldots \ \pi_{i-1} \ \pi_j \ \ldots \ \pi_n)$, onde $\pi_{i-1} = \pi_j - 1 \neq \pi_i - 1$ e $\pi_{j-1} = \pi_1 - 1 \neq \pi_j - 1$. É fácil ver que π_1 determina unicamente j e j determina unicamente i. Para

que *breakpoints* de transposição de prefixo e transposição de sufixo sejam respeitados, deve-se garantir que $2 \leq i < j \leq n$.

Suponha que $\tau_s(i,j)$ remove dois *breakpoints* de uma vez de π. Nesse caso, $\pi \cdot \tau_s(i,j) = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_j \ \ldots \ \pi_n \ \pi_i \ \ldots \ \pi_{j-1})$, onde $\pi_i = \pi_n + 1 \neq \pi_{i-1} + 1$ e $\pi_j = \pi_{i-1} + 1 \neq \pi_{j-1} + 1$. É fácil ver que π_n determina unicamente i e i determina unicamente j. Da mesma forma, deve-se garantir que $2 \leq i < j \leq n$. □

Ambos `2-PTST` e `2-PTSTg` são apresentados nos Algoritmos 18 e 19, respectivamente.

Algoritmo 17 Removendo um *breakpoint* de transposição de prefixo e transposição de sufixo com uma operação de transposição de prefixo ou de sufixo

```
REMOVE_1_BREAKPOINT(π, n)
 1  Seja r um escolhido aleatoriamente entre 0 ou 1
 2  if r = 0 then
 3       Seja i a posição do último elemento da primeira strip de π
 4       if π_i = n then
 5            j ← π^{-1}(π_1 − 1) + 1
 6       else
 7            j ← π^{-1}(π_i + 1)
 8       π ← π · τ_p(i + 1, j)
 9  else
10       Seja j a posição do primeiro elemento da última strip de π
11       if π_j = 1 then
12            i ← π^{-1}(π_n + 1) − 1
13       else
14            i ← π^{-1}(π_j − 1)
15       π ← π · τ_s(i + 1, j)
16  return π
```

Algoritmo 18 Algoritmo de 2-aproximação para SbPTST

```
2-PTST(π, n)
1  d ← 0
2  while π ≠ ι_n do
3       π ← REMOVE_1_BREAKPOINT(π, n)
4       d ← d + 1
5  return d
```

Lema 77. *Seja π uma permutação qualquer. Então, $d_{\tau_p\tau_s}(\pi) \leq b_{\tau_p\tau_s}(\pi)$.*

Demonstração. No pior caso, sempre é possível remover um *breakpoint* com uma operação de transposição de prefixo ou de sufixo. Além disso, note que a última transposição remove apenas um *breakpoint* de transposição de prefixo e transposição de sufixo. □

Teorema 78. SbPTST *é 2-aproximável.*

Algoritmo 19 Algoritmo de 2-aproximação para SbPTST, versão gulosa

2-PTSTg(π, n)

```
1  d ← 0
2  while  π ≠ ι_n  do
3        // Tenta remover dois breakpoints
4        jp ← π^{-1}(π_1 − 1) + 1
5        ip ← π^{-1}(π_jp − 1) + 1
6        is ← π^{-1}(π_n + 1)
7        js ← π^{-1}(π_{is−1} + 1)
8        if  2 ≤ ip and ip < jp and jp ≤ n  then
9              if  2 ≤ is and is < js and js ≤ n  then
10                   Escolha aleatoriamente fazer τ_p(ip, jp) ou τ_s(is, js)
11             else
12                   π ← π · τ_p(ip, jp)
13       else if 2 ≤ is and is < js and js ≤ n  then
14             π ← π · τ_s(is, js)
15       // Remove apenas um breakpoint
16       else
17             π ← Remove_1_breakpoint(π, n)
18       d ← d + 1
19 return  d
```

Algoritmo 20 Ordenando π_n^{14} com transposições de prefixo e transposições de sufixo

PTST_Familia_14 ($\pi = \pi_n^{14}$, n ≥ 4)

```
1  m ← 4⌊n/4⌋
2  for  i ← 1 to m/4 − 1  do
3        π ← π · τ_p(5, m − 2(i − 1))
4  π ← π · τ_p(3, m/2 + 2)
5  x ← π_n^{-1} + 1
6  y ← m + 1
7  for  i ← 1 to m/2  do
8        z ← π_x
9        π ← π · τ_p(x, y)
10       y ← π^{-1}_{z−1} + 1
11       w ← π_y − 1
12       x ← π_w^{-1} + 1
13 for  i ← m + 1 to n  do
14       π ← π · τ_p(i, i + 1)
```

Demonstração. Diretamente dos Lemas 68 e 77. □

Lema 79. *Para* $n \geq 3$, $\left\lceil \frac{n-1}{2} \right\rceil + 1 \leq d_{\tau_p\tau_s}(\pi_n^{14}) \leq n - \left\lfloor \frac{n}{4} \right\rfloor$.

Demonstração. Primeiro note que $\left\lceil \frac{n-1}{2} \right\rceil$ é um limitante inferior válido porque $b_{\tau_p\tau_s}(\pi_n^{14}) = n-1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para todo π. No entanto, tal limitante não é justo e a distância é pelo menos $\left\lceil \frac{n-1}{2} \right\rceil + 1$. Se o limitante fosse justo, quando n fosse par uma ordenação precisaria ter $\left\lceil \frac{n-1}{2} \right\rceil - 1$ operações que removem dois *breakpoints* de uma vez e uma operação que remove um *breakpoint*. Podemos ver que remover dois *breakpoints* na primeira operação já não é possível. Além disso, as únicas formas de remover um *breakpoint* na primeira operação são colocando n depois de $n-1$ ou colocando 1 antes do 2. De qualquer forma, a permutação gerada ainda não permitiria a remoção de dois *breakpoints*. Quando n fosse ímpar, uma ordenação poderia apenas ter operações que removem dois *breakpoints*. Contudo, isso já não é possível na primeira operação. O limitante superior é verdadeiro devido ao algoritmo apresentado por Dias e Meidanis [19] que ordena η_n usando no máximo $n - \left\lfloor \frac{n}{4} \right\rfloor$ transposições de prefixo e é apresentado no Algoritmo 20. □

Teorema 80. *Para* $n \geq 3$, $\left\lceil \frac{n-1}{2} \right\rceil + 1 \leq D_{\tau_p\tau_s}(n) \leq n - \log_{\frac{9}{2}} n$.

Demonstração. O limitante inferior é verdadeiro devido à família π_n^{14}, como o Lema 79 mostra. O limitante superior é verdadeiro porque $D_{\tau_p\tau_s}(n) \leq D_{\tau_p}(n)$, uma vez que $d_{\tau_p\tau_s}(\pi) \leq d_{\tau_p}(\pi)$ para qualquer π. □

A.12 Problema de Ordenação por Reversões de Prefixo, Transposições de Prefixo, Reversões de Sufixo e Transposições de Sufixo

Lema 81. *Seja* π *uma permutação qualquer. Então,* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \frac{b_{\rho_p\rho_s}(\pi)}{2}$.

Demonstração. Uma reversão de prefixo $\rho_p(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* de reversão de prefixo e reversão de sufixo apenas no último par, uma vez que (π_0, π_1) nunca é considerado *breakpoint* desse tipo. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \rho_p) \in \{-1, 0, 1\}$. Equivalentemente, uma reversão de sufixo $\rho_s(j)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas no primeiro par, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de reversão de prefixo e reversão de sufixo. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \rho_s) \in \{-1, 0, 1\}$. Uma transposição de prefixo $\tau_p(i, j)$ separa os pares de elementos $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos pares. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. Por fim, uma transposição de sufixo $\tau_s(j, k)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$, $[\pi_{k-1}, \pi_k]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas nos dois primeiros pares. Assim, $\Delta b_{\rho_p\rho_s}(\pi, \tau_s) \in \{-2, -1, 0, 1, 2\}$. □

Lema 82. *Seja* π *uma permutação qualquer. Então,* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \frac{b_{\tau_p\tau_s}(\pi)}{n}$.

Demonstração. É fácil ver que $\Delta b_{\tau_p\tau_s}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$ e $\Delta b_{\tau_p\tau_s}(\pi, \tau_s) \in \{-2, -1, 0, 1, 2\}$. No entanto, não é possível fazer uma análise similar para $\Delta b_{\tau_p\tau_s}(\pi, \rho_p)$ e $\Delta b_{\tau_p\tau_s}(\pi, \rho_s)$, isto é, para a variação no número de *breakpoints* de transposição de prefixo e transposição de sufixo após uma reversão de prefixo ou uma reversão de sufixo. Isso ocorre porque uma reversão $\rho_p(i)$ (resp. $\rho_s(j)$) pode alterar a configuração de *breakpoints* de transposição de prefixo e transposição de sufixo em todo o intervalo $\pi_1, \ldots, \pi_i$ (resp. $\pi_j, \ldots, \pi_n$), o que, no pior caso, faz com que $\Delta b_{\tau_p\tau_s}(\pi, \rho_p) \in \{-n, \ldots, n\}$ (resp. $\Delta b_{\tau_p\tau_s}(\pi, \rho_s) \in \{-n, \ldots, n\}$). □

Lema 83. *Seja π uma permutação qualquer e $G_b(\pi)$ seu grafo de breakpoints. Uma sequência de no máximo três reversões de prefixo, transposições de prefixo, reversões de sufixo ou transposições de sufixo que remove pelo menos um breakpoint de reversão de prefixo e reversão de sufixo existe se $G_b(\pi)$ contém pelo menos uma das oito arestas azuis a seguir:*

(1) (π_1, π_j) do tipo 4 com $\pi_1 \neq 1$ e $j \leq n$;

(2) (π_i, π_n) do tipo 4 com $\pi_n \neq n$ e $i \geq 1$;

(3) (π_1, π_j) do tipo 1 com $\pi_1 \neq 1$ e $j \leq n$;

(4) (π_i, π_n) do tipo 2 com $\pi_n \neq n$ e $i \geq 1$;

(5) (π_i, π_j) do tipo 3 com $\pi_1 = 1$, $i \geq 1$ e $j \leq n$;

(6) (π_i, π_j) do tipo 3 com $\pi_n = n$, $i \geq 1$ e $j \leq n$;

(7) (π_i, π_j) do tipo 2 com $\pi_1 = 1$, $i \geq 1$ e $j \leq n$, onde π_i é o último elemento da primeira strip de π;

(8) (π_i, π_j) do tipo 1 com $\pi_n = n$, $i \geq 1$ e $j \leq n$, onde π_j é o primeiro elemento da última strip de π.

Demonstração. Se existe uma aresta azul (π_i, π_j), então $\pi_j = \pi_i \pm 1$. Para criar uma adjacência entre π_i e π_j sem criar novos *breakpoints* deve-se fazer, respectivamente, para cada tipo de aresta:

(1) Uma transposição de prefixo $\tau_p(k, j+1)$ onde (π_{k-1}, π_k) é uma aresta vermelha presa, $i < k < j$, que une π_1 e π_j;

(2) Uma transposição de sufixo $\tau_s(i, k)$ onde (π_{k-1}, π_k) é uma aresta vermelha presa, $i < k < j$, que une π_i e π_n;

(3) Uma reversão de prefixo $\rho_p(j-1)$, que une π_1 e π_j;

(4) Uma reversão de sufixo $\rho_s(i+1)$, que une π_i e π_n;

(5) Uma transposição de prefixo $\tau_p(i+1, j)$, que une π_i e π_j;

(6) Uma transposição de sufixo $\tau_s(i+1, j)$, que une π_i e π_j;

(7) Uma reversão de prefixo $\rho_p(j)$, que gera $\pi' = (\pi_j \ \ldots \ \pi_i \ \ldots \ \pi_1 \ \pi_{j+1} \ \ldots \ \pi_n)$ sem remover nenhum *breakpoint*, seguida por uma reversão de prefixo $\tau_p(j-i)$, que gera $\pi'' = (\pi_{i+1} \ \ldots \ \pi_j \ \pi_i \ \ldots \ \pi_1 \ \pi_{j+1} \ \ldots)$ ao agir sobre a aresta azul $(\pi'_1 = \pi_j, \pi'_{j-i+1} = \pi_i)$, seguida por uma operação para lidar com uma aresta do tipo 4 ou do tipo 1, uma vez que a

aresta vermelha ($\pi''_0 = \pi_0, \pi''_1 = \pi_{i+1}$) existe. Em resumo, usa-se 3 operações para remover 2 *breakpoints*;

(8) Uma reversão de sufixo $\rho_s(i)$, que gera $\pi' = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_n \ \ldots \ \pi_j \ \ldots \ \pi_i)$ sem remover nenhum *breakpoint*, seguida por uma reversão de sufixo $\rho_s(n+1-(j-i))$, que gera $\pi'' = (\ldots \ \pi_{i-1} \ \pi_n \ \ldots \ \pi_j \ \pi_i \ \ldots \ \pi_{j-1})$ ao agir sobre a aresta azul ($\pi'_{n-(j-i)} = \pi_j, \pi'_n = \pi_i$), seguida por uma operação para lidar com uma aresta do tipo 4 ou 2, uma vez que a aresta vermelha ($\pi''_n = \pi_{j-1}, \pi''_{n+1} = \pi_{n+1}$) existe. Em resumo, usamos 3 operações para remover 2 *breakpoints*.

□

Lema 84. *Seja π uma permutação qualquer. Então, $d_{\rho_s\tau_s}(\pi) \geq \frac{b_{\rho_s}(\pi)}{2}$.*

Demonstração. Uma reversão de sufixo $\rho_s(j)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* de reversão de sufixo apenas no primeiro par, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de reversão de sufixo. Assim, $\Delta b_{\rho_s}(\pi, \rho_s) \in \{-1, 0, 1\}$. Uma transposição de sufixo $\tau_s(j,k)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$, $[\pi_{k-1}, \pi_k]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas nos dois primeiros pares. Assim, $\Delta b_{\rho_s}(\pi, \tau_s) \in \{-2, -1, 0, 1, 2\}$. □

Lema 85. *Seja π uma permutação qualquer. Então, $d_{\rho_s\tau_s}(\pi) \leq \frac{3b_{\rho_s}(\pi)}{2}$.*

Demonstração. No pior caso, serão aplicadas três operações para remover dois *breakpoints* de reversão de sufixo. □

Teorema 86. SbSRST *é 3-aproximável.*

Demonstração. Diretamente dos Lemas 84 e 85. □

Lema 87. *Seja $\pi \neq \iota_n$ uma permutação na qual não existem as seis primeiras arestas boas do Lema 83, que garantem remover um breakpoint com um único movimento. Então, π é de uma das três formas:*

$$\eta_n \tag{15}$$

ou

$$\sigma^1 = (\underline{1\ 2\ \ldots\ k} \ldots\ldots \underline{k{+}i\ \ldots\ k{+}1} \ldots\ldots \underline{j{-}1\ \ldots\ j{-}\ell} \ldots\ldots \underline{j\ j{+}1\ \ldots\ n}) \tag{16}$$

sendo que (i) $i \geq 2$ se $k \geq 1$ ou $i \geq 1$ se $k \geq 2$, (ii) $\ell \geq 2$ se $j \geq 1$ ou $\ell \geq 1$ se $j \geq 2$, e (iii) a ordem relativa entre as strips que contém $k+i$ e $j-\ell$ é irrelevante. Ou então

$$\sigma^2 = (\underline{n\ n-1\ \ldots\ j}\ \pi_{n-j+2}\ \ldots\ldots\ \pi_{n-k}\ \underline{k\ k-1\ \ldots\ 1}) \tag{17}$$

com $\pi_{n-j+2} \neq j-1$, $\pi_{n-k} \neq k+1$, $j < n$ e $k > 1$.

Demonstração. Note que η_n tem apenas duas arestas vermelhas $(0, n)$ e $(1, n+1)$ e duas arestas azuis $(0, 1)$ e $(n, n+1)$, que claramente não formam arestas boas devido aos limitantes sobre os índices declarados no Lema 83.

É fácil perceber que quando $\pi_1 = 1$ e $\pi_n = n$, nenhuma das quatro primeiras arestas boas do Lema 83 serão encontradas. Assim, os algoritmos apenas procurarão por arestas do tipo 3 com $i \geq 1$ e $j \leq n$.

Não é difícil perceber que para que exista uma aresta do tipo 3 em π, devem existir duas *strips* crescentes $s_1 = \langle k, \ldots, k+\ell \rangle$ e $s_2 = \langle k+\ell+1, \ldots, k+\ell+w \rangle$ tais que s_1 apareça antes de s_2 ou então devem existir duas *strips* decrescentes $s'_1 = \langle k+\ell, \ldots, k \rangle$ e $s'_2 = \langle k+\ell+w, \ldots, k+\ell+1 \rangle$ tais que s'_2 apareça antes de s'_1. Além disso, em ambos os casos, não se pode ter ambas *strips* como *singletons* simultaneamente. Portanto, $\ell \geq 1$ se $w \geq 2$ ou $\ell \geq 2$ se $w \geq 1$.

Em especial, como a primeira *strip* $p = \langle 1, \ldots, k \rangle$ é crescente ou *singleton*, pois estamos assumindo que $\pi_1 = 1$, sua próxima *strip* em ι_n deve ser decrescente em π (ela não pode aparecer antes de p e se ela fosse crescente haveria uma aresta boa do tipo 3). Equivalentemente, como a última *strip* $u = \langle j, \ldots, n \rangle$ também é crescente ou *singleton*, porque $\pi_n = n$, sua *strip* anterior em ι_n deve ser decrescente em π (ela não pode aparecer depois de u e se ela fosse crescente haveria uma aresta boa do tipo 3).

Note que, a cada duas *strips* em π que são consecutivas em ι_n ou em η_n, as ordens relativas citadas acima devem ser respeitadas para que não existam arestas boas do tipo 3. Por isso, vários são os formatos de permutações que σ^1 representa. No entanto, escolhemos por enfatizar apenas o formato das *strips* mais próximas à primeira e à última, uma vez que temos $\pi_1 = 1$ e $\pi_n = n$.

Quando $\pi_1 \neq 1$, os algoritmos procuram por arestas boas de prefixo (π_1, π_j) do tipo 1 ou 4 com $j \leq n$. Assim, se $\pi_1 = n$, π_j deve ser $n \pm 1$. Se $\pi_j = n + 1$, então a aresta não é boa. Por outro lado, se $\pi_j = n - 1$, então a aresta é boa apenas se $j > 2$. Portanto, se a primeira *strip* de π começa com n e não é um *singleton*, então arestas boas de prefixo do tipo 1 ou 4 não existem. Para qualquer outro valor, vai haver algum $\pi_j = \pi_1 + 1$ ou $\pi_j = \pi_1 - 1$ com $j \leq n$ e, portanto, tais tipos de aresta existirão.

Quando $\pi_n \neq n$, os algoritmos procuram por arestas boas de sufixo (π_i, π_n) do tipo 2 ou 4 com $i \geq 1$. Assim, se $\pi_n = 1$, π_i deve ser 0 ou 2. Se $\pi_i = 0$, então a aresta não é boa. Por outro lado, se $\pi_i = 2$, então a aresta é boa apenas se $i < n - 1$. Portanto, se a última *strip* de π termina em 1 e não é um *singleton*, arestas boas de sufixo do tipo 2 ou 4 não existem. Para qualquer outro valor, vai haver algum $\pi_i = \pi_n + 1$ ou $\pi_i = \pi_n - 1$ com $i \geq 1$ e, portanto, tais tipos de aresta existirão.

Com isso, se $\pi_1 \neq 1$ e $\pi_n \neq n$, arestas boas não existem apenas quando π está no formato de σ^2. □

Lema 88. *Seja $\pi \neq \iota_n$ uma permutação de uma das formas dadas no Lema 87. Se $\pi = \eta_n$, então uma reversão $\rho_p(n)$ a ordena. Caso contrário, uma transposição $\tau_p(i+1, n+1)$, onde π_i é o*

último elemento da primeira strip, concatena a primeira strip com a última sem alterar o número de breakpoints. Após ela, sempre é possível continuar removendo pelo menos um breakpoint com uma única operação.

Demonstração. Quando $\pi = \sigma^1$, concatenar as *strips* leva a $\pi' = (\ldots\ldots \underline{j\ j+1\ \ldots\ n}\ \underline{1\ 2\ \ldots\ k})$ e quando $\pi = \sigma^2$, concatenar as *strips* leva a $\pi' = (\ldots\ldots \underline{k\ k-1\ \ldots 1}\ \underline{n\ n-1\ \ldots\ j})$. Em ambos os casos, π' não é de nenhum formato dado no Lema 87 e, portanto, tem arestas boas.

Além disso, a permutação voltaria a ser de um daqueles formatos apenas quando $\pi_1 = 1$ e $\pi_n = n$ ou quando $\pi_1 = n$ e $\pi_n = 1$, o que só iria acontecer se o algoritmo separasse 1 e n. □

Lema 89. *Seja (π_i, π_j) uma aresta azul do tipo 4 com $i = 1$ ou com $j = n$ existente em uma permutação $\pi \neq \iota_n$ qualquer. Se a aresta vermelha entre os elementos 1 e n for a única aresta presa entre π_i e π_j, então na verdade $i = 1$, $j = n$ e a permutação é de uma das duas formas:*

$$\pi' = (\underline{k\ k-1\ k-2\ \ldots\ 2\ 1}\ \underline{n\ n-1\ \ldots\ k+2\ k+1}) \tag{31}$$

ou

$$\pi'' = (\underline{k+1\ k+2\ \ldots\ n-1\ n}\ \underline{1\ 2\ \ldots\ k-2\ k-1\ k}) \tag{32}$$

Além disso, ao agir em tal aresta os algoritmos estarão realizando sua última ou penúltima operação.

Demonstração. Note que, se entre π_i e π_j existe uma única aresta vermelha, então o segmento $\pi_i, \ldots, \pi_j$ só pode ser formado por duas *strips*. Além disso, como essa única aresta está entre os elementos 1 e n e como, por definição, uma aresta azul é tal que $\pi_i = \pi_j \pm 1$, π só pode estar no formato π' ou π''.

Os algoritmos então realizarão uma transposição para agir sobre a aresta do tipo 4. Se $\pi = \pi'$, a transposição $\tau_p(k+1, n+1) = \tau_s(1, k+1)$ é feita, gera a permutação identidade e os algoritmos realizaram sua última operação. Se $\pi = \pi''$, a transposição $\tau_p(n-k+1, n+1) = \tau_s(1, n-k+1)$ é feita, gera a permutação reversa e os algoritmos realizaram sua penúltima operação. Quando $\pi = \eta_n$, o algoritmo explicitamente realiza uma operação $\rho_p(n)$ para ordená-la. □

Lema 90. *A operação explicada no Lema 88 é realizada no máximo uma vez pelos algoritmos* `2-PRPTSRST` *e* `2-PRPTSRSTg`.

Demonstração. Se sempre for possível remover pelo menos um *breakpoint* com uma única operação, nenhuma permutação em alguma das formas dadas no Lema 87 é alcançada e assim a operação nunca é realizada.

Caso contrário, após concatenar as *strips* uma vez, os elementos 1 e n são unidos e os algoritmos não podem mais separá-los, a não ser que a aresta vermelha entre eles seja única. No entanto, nesse caso, os algoritmos estarão realizando sua última ou penúltima operação, conforme o demonstrado no Lema 89. □

Lema 91. *Seja π uma permutação qualquer. Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq b_{\rho_p\rho_s}(\pi) + 2$.*

Demonstração. Enquanto os algoritmos conseguem encontrar arestas boas, eles sempre removem pelo menos um *breakpoint* por meio de uma operação. Quando não for possível eles realizam uma transposição que não remove nem adiciona *breakpoints* e une a primeira e a última *strips*, segundo o Lema 88. Isso acontece no máximo uma única vez, segundo o Lema 90. A partir disso, é possível continuar removendo um *breakpoint* com um movimento até que a permutação identidade ou a reversa sejam atingidas, de acordo com os Lemas 88 e 89. No último caso, uma reversão extra precisa ser feita para ordenar a permutação. Com isso, temos que os algoritmos realizam no máximo $b_{\rho_p\rho_s}(\pi) + 2$ operações para ordenar uma permutação qualquer. □

Teorema 92. SbPRPTSRST *é 2-aproximável assintoticamente.*

Demonstração. Dos Lemas 81 e 91, temos que o fator de aproximação teórico é $\frac{b_{\rho_p\rho_s}(\pi)+2}{\frac{b_{\rho_p\rho_s}(\pi)}{2}}$. Para $b_{\rho_p\rho_s}(\pi)$ suficientemente grande, temos

$$\lim_{b_{\rho_p\rho_s}(\pi)\to\infty} \frac{b_{\rho_p\rho_s}(\pi) + 2}{\frac{b_{\rho_p\rho_s}(\pi)}{2}} = \lim_{b_{\rho_p\rho_s}(\pi)\to\infty} \frac{2b_{\rho_p\rho_s}(\pi) + 4}{b_{\rho_p\rho_s}(\pi)} = 2 + \lim_{b_{\rho_p\rho_s}(\pi)\to\infty} \frac{4}{b_{\rho_p\rho_s}(\pi)} = 2 + \epsilon.$$

□

`2-PRPTSRST` e `2-PRPTSRSTg` são apresentados nos Algoritmos 24 e 25, respectivamente. Apesar dos Algoritmos 21, 22 e 23 serem utilizados em ambos, deve-se sempre ter em mente que eles percorrem as permutações de forma diferente, conforme já explicado. Note também que, como os Algoritmos 21 e 23 mostram, podemos escolher entre uma transposição de prefixo ou uma transposição de sufixo dando preferência para aquela que remove dois *breakpoints* (o que é possível de ser feito mesmo no caso do algoritmo `2-PRPTSRST`).

Lema 93. *Para $n \geq 6$, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) \leq \lceil \frac{n}{2} \rceil + 1$ se n é par e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) = \lceil \frac{n}{2} \rceil + 1$ se n é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) \leq \lceil \frac{n}{2} \rceil + 1$, para qualquer n, por causa do Algoritmo 26. Além disso, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n^{19}) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n^{19}) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Note que $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$ quando n é par e $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$ quando n é ímpar.

Quando n é ímpar, para o limitante inferior ser justo, a ordenação deveria ter apenas operações que removem dois *breakpoints*. Existe apenas uma operação que remove dois *breakpoints* no primeiro passo, e ela transforma a permutação em $\pi' = (6\ 8\ 10\ \ldots\ n-1\ n\ n-2\ n-4\ \ldots\ 5\ 3\ 2\ 4\ 1)$. A partir daqui, para remover dois *breakpoints* também existe uma única forma: a cada passo, colocar o número par i que está na posição π_1 entre os números ímpares $i+1$ e $i-1$. Isso utiliza $\lceil \frac{n}{2} \rceil - 3 + 1$ movimentos e deixa a permutação no formato $(n\ n-1\ n-2\ \ldots\ 7\ 6\ 5\ 3\ 2\ 4\ 1)$, faltando ainda três movimentos para ordená-la. Note que se menos do que dois *breakpoints* fossem removidos por operação, então mais passos seriam necessários para ordenar. Portanto, a ordenação apresentada é ótima. □

Algoritmo 21 Lidando com arestas boas do tipo 4

PRPTSRST_ARESTA_TIPO_4(π, $\mathtt{n}$)

```
 1  if   π_1 ≠ 1  and G_b(π) tem uma ABP (π_1, π_jp) do tipo 4  and jp ≤ n   then
 2       Seja (kp − 1, kp) uma aresta vermelha presa entre π_1 e π_jp
 3       if   π_n ≠ n  and G_b(π) tem uma ABS (π_is, π_n) do tipo 4  and is ≥ 1   then
 4            Seja (ks − 1, ks) uma aresta vermelha presa entre π_is e π_n
              // Tenta escolher a transposição que remove 2 breakpoints de uma vez:
 5            if   π_kp − 1 = π_jp+1 ± 1  and π_jp ≠ π_jp+1 ± 1   then
 6                 π ← π · τ_p(kp, jp + 1)
 7            else if π_ks − 1 = π_js+1 ± 1  and π_is−1 ≠ π_is ± 1   then
 8                 π ← π · τ_s(ks, js + 1)
              // Se não for possível, escolha a de menor tamanho
 9            else if jp < n − is   then
10                 π ← π · τ_p(kp, jp + 1)
11            else
12                 π ← π · τ_s(ks, js + 1)
13       else
14            π ← π · τ_p(kp, jp + 1)
15  else if π_n ≠ n  and G_b(π) tem uma ABS (π_is, π_n) do tipo 4  and is ≥ 1   then
16       Seja (ks-1, ks) uma aresta vermelha presa entre π_is e π_n
17       π ← π · τ_s(ks, js + 1)
18  return   π
```

Algoritmo 22 Lidando com arestas boas de prefixo do tipo 1 e arestas boas de sufixo do tipo 2

PRPTSRST_ARESTA_TIPO_1_2(π, $\mathtt{n}$)

```
 1  if   π_1 ≠ 1  and G_b(π) tem uma ABP (π_1, π_jp) do tipo 1  and jp ≤ n   then
 2       if   π_n ≠ n  and G_b(π) tem uma ABS (π_is, π_n) do tipo 2  and is ≥ 1   then
              // Escolha a de menor tamanho, por convenção
 3            if   jp < n − is   then
 4                 π ← π · ρ_p(jp − 1)
 5            else
 6                 π ← π · ρ_s(is + 1)
 7       else
 8            π ← π · ρ_p(jp − 1)
 9  else if π_n ≠ n  and G_b(π) tem uma ABS (π_is, π_n) do tipo 2  and is ≥ 1   then
10       π ← π · ρ_s(is + 1)
11  return   π
```

Algoritmo 23 Lidando com arestas boas do tipo 3

PRPTSRST_ARESTA_TIPO_3(π, $\mathtt{n}$)

```
1  if  π_1 = 1  and G_b(π) tem ABP (π_ip, π_jp) do tipo 3  and 1 ≤ ip < jp ≤ n  then
2      if  π_n = n  and G_b(π) tem ABS (π_is, π_js) do tipo 3  and 1 ≤ is < js ≤ n  then
           // Primeiro tenta escolher a transposição que elimina 2 breakpoints de uma vez:
3          if  π_1 = π_{jp−1} ± 1  then
4              π ← π · τ_p(ip + 1, jp)
5          else if π_n = π_{is+1} ± 1  then
6              π ← π · τ_s(is + 1, js)
           // Se não for possível, escolha a menor, por convenção
7          else if jp < n − is  then
8              π ← π · τ_p(ip + 1, jp)
9          else
10             π ← π · τ_s(is + 1, js)
11     else
12         π ← π · τ_p(ip + 1, jp)
13 else if π_n = n  and G_b(π) tem ABS (π_is, π_js) do tipo 3  and 1 ≤ is < js ≤ n  then
14     π ← π · τ_s(is + 1, js)
15 return  π
```

Algoritmo 24 Algoritmo assintótico de 2-aproximação para SbPRPTSRST

2-PRPTSRST(π, $\mathtt{n}$)

```
1  d ← 0
2  while  π ≠ ι_n  do
3      if  π_1 ≠ 1  and G_b(π) tem uma ABP do tipo 4  or
       π_n ≠ n  and G_b(π) tem uma ABS do tipo 4  then
4          π ← PRPTSRST_ARESTA_TIPO_4(π, n)
5      else if π_1 ≠ 1  and G_b(π) tem uma ABP do tipo 1  or
       π_n ≠ n  and G_b(π) tem uma ABS do tipo 2  then
6          π ← PRPTSRST_ARESTA_TIPO_1_2(π, n)
7      else if π_1 ≠ 1  and G_b(π) tem uma ABP do tipo 3  or
       π_n ≠ n  and G_b(π) tem uma ABS do tipo 3  then
8          π ← PRPTSRST_ARESTA_TIPO_3(π, n)
9      else if π = η_n  then
10         π ← π · ρ_p(n)
11     else
12         Seja k a posição do último elemento da primeira strip de π
13         π ← π · τ_p(k + 1, n + 1)
14     d ← d + 1
15 return  d
```

Algoritmo 25 Algoritmo assintótico de 2-aproximação para SbPRPTSRST, versão gulosa

2-PRPTSRSTg(π, $\mathtt{n}$)

```
 1  d ← 0
 2  while  π ≠ ι_n  do
 3       if  π_1 ≠ 1  and G_b(π) tem uma ABP do tipo 4  or
          π_n ≠ n  and G_b(π) tem uma ABS do tipo 4  then
 4            π ← PRPTSRST_ARESTA_TIPO_4(π, n)
 5       else if π_1 ≠ 1  and G_b(π) tem uma ABP do tipo 3  or
          π_n ≠ n  and G_b(π) tem uma ABS do tipo 3  then
 6            π ← PRPTSRST_ARESTA_TIPO_3(π, n)
 7       else if π_1 ≠ 1  and G_b(π) tem uma ABP do tipo 1  or
          π_n ≠ n  and G_b(π) tem uma ABS do tipo 2  then
 8            π ← PRPTSRST_ARESTA_TIPO_1_2(π, n)
 9       else if π = η_n  then
10            π ← π · ρ_p(n)
11       else
12            Seja k a posição do último elemento da primeira strip de π
13            π ← π · τ_p(k + 1, n + 1)
14       d ← d + 1
15  return  d
```

Algoritmo 26 Ordenando π_n^{19} com reversões de prefixo, transposições de prefixo, reversões de sufixo e transposições de sufixo

PRPTSRST_Familia_19 ($\pi = \pi_n^{19}$, $\mathtt{n} \geq 6$)

```
1  if  n mod 2 = 0  then
2       while  π_1 ≠ 5  do
3            π ← π · τ_p(2, π_{π_1+1}^{-1})
4       π ← π · τ_p(n − 1, n) · τ_p(3, 4) · τ_p(n − 1, n + 1) · ρ_p(2)
5  else
6       π ← π · τ_p(π_3^{-1}, π_4^{-1}) · τ_s(π_2^{-1}, π_{n-2}^{-1}) · τ_p(2, π_1^{-1})
7       while  π_1 ≠ n − 1  do
8            π ← π · τ_p(2, π_{π_1−1}^{-1})
9       π ← π · ρ_p(n − 1) · ρ_p(2)
```

Teorema 94. *Para $n \geq 6$, $D_{\rho_p\tau_p\rho_s\tau_s}(n) \geq \lceil \frac{n}{2} \rceil$ e para $n \geq 1$, $D_{\rho_p\tau_p\rho_s\tau_s}(n) \leq n - \log_{\frac{9}{2}} n$.*

Demonstração. O limitante inferior é verdadeiro devido à família π_n^{19}, conforme o Lema 93 mostra. O limitante superior é verdadeiro porque $D_{\rho_p\tau_p\rho_s\tau_s}(n) \leq \min\{D_{\rho_p\rho_s}(n),\ D_{\tau_p\tau_s}(n)\} \leq \min\{D_{\rho_p}(n),\ D_{\tau_p}(n)\}$, uma vez que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \min\{d_{\rho_p\rho_s}(\pi),\ d_{\tau_p\tau_s}(\pi)\} \leq \min\{d_{\rho_p}(\pi),\ d_{\tau_p}(\pi)\}$ para qualquer π. □

A.13 Problema de Ordenação por Reversões de Prefixo com Sinal, Transposições de Prefixo, Reversões de Sufixo com Sinal e Transposições de Sufixo

Lema 95. *Seja π uma permutação com sinal qualquer. Então, $d_{\bar{\rho_p}\tau_p\bar{\rho_s}\tau_s}(\pi) \geq \frac{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}{2}$.*

Demonstração. Uma reversão de prefixo $\bar{\rho_p}(i)$ separa os pares de elementos $[\pi_0, \pi_1]$ e $[\pi_i, \pi_{i+1}]$, podendo criar ou remover *breakpoints* de reversão de prefixo com sinal e reversão de sufixo com sinal apenas no último par, uma vez que (π_0, π_1) nunca é considerado *breakpoint* desse tipo. Assim, $\Delta b_{\bar{\rho_p}\bar{\rho_s}}(\pi, \bar{\rho_p}) \in \{-1, 0, 1\}$. Equivalentemente, uma reversão de sufixo $\bar{\rho_s}(j)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas no primeiro par, uma vez que (π_n, π_{n+1}) nunca é considerado *breakpoint* de reversão de prefixo com sinal e reversão de sufixo com sinal. Assim, $\Delta b_{\bar{\rho_p}\bar{\rho_s}}(\pi, \bar{\rho_s}) \in \{-1, 0, 1\}$. Uma transposição de prefixo $\tau_p(i, j)$ separa os pares de elementos $[\pi_0, \pi_1]$, $[\pi_{i-1}, \pi_i]$ e $[\pi_{j-1}, \pi_j]$, podendo criar ou remover *breakpoints* apenas nos dois últimos pares. Assim, $\Delta b_{\bar{\rho_p}\bar{\rho_s}}(\pi, \tau_p) \in \{-2, -1, 0, 1, 2\}$. Por fim, uma transposição de sufixo $\tau_s(j, k)$ separa os pares de elementos $[\pi_{j-1}, \pi_j]$, $[\pi_{k-1}, \pi_k]$ e $[\pi_n, \pi_{n+1}]$, podendo criar ou remover *breakpoints* apenas nos dois primeiros pares. Assim, $\Delta b_{\bar{\rho_p}\bar{\rho_s}}(\pi, \tau_s) \in \{-2, -1, 0, 1, 2\}$. □

Lema 96. *Seja $\pi \neq \iota_n$ uma permutação com sinal para a qual não é possível remover um ou dois breakpoints com uma operação. Então π é de uma das cinco formas:*

$$\bar{\eta_n} \tag{33}$$

ou

$$\mu^1 = (\underline{1\ 2\ \ldots\ k}\ \ldots\ -(k+1)\ \ldots\ -(i-1)\ \ldots\ \underline{i\ i+1\ \ldots\ n}) \tag{34}$$

isto é, $\pi_1 = 1$, $\pi_n = n$, e os elementos que podem aumentar a primeira e a última strips são negativos (a posição relativa entre $-(k+1)$ e $-(i-1)$ é irrelevante). Ou

$$\mu^2 = (\underline{-n\ -(n-1)\ldots -i}\ \ldots\ (i-1)\ \ldots\ (k+1)\ \ldots\ \underline{-k\ -(k-1)\ldots -1}) \tag{35}$$

isto é, $\pi_1 = -n$, $\pi_n = -1$, e os elementos que podem aumentar a primeira e a última strips são

positivos (a posição relativa entre $i-1$ e $k+1$ é irrelevante). Ou

$$\mu^3 = (\underline{k+1\ k+2\ \dots\ n}\ \underline{1\ 2\ \dots\ k}) \tag{36}$$

ou então

$$\mu^4 = (\underline{-k\ -(k-1)\ \dots\ -1}\ \underline{-n\ -(n-1)\ \dots\ -(k+1)}). \tag{37}$$

Em todos os casos, $1 \leq k$ *e* $k+1 < i \leq n$.

Demonstração. É fácil ver que quando $\pi = \bar{\eta_n}$, os primeiros três passos principais do algoritmo não podem ser realizados. Devemos explicitamente fazer $\bar{\rho_p}(n)$ para ordená-la.

Suponha que $\pi_1 = \ell$, para $\ell \neq 1$ e $\ell \neq -n$, independente do valor de π_n. Ou $+(\ell-1)$ ou $-(\ell-1)$ deve existir em π. Mas nesse caso, uma transposição ou uma reversão podem ser realizadas para remover um *breakpoint*, o que é uma contradição. Agora suponha que $\pi_n = \ell$, para $\ell \neq n$ e $\ell \neq -1$, independente do valor de π_1. Ou $-(\ell+1)$ ou $+(\ell+1)$ deve existir em π. Mas nesse caso, uma reversão ou uma transposição também podem ser realizadas para remover um *breakpoint*, o que é uma contradição.

Como π_1 pode apenas ser 1 ou $-n$ e π_n pode ser apenas n ou -1, μ^1 e μ^2 são claramente as únicas possíveis formas que permitem isso. Além disso, se os elementos que podem aumentar a primeira e a última *strips* não fossem de sinais opostos com relação aos sinais de tais *strips*, uma transposição que remove um *breakpoint* poderia ser feita.

É fácil ver que, como o algoritmo não permite que n e 1 ou -1 e $-n$ sejam separados, os casos quando $\pi = \mu^3$ ou $\pi = \mu^4$ não podem ser lidados pelos três primeiros casos principais. □

Lema 97. *Seja π uma permutação com sinal conforme descrito no Lema 96 tal que $\pi \neq \bar{\eta_n}$. Então, uma transposição que concatena a primeira strip com a última não cria novos breakpoints e garante que as próximas operações sempre conseguirão remover pelo menos um breakpoint.*

Demonstração. Seja $\pi = \mu^1$. Para concatenar as *strips*, a transposição $\tau_p(k+1, n+1)$ é suficiente. Então, $\pi' = \pi \cdot \tau_p(k+1, n+1) = (\dots -(k+1) \dots -(i-1) \dots \underline{i\ i+1\ \dots\ n}\ \underline{1\ 2\ \dots\ k})$. Quando $\pi = \mu^2$, a transposição $\tau_p(n-i+2, n+1)$ é suficiente e $\pi' = \pi \cdot \tau_p(n-i+2, n+1) = (\dots k+1 \dots j-1 \dots \underline{-k\ -(k-1)\ \dots\ -1}\ \underline{-n\ -(n-1)\ \dots\ -i})$. Em ambos os casos, π' não é de nenhuma das formas dadas no Lema 96. Portanto, é possível remover pelo menos um *breakpoint* com uma operação.

Note que μ^1 e μ^2 aparecerão novamente quando $\pi_1 = 1$ e $\pi_n = n$ ou quando $\pi_1 = -n$ e $\pi_n = -1$. No entanto, isso pode acontecer apenas se o algoritmo separar os elementos n e 1 ou -1 e $-n$, o que não é permitido. Por outro lado, essa separação terá que acontecer quando $\pi = \mu^3$ e $\pi = \mu^4$. Mas nesse ponto, o algoritmo estará realizando sua última ou penúltima operação: quando $\pi = \mu^3$, a concatenação das *strips* levará a ι_n e quando $\pi = \mu^4$, a concatenação levará a $\bar{\eta_n}$, que será seguida diretamente por uma reversão que a ordenará. □

Algoritmo 27 Algoritmo assintótico de 2-aproximação para SbSigPRPTSigSRST

2-SPRPTSSRST(π, $\mathtt{n}$)

```
1  d ← 0
2  while  π ≠ ι_n  do
3      // Tenta remover dois breakpoints com uma transposição de prefixo
4      if  existe π_{j−1} = π_1 − 1  and existe π_{i−1} = π_j − 1  and 2 ≤ i < j ≤ n  then
5          π ← π · τ_p(i, j)
6      // Tenta remover dois breakpoints com uma transposição de sufixo
7      else if existe π_i = π_n + 1  and existe π_j = π_{i−1} + 1  and 2 ≤ i < j ≤ n  then
8          π ← π · τ_s(i, j)
9      // Tenta remover um breakpoint
10     else
11         Seja ip a posição do último elemento da primeira strip de π
12         Seja js a posição do primeiro elemento da última strip de π
13         if  existe π_j = π_ip + 1  and j ≤ n  then
14             π ← π · τ_p(ip + 1, j)
15         else if existe π_j = π_1 − 1  and j > 1  then
16             π ← π · τ_p(ip + 1, j + 1)
17         else if existe π_i = π_js − 1  and i ≥ 2  then
18             π ← π · τ_p(i + 1, j)
19         else if existe π_i = π_n + 1  and i < n  then
20             π ← π · τ_p(i, j)
21         else if existe π_j = −π_1 + 1  and 2 ≤ j ≤ n  then
22             π ← π · ρ̄_p(j − 1)
23         else if existe π_i = −π_n − 1  and 1 ≤ i ≤ n − 1  then
24             π ← π · ρ̄_s(i + 1)
25         // formas especiais
26         else if π = η̄_n  then
27             π ← π · ρ̄_p(n)
28         else
29             π ← π · τ_p(ip, n + 1)
30     d ← d + 1
31 return  d
```

Lema 98. *Seja π uma permutação qualquer. Então, $d_{\bar{\rho_p}\tau_p\bar{\rho_s}\tau_s}(\pi) \leq b_{\bar{\rho_p}\bar{\rho_s}}(\pi) + 2$.*

Demonstração. Enquanto π não for das formas descritas no Lema 96, pode-se remover pelo menos um *breakpoint* com uma operação. Duas operações extras que não removem nem criam *breakpoints* podem ser necessárias: uma para lidar com μ^1 ou μ^2 e uma para lidar com $\bar{\eta_n}$ (depois que μ^4 for alcançada ou simplesmente quando a própria $\bar{\eta_n}$ é alcançada). Portanto, no máximo $b_{\bar{\rho_p}\bar{\rho_s}}(\pi) + 2$ operações ordenam qualquer permutação. □

Teorema 99. SbSigPRPTSigSRST *é 2-aproximável assintoticamente.*

Demonstração. Dos Lemas 95 e 98, temos que o fator de aproximação teórico é $\frac{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)+2}{\frac{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}{2}}$. Para $b_{\bar{\rho_p}\bar{\rho_s}}(\pi)$ suficientemente grande, temos

$$\lim_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)\to\infty} \frac{b_{\bar{\rho_p}\bar{\rho_s}}(\pi) + 2}{\frac{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)}{2}} = \lim_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)\to\infty} \frac{2b_{\bar{\rho_p}\bar{\rho_s}}(\pi) + 4}{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)} = 2 + \lim_{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)\to\infty} \frac{4}{b_{\bar{\rho_p}\bar{\rho_s}}(\pi)} = 2 + \epsilon.$$

□

B Resultados

Todos os algoritmos explicados foram implementados em C e executados sobre um mesmo conjunto de instâncias. Foram criados dois conjuntos (um com sinal e um sem) com 1990000 permutações arbitrárias cada um, sendo 10000 de cada tamanho n, com n variando entre 10 e 1000 em intervalos de 5. Com exceção dos algoritmos cujos nomes terminam em `x`, que executaram apenas sobre as primeiras 590000 permutações, com n até 300, todos os outros algoritmos executaram sobre todas as permutações desses conjuntos. Todas as permutações são compostas apenas por *singletons*. Além disso, testamos os algoritmos para problemas sem sinal sobre todas as permutações de tamanho até 10 e os algoritmos para problemas com sinal sobre todas as permutações de tamanho até 9. As Figuras 3 a 12 apresentam os resultados. Os eixos x dos gráficos apresentam um valor de n e os eixos y apresentam a média dos fatores de aproximação de todas as permutações daquele tamanho. Tal fator foi calculado pela divisão do tamanho da sequência de ordenação dada pelo algoritmo com o limitante inferior teórico do devido problema.

Como alguns dos algoritmos desenvolvidos são assintóticos de fator 2, é esperado que algumas vezes o fator de aproximação seja acima de 2. Em todos eles, sempre existem permutações de tamanho menor do que 9 ou 10 (dependendo se o problema é com sinal ou sem, respectivamente) para as quais o fator ficou acima de 2. Para `2-SPRPT`, isso aconteceu em 30,48% das permutações enquanto que para sua versão melhorada, `2-SPRPTx`, isso aconteceu em apenas 3,78%. Para `2-PRSR`, `2-PRSRg` e `2-PRSRx`, a porcentagem de permutações foi muito pequena, não ultrapassando 0,01%. Curiosamente, em `2-PRSRg` isso aconteceu em 23 permutações a mais do que em `2-PRSR`, sendo que, de forma geral, o primeiro apresenta melhores resultados do que o segundo. Em `2-SPRSSR`, 0,06% das permutações apresentaram fator maior do que 2 enquanto que em `2-SPRSSRx` essa porcentagem foi menor do que 0,01%. Para `2-PRPTSRST`, `2-PRPTSRSTg` e `2-PRPTSRSTx`, essas porcentagens foram de 22,03%, 3,12% e 0,15%, respectivamente. Finalmente, para `2-SPRPTSSRST` e `2-SPRPTSSRSTx`, isso aconteceu em 35,23% e 1,25% das permutações de tamanho pequeno, respectivamente.

Com relação às permutações de tamanho entre 10 e 1000 que foram testadas, para o `2-SPRPT`, apenas 0,41% delas tiveram fator de aproximação acima de 2 e isso aconteceu apenas quando $n \leq 100$. Desta quantidade, 98,65% aconteceram quando $n \leq 50$. Para o `2-SPRPTx`, 0,02% das permutações apresentaram essa característica e apenas quando $n = 10$ e $n = 15$. Lembrando que para este último, 590000 permutações foram testadas enquanto que para o primeiro, 1990000 permutações foram testadas. Para `2-SPRSSR`, apenas uma dessas permutações, de tamanho 10, ficou com fator de aproximação maior do que 2. Para `2-SPRPTSSRST`, 0,44% das permutações testadas apresentaram essa característica, todas elas quando $n \leq 105$, sendo que 99,35% destas aconteceram quando $n \leq 50$. Em sua versão melhorada, `2-SPRPTSSRSTx`, apenas 29 das permutações testadas apresentaram fator maior do que 2, todas elas com tamanho 10. Para SbPRPTSRST o resultado obtido foi ainda mais interessante. Em `2-PRPTSRST`, sempre houve permutações com fator de aproximação acima de 2, de forma que 2,44% das permutações tes-

tadas apresentaram essa característica. Sua primeira versão melhorada, `2-PRPTSRSTg`, fez tal porcentagem diminuir para menos de 0,01% e as ocorrências foram apenas quando $n = 10$ e $n = 15$. Sua segunda versão melhorada, `2-PRPTSRSTx`, não obteve nenhuma permutação com tal característica. Nos demais algoritmos, `2-PRSR`, `2-PRSRg`, `2-PRSRx` e `2-SPRSSRx`, nenhuma das permutações de tamanho grande que foram testadas apresentaram fator acima de 2.

De forma geral, para os problemas sem sinal podemos ver que a simples mudança na forma como a permutação é percorrida gera melhores resultados na prática. Isso significa que operações maiores (especialmente reversões) são preferíveis. Como esperado, problemas que envolvem apenas rearranjos de prefixo são equivalente às suas versões em sufixo. Além disso, era esperado que os problemas nos quais ambas versões de prefixo e sufixo fossem permitidas obteriam melhores resultados do que aqueles para os quais apenas a versão em prefixo é permitida. É interessante notar que isso não acontece para o `2-PTST`.

Destacamos também que todos os algoritmos descritos na Seção 6.10 realmente obtiveram melhores resultados na prática. As Figuras 8 a 12 apresentam seus resultados juntamente com os resultados dos melhores algoritmos, até então, para comparação. Todos os fatores de aproximação obtidos por estes algoritmos, inclusive para permutações de tamanhos pequenos, foram reduzidos com relação aos algoritmos anteriores. De fato, quando comparamos o tamanho da sequência de ordenação que cada algoritmo encontrou para cada uma das 590000 permutações de tamanho entre 10 e 300 que foram testadas, aconteceu que `2-PTSTg` obteve alguns valores melhores do que `2-PTSTx`, `2-PRPTSRSTg` obteve alguns valores melhores do que `2-PRPTSRSTx` e `2-SPRPTSSRST` obteve alguns valores melhores do que `2-SPRPTSSRSTx`. Mesmo assim, isso aconteceu em menos de 31, 44 e 4 permutações para cada valor de n, respectivamente. Para todos os outros problemas, o algoritmo cujo nome termina em `x` sempre obteve melhores resultados. As Tabelas 3 e 4 mostram a média de operações que são feitas a menos para cada valor de n.

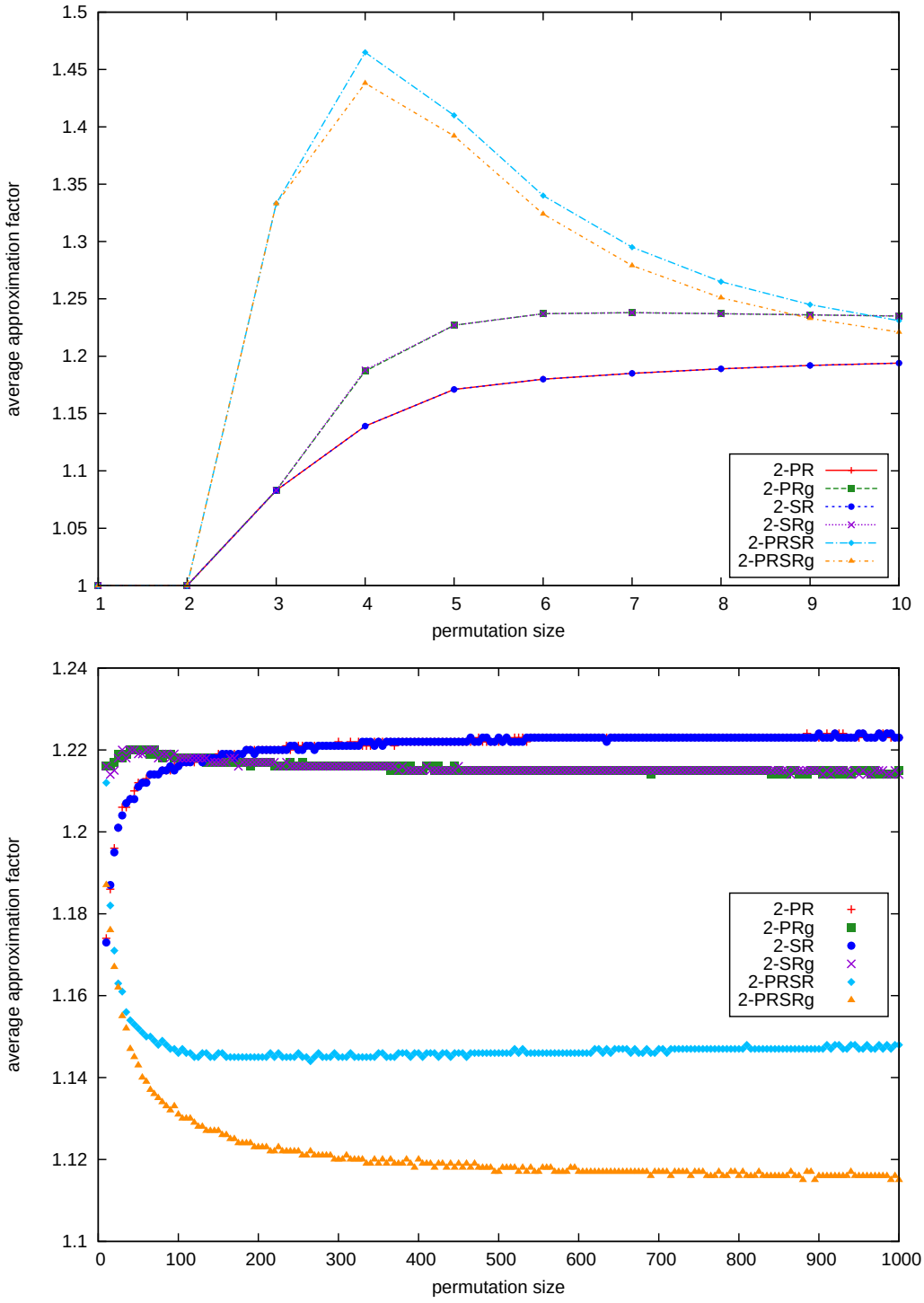


Figura 3: Fatores médios de aproximação de `2-PR`, `2-PRg`, `2-SR`, `2-SRg`, `2-PRSR` e `2-PRSTg` conforme o tamanho da permutação aumenta

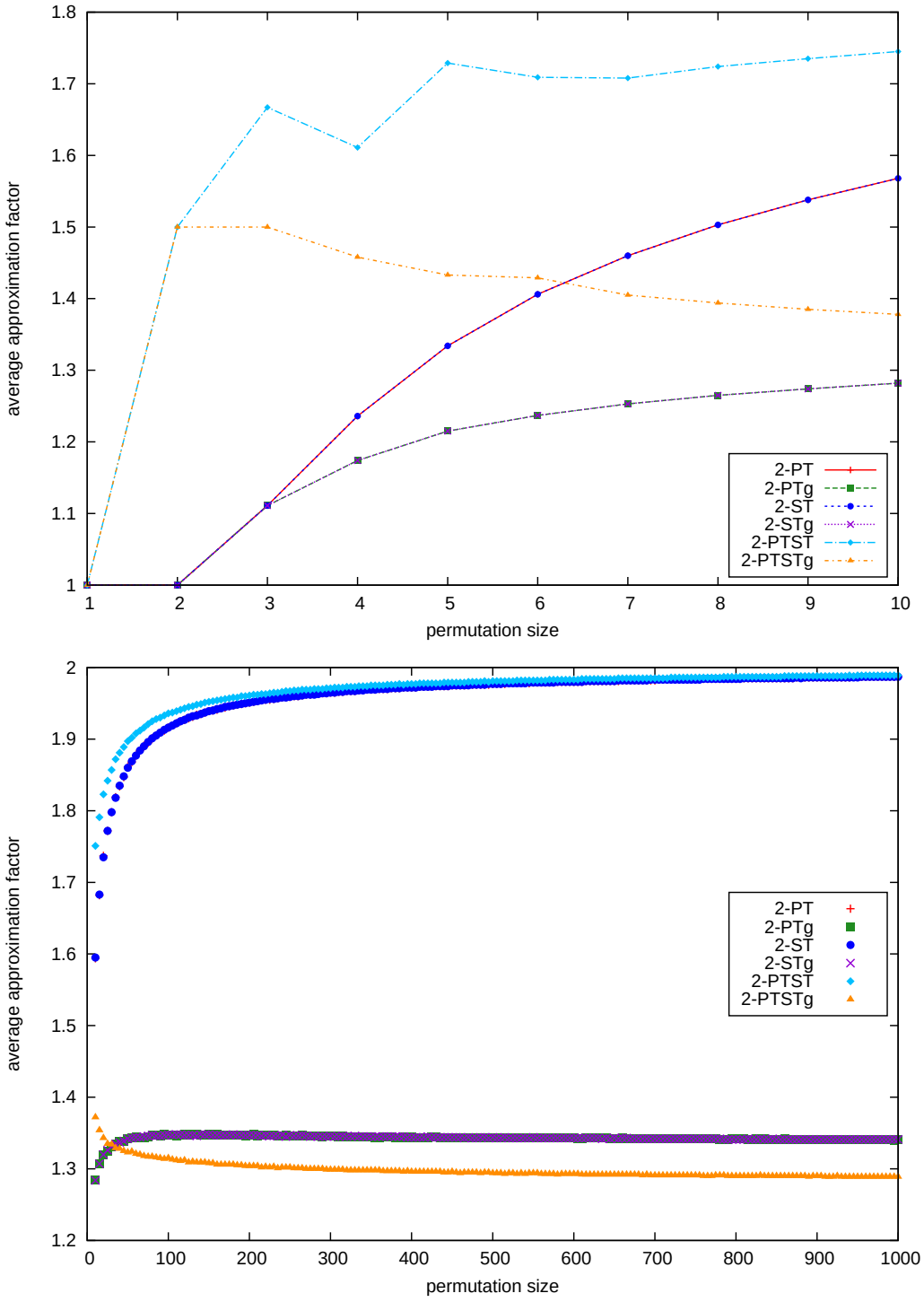


Figura 4: Fatores médios de aproximação de `2-PT`, `2-PTg`, `2-ST`, `2-STg`, `2-PTST` e `2-PTSTg` conforme o tamanho da permutação aumenta

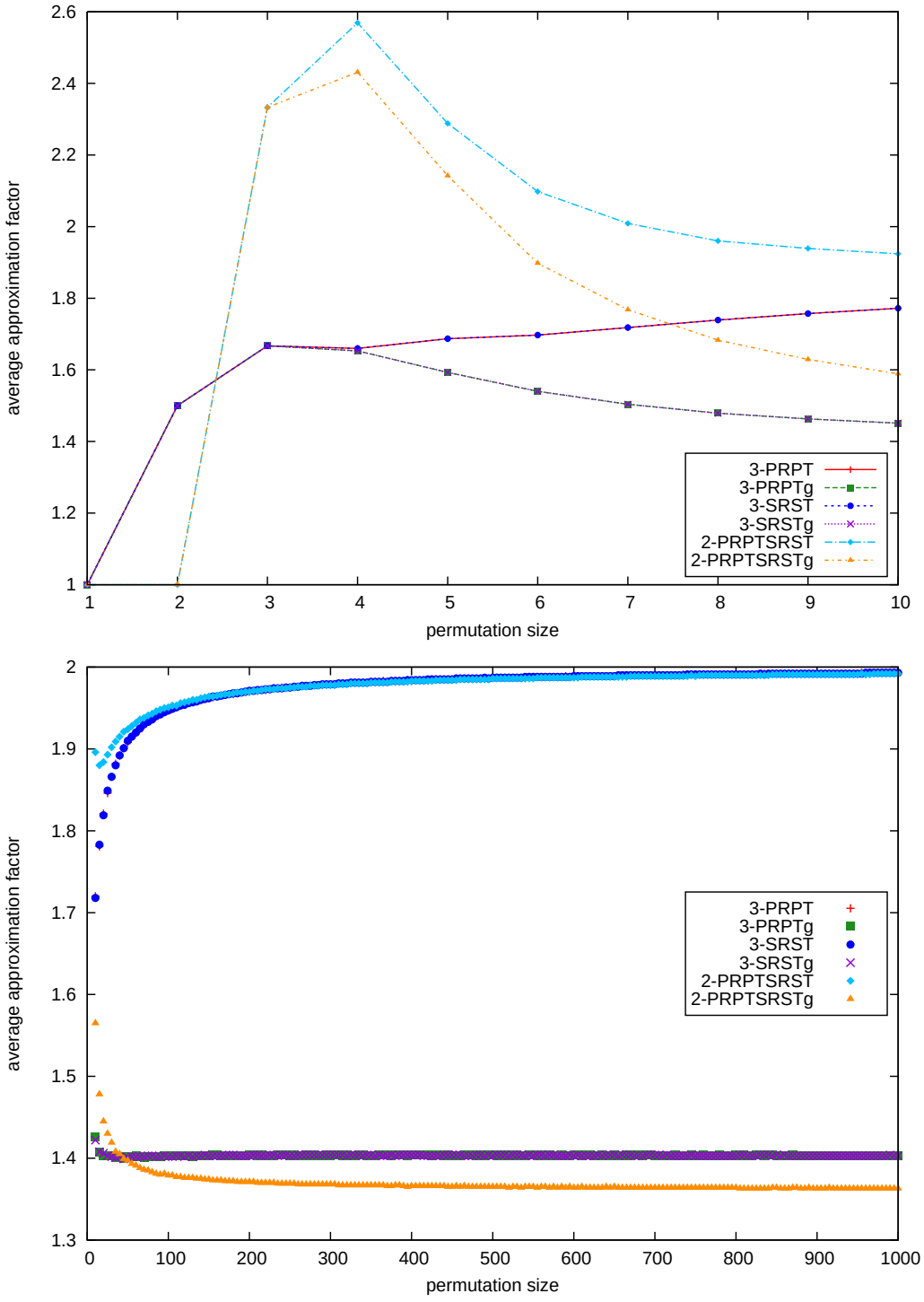


Figura 5: Fatores médios de aproximação de `3-PRPT`, `3-PRPTg`, `3-SRST`, `3-SRSTg`, `2-PRPTSRST` e `2-PRPTSRSTg` conforme o tamanho da permutação aumenta

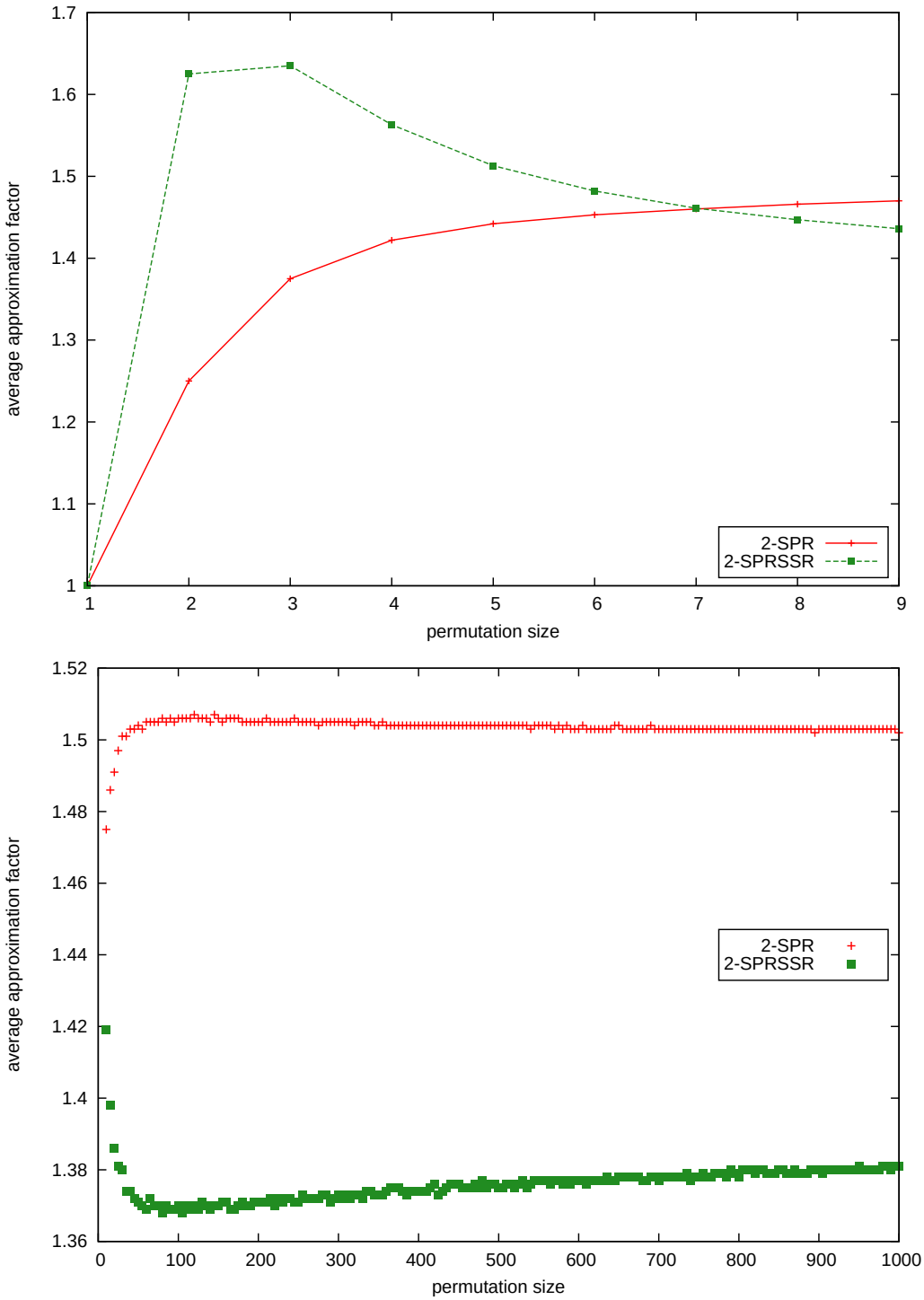


Figura 6: Fatores médios de aproximação de `2-SPR` e `2-SPRSSR` conforme o tamanho da permutação aumenta

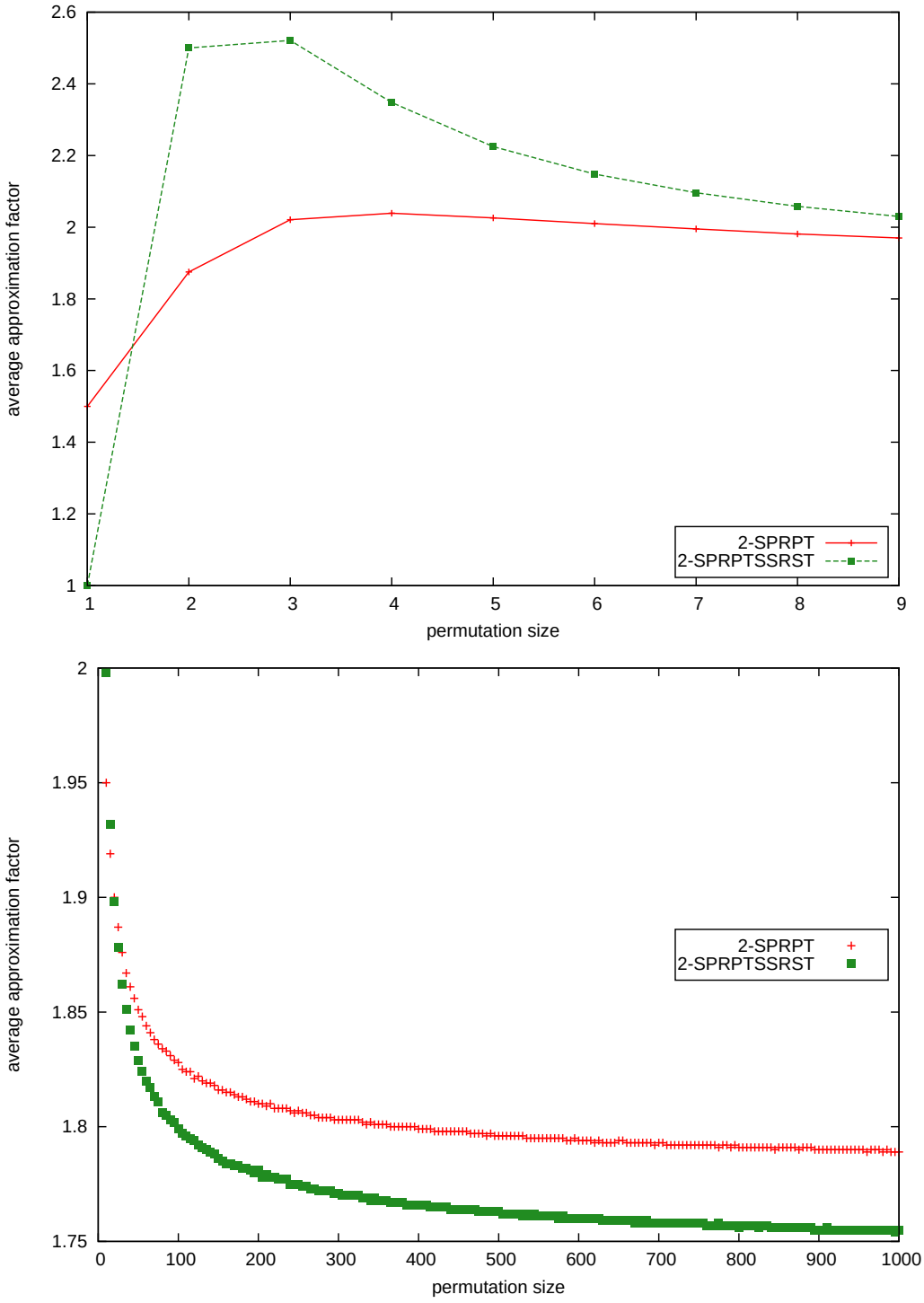


Figura 7: Fatores médios de aproximação de `2-SPRPT` e `2-SPRPTSSRST` conforme o tamanho da permutação aumenta

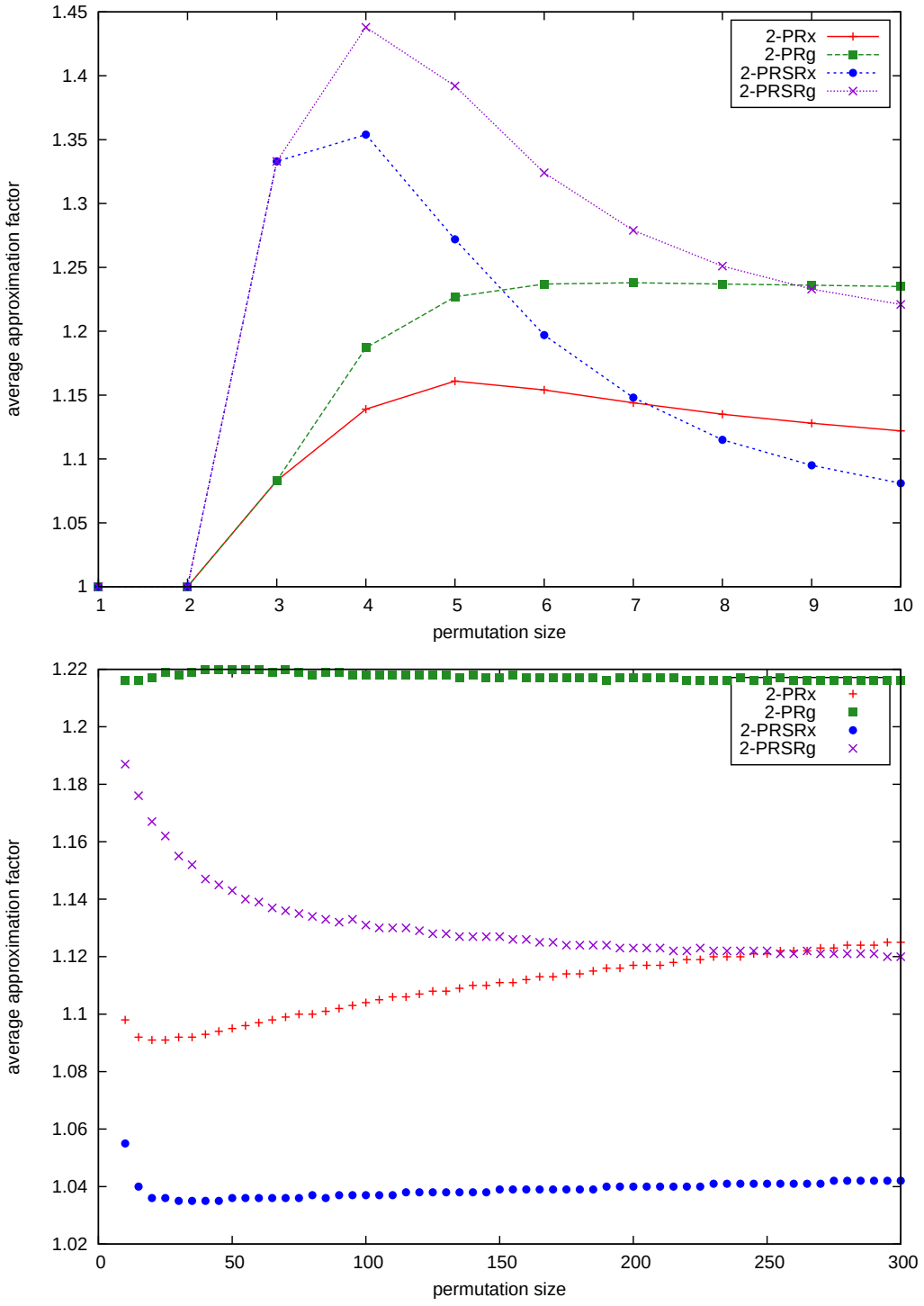


Figura 8: Fatores médios de aproximação de `2-PRx` e `2-PRSRx` conforme o tamanho da permutação aumenta. Os resultados de `2-PRg` e `2-PRSRg` são apresentados novamente, para comparação

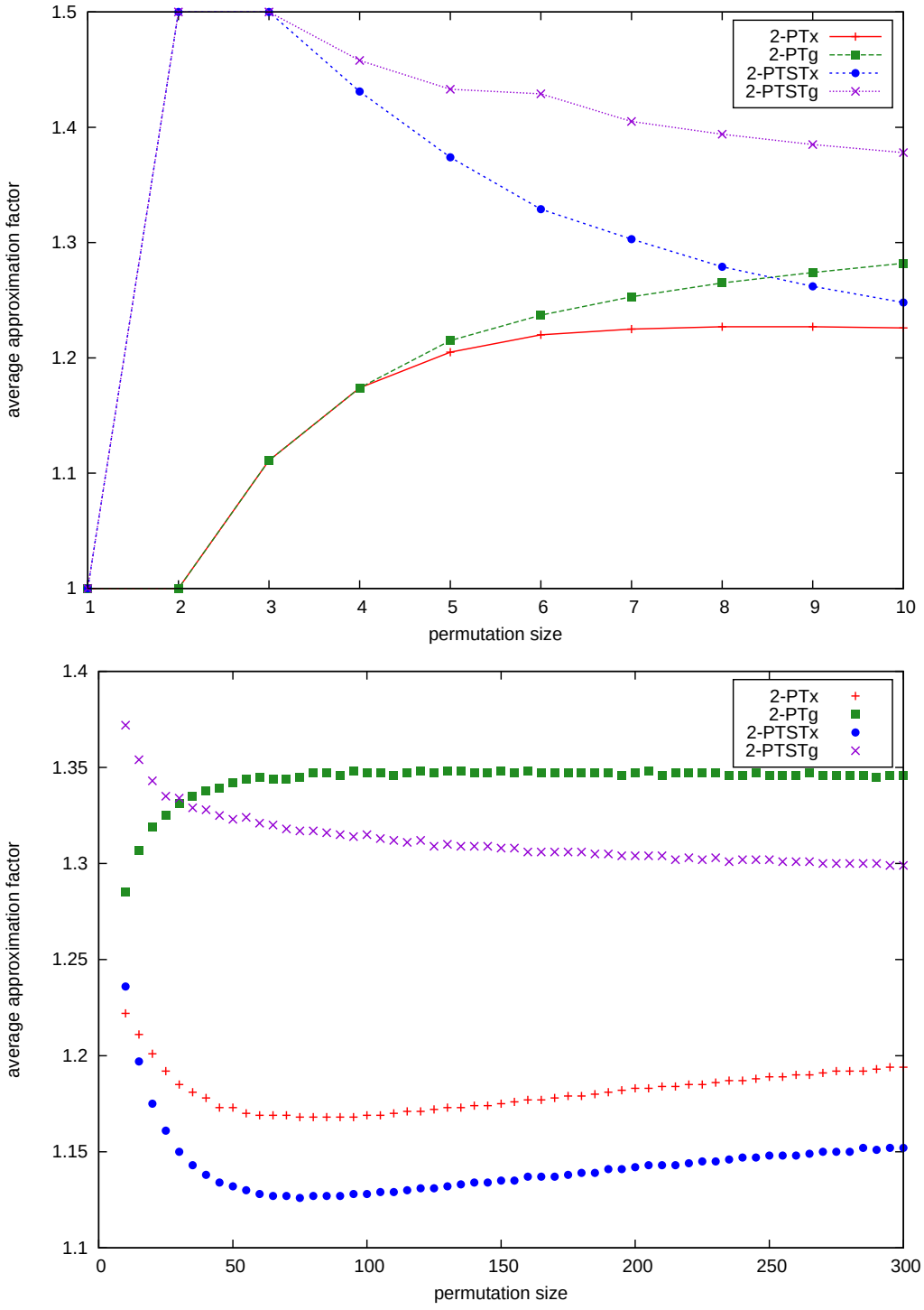


Figura 9: Fatores médios de aproximação de `2-PTx` e `2-PTSTx` conforme o tamanho da permutação aumenta. Os resultados de `2-PTg` e `2-PTSTg` são apresentados novamente, para comparação

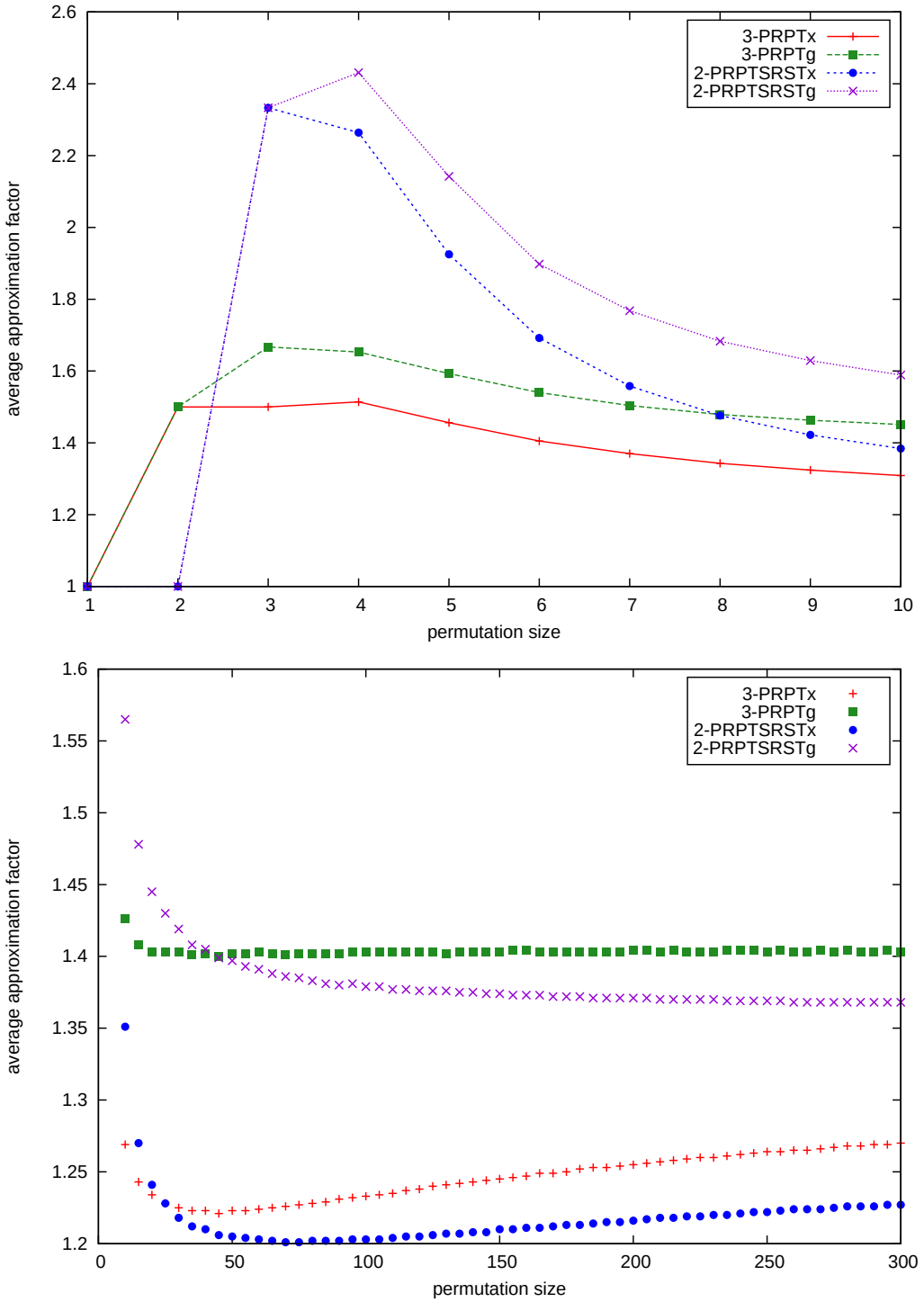


Figura 10: Fatores médios de aproximação de `3-PRPTx` e `2-PRPTSRSTx` conforme o tamanho da permutação aumenta. Os resultados de `3-PRPTg` e `2-PRPTSRSTg` são apresentados novamente, para comparação

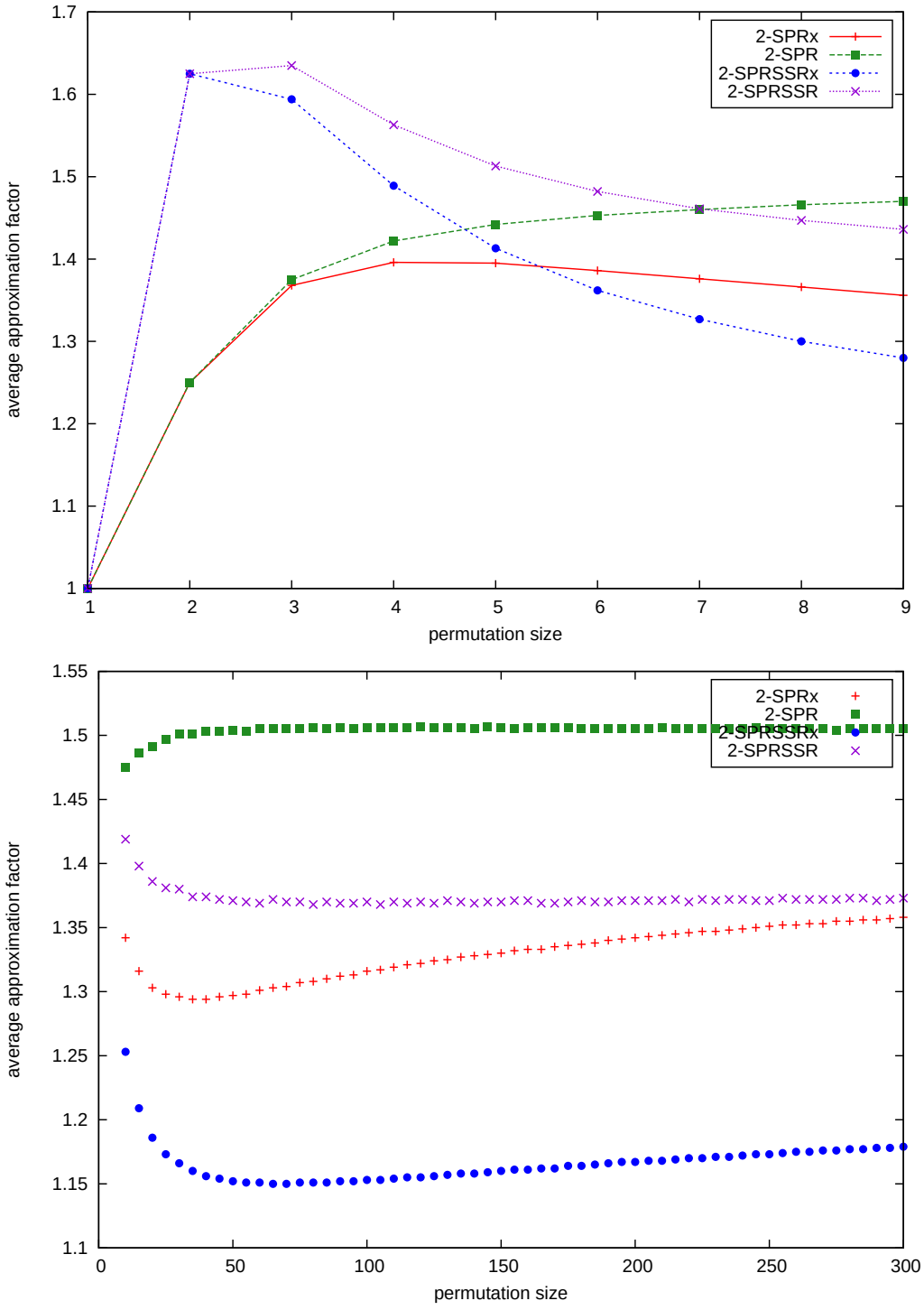


Figura 11: Fatores médios de aproximação de `2-SPRx` e `2-SPRSSRx` conforme o tamanho da permutação aumenta. Os resultados de `2-SPR` e `2-SPRSSR` são apresentados novamente, para comparação

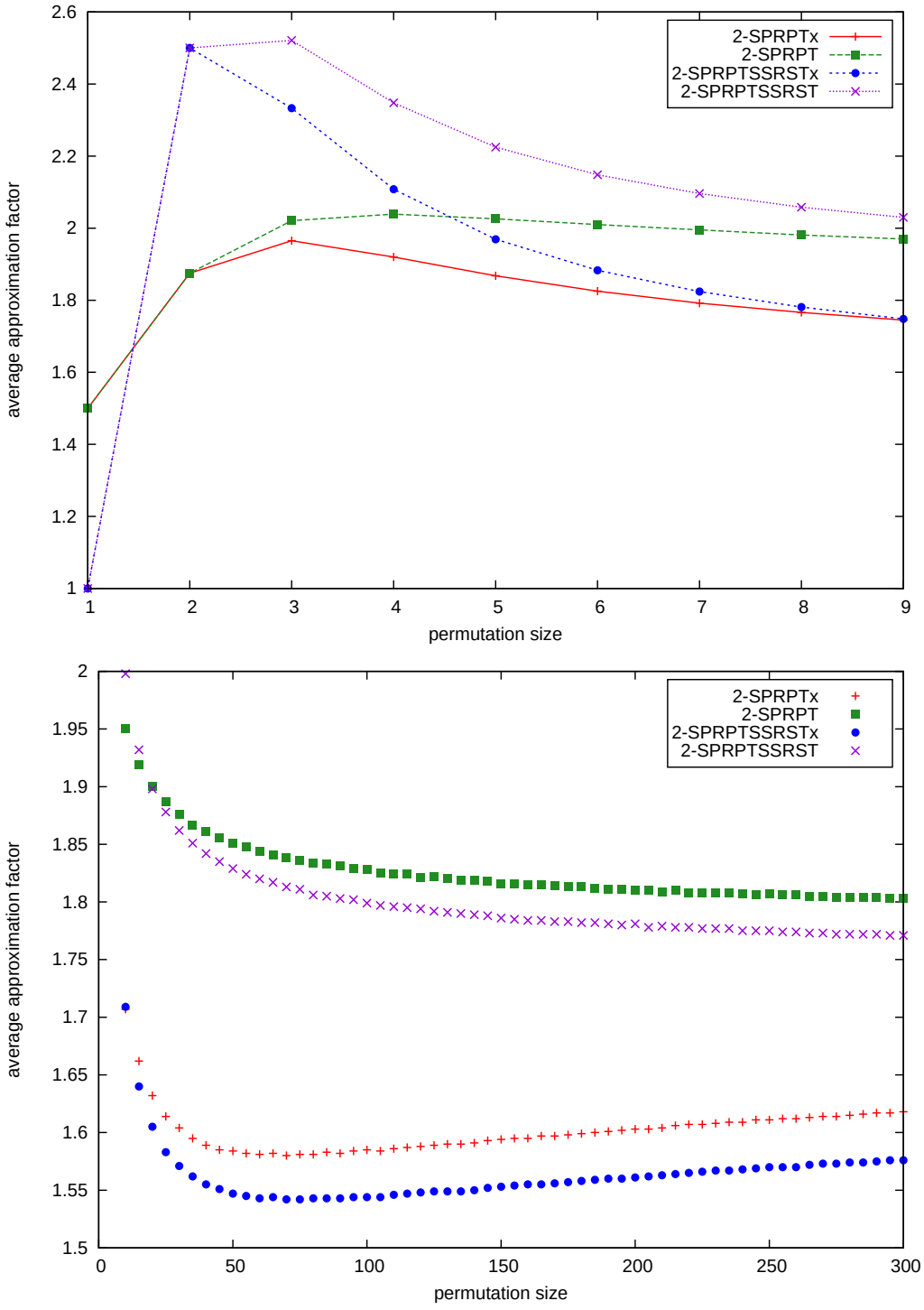


Figura 12: Fatores médios de aproximação de `2-SPRPTx` e `2-SPRPTSSRSTx` conforme o tamanho da permutação aumenta. Os resultados de `2-SPRPT` e `2-SPRPTSSRST` são apresentados novamente, para comparação

Tabela 3: Número médio de operações feitas a menos pelos algoritmos 2-PRx, 2-PRSRx, 2-PTx, 2-PTSTx, 3-PRPTx e 2-PRPTSRSTx com relação às operações feitas pelos algoritmos 2-PRg, 2-PRSRg, 2-PTg, 2-PTSTg, 3-PRPTg e 2-PRPTSRSTg, respectivamente. Entre parênteses aparece a quantidade de permutações do tamanho correspondente para as quais a redução não foi possível

n	2-PRx × 2-PRg	2-PRSRx × 2-PRSRg	2-PTx × 2-PTg	2-PTSTx × 2-PTSTg	3-PRPTx × 3-PRPTg	2-PRPTSRSTx × 2-PRPTSRSTg
10	1.059	1.048	0.311	0.619 (27)	0.683	0.839 (44)
15	1.730	1.761	0.719	1.102 (31)	1.134	1.337 (35)
20	2.372	2.345	1.182	1.599 (22)	1.591	1.826 (39)
25	3.066	2.890	1.672	2.100 (29)	2.078	2.323 (25)
30	3.641	3.361	2.190	2.672 (29)	2.558	2.798 (13)
35	4.304	3.858	2.703	3.184 (28)	3.021	3.234 (15)
40	4.921	4.261	3.212	3.719 (23)	3.482	3.722 (13)
45	5.520	4.702	3.741	4.224 (24)	3.925	4.137 (7)
50	6.118	5.131	4.242	4.711 (25)	4.391	4.615 (9)
55	6.718	5.542	4.785	5.253 (30)	4.836	5.009 (7)
60	7.240	6.003	5.267	5.707 (20)	5.265	5.439 (3)
65	7.781	6.367	5.714	6.169 (16)	5.653	5.876 (3)
70	8.341	6.788	6.145	6.615 (12)	6.045	6.280 (4)
75	8.805	7.233	6.641	7.057 (14)	6.465	6.717 (5)
80	9.303	7.639	7.147	7.515 (11)	6.880	7.084 (4)
85	9.846	7.991	7.604	7.978 (13)	7.276	7.452 (1)
90	10.409	8.337	8.018	8.401 (16)	7.627	7.826
95	10.821	8.882	8.551	8.767 (17)	8.044	8.274 (3)
100	11.219	9.168	8.881	9.256 (13)	8.409	8.660 (2)
105	11.774	9.560	9.342	9.575 (13)	8.758	9.046 (2)
110	12.237	9.974	9.684	10.001 (15)	9.109	9.316 (2)
115	12.733	10.455	10.166	10.364 (9)	9.464	9.732 (1)
120	13.112	10.788	10.572	10.779 (11)	9.812	10.084
125	13.573	11.125	10.948	11.056 (11)	10.115	10.477 (1)
130	14.117	11.470	11.421	11.477 (9)	10.416	10.830 (3)
135	14.491	11.761	11.767	11.817 (13)	10.808	11.133 (1)
140	15.015	12.195	12.162	12.152 (9)	11.144	11.502 (2)
145	15.368	12.693	12.534	12.620 (10)	11.486	11.859 (1)
150	15.797	13.098	12.955	12.910 (6)	11.768	12.173
155	16.360	13.287	13.316	13.281 (6)	12.119	12.512
160	16.704	13.735	13.720	13.496 (5)	12.441	12.825
165	17.035	14.071	14.012	13.889 (10)	12.660	13.172
170	17.562	14.384	14.385	14.217 (3)	12.986	13.415 (1)
175	17.888	14.702	14.730	14.552 (5)	13.286	13.776 (1)
180	18.422	15.034	15.093	14.899 (2)	13.599	14.103
185	18.796	15.524	15.444	15.268 (6)	13.871	14.301
190	19.051	15.812	15.779	15.544 (9)	14.136	14.714
195	19.558	16.156	16.057	15.820 (4)	14.454	15.064
200	19.924	16.401	16.454	16.155 (1)	14.778	15.336
205	20.388	16.810	16.864	16.444 (2)	15.036	15.649 (1)
210	20.717	17.293	17.077	16.869 (1)	15.278	15.905
215	21.041	17.495	17.473	17.022 (2)	15.629	16.210
220	21.348	17.873	17.844	17.362 (2)	15.842	16.543
225	21.872	18.413	18.171	17.667 (1)	15.998	16.836
230	22.085	18.622	18.454	18.043 (4)	16.394	17.089
235	22.610	18.920	18.732	18.153 (3)	16.672	17.309
240	22.999	19.311	19.059	18.545 (4)	16.932	17.642
245	23.254	19.726	19.438	18.981 (2)	17.159	17.916
250	23.652	19.934	19.689	19.082	17.351	18.151
255	24.150	20.216	20.045	19.400 (2)	17.695	18.497
260	24.367	20.577	20.297	19.761 (5)	17.892	18.671
265	24.782	21.186	20.716	20.023 (4)	18.222	18.992
270	25.122	21.400	20.994	20.249 (3)	18.495	19.298
275	25.609	21.679	21.264	20.569 (1)	18.669	19.593 (1)
280	25.882	21.976	21.551	20.863 (5)	18.982	19.857
285	26.168	22.423	21.941	21.072 (4)	19.180	20.115 (1)
290	26.611	22.725	22.047	21.411 (1)	19.407	20.390
295	26.888	22.947	22.484	21.696 (1)	19.715	20.625
300	27.292	23.321	22.845	21.945	19.935	20.913

Tabela 4: Número médio de operações feitas a menos pelos algoritmos `2-SPRx`, `2-SPRSSRx`, `2-SPRPTx` e `2-SPRPTSSRSTx` com relação às operações feitas pelos algoritmos `2-SPR`, `2-SPRSSR`, `2-SPRPT` e `2-SPRPTSSRST`, respectivamente. Entre parênteses aparece a quantidade de permutações do tamanho correspondente para as quais a redução não foi possível

n	`2-SPRx` × `2-SPR`	`2-SPRSSRx` × `2-SPRSSR`	`2-SPRPTx` × `2-SPRPT`	`2-SPRPTSSRSTx` × `2-SPRPTSSRST`
10	1.322	1.490	1.217	1.302 (1)
15	2.537	2.648	1.929	2.043 (1)
20	3.756	3.802	2.676	2.788 (1)
25	4.983	5.000	3.414	3.541 (4)
30	6.133	6.214	4.080	4.218 (1)
35	7.279	7.295	4.776	4.905
40	8.354	8.484	5.448	5.592 (1)
45	9.324	9.601	6.089	6.242
50	10.365	10.688	6.682	6.905
55	11.241	11.827	7.321	7.540
60	12.203	12.851	7.891	8.159
65	13.154	14.178	8.421	8.729 (1)
70	14.045	15.138	9.008	9.351
75	14.886	16.244	9.574	9.929 (1)
80	15.834	17.147	10.099	10.424
85	16.557	18.346	10.629	10.996
90	17.391	19.308	11.181	11.574
95	18.250	20.401	11.638	12.113
100	18.987	21.466	12.176	12.612
105	19.856	22.406	12.647	13.141
110	20.573	23.612	13.097	13.602
115	21.320	24.418	13.627	14.142
120	22.154	25.545	13.982	14.623
125	22.739	26.451	14.518	15.062
130	23.460	27.617	14.937	15.614
135	24.200	28.468	15.438	16.131
140	24.825	29.304	15.931	16.558
145	25.702	30.456	16.312	16.994
150	26.327	31.228	16.672	17.384
155	26.799	32.347	17.188	17.749
160	27.638	33.296	17.567	18.228
165	28.446	34.002	17.991	18.753
170	28.991	34.927	18.440	19.174
175	29.636	35.947	18.812	19.636
180	30.229	37.024	19.252	20.055
185	30.863	37.767	19.544	20.474
190	31.484	38.634	19.959	20.864
195	32.102	39.664	20.396	21.328
200	32.745	40.572	20.763	21.843
205	33.344	41.423	21.178	22.000
210	33.968	42.387	21.513	22.506
215	34.440	43.485	21.931	22.936
220	35.152	43.873	22.168	23.259
225	35.628	45.085	22.667	23.684
230	36.439	45.815	22.986	24.051
235	36.818	46.840	23.316	24.479
240	37.493	47.722	23.764	24.799
245	38.156	48.476	23.968	25.177
250	38.551	49.329	24.450	25.537
255	39.146	50.440	24.763	25.872
260	39.708	51.195	25.237	26.362
265	40.364	51.930	25.442	26.564
270	40.872	52.692	25.830	26.897
275	41.125	53.607	26.149	27.267
280	42.201	54.647	26.490	27.691
285	42.423	55.537	26.852	28.076
290	43.103	55.928	27.090	28.497
295	43.587	57.027	27.455	28.661
300	44.194	57.992	27.741	29.159

C Famílias de Permutações

Esta seção apresenta algumas famílias de permutações para alguns dos problemas estudados bem como algoritmos que podem ordená-las. Cada família criada possui uma tabela que mostra seu formato para tamanhos pequenos e seu formato genérico, bem como suas distâncias (que para tamanhos menores do que 13 são exatas) e os valores do diâmetro, para comparação. O valor descrito na coluna da distância nas linhas correspondentes ao formato genérico da permutação indica quantas operações o algoritmo criado utiliza para ordená-la. Esse algoritmo é dado logo após tal tabela. Por fim, um lema relacionado à distância exata da família é apresentado.

As Tabelas 35, 53, 64 e 95 apresentam um resumo das famílias encontradas para o SbPRPT, SbPRSR, SbPTST e SbPRPTSRST, respectivamente. Para cada família, as tabelas apresentam o número de *breakpoints* e a distância ou os limitantes da distância encontrados.

C.1 Famílias para SbPRPT

Tabela 5: SbPRPT – Família 1

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(7 5 3 6 2 4 1)	5	5
8	(7 5 3 8 2 4 6 1)	5	5
9	(9 7 5 3 8 2 4 6 1)	6	6
10	(9 7 5 3 10 2 4 6 8 1)	6	7
11	(11 9 7 5 3 10 2 4 6 8 1)	7	7
12	(11 9 7 5 3 12 2 4 6 8 10 1)	7	8
13	(13 11 9 7 5 3 12 2 4 6 8 10 1)	8	9
14	(13 11 9 7 5 3 14 2 4 6 8 10 12 1)	?	?
⋮	⋮	⋮	⋮
$2k$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 3\ \ n\ \ 2\ \ 4\ \ 6\ \ \ldots\ \ n{-}2\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 3\ \ n{-}1\ \ 2\ \ 4\ \ 6\ \ \ldots\ \ n{-}3\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 28 SbPRPT – Pseudocódigo para ordenar permutações da família 1

```
SbPRPT_Familia_1(π ← π_n, n ≥ 7)
 1  if n mod 2 = 0 then
 2      π ← π · τ_p(π_4^{-1}, n) · τ_p(n/2, π_n^{-1})
 3      while π_1 ≠ 3 do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(n − 1, n + 1) · ρ_p(2)
 6  else
 7      π ← π · τ_p(2, n + 1) · τ_p(π_4^{-1}, π_1^{-1}) · τ_p(⌊n/2⌋, π_{n−1}^{-1})
 8      while π_1 ≠ 3 do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(n − 2, n) · ρ_p(2)
```

Lema 100. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 3\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 28. □

Tabela 6: SbPRPT – Família 2.

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(1 3 7 5 2 6 4)	5	5
8	(1 3 7 5 2 8 6 4)	5	5
9	(1 3 5 9 7 2 8 6 4)	6	6
10	(1 3 5 9 7 2 10 8 6 4)	6	7
11	(1 3 5 7 11 9 2 10 8 6 4)	7	7
12	(1 3 5 7 11 9 2 12 10 8 6 4)	7	8
13	(1 3 5 7 9 13 11 2 12 10 8 6 4)	8	9
14	(1 3 5 7 9 13 11 2 14 12 10 8 6 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 5 ... $n-5$ $n-1$ $n-3$ 2 n $n-2$ $n-4$ ... 4)	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k+1$	(1 3 5 ... $n-4$ n $n-2$ 2 $n-1$ $n-3$ $n-5$ ... 4)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 29 SbPRPT – Pseudocódigo para ordenar permutações da família 2

```
SbPRPT_Familia_2(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_{n-1}^{-1} + 1, π_n^{-1})
 3      if  n > 8  then
 4          π ← π · τ_p(π_{n-5}^{-1}, π_{n-6}^{-1})
 5      while  π_1 ≠ 5  do
 6          π ← π · τ_p(π_{π_1-2}^{-1}, π_{π_1-1}^{-1} + 1)
 7      π ← π · τ_p(π_n^{-1} + 1, n + 1) · τ_p(2, n − 1) · τ_p(3, n − 2) · ρ_p(2)
 8  else
 9      π ← π · τ_p(π_n^{-1} + 1, π_{n-1}^{-1}) · τ_p(π_n^{-1}, π_{n-1}^{-1} + 1) · ρ_p(n)
10      while  π_1 ≠ n − 3  do
11          π ← π · τ_p(2, π_{π_1-1}^{-1})
12      π ← π · ρ_p(π_2^{-1}) · ρ_p(2)
```

Lema 101. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 5\ \ldots\ n{-}5\ n{-}1\ n{-}3\ 2\ n\ n{-}2\ n{-}4\ \ldots\ 4) & \textit{se } n \textit{ é par} \\ (1\ 3\ 5\ \ldots\ n{-}4\ n\ n{-}2\ 2\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 29. □

Tabela 7: SbPRPT – Família 3

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(7 4 6 2 5 3 1)	5	5
8	(7 4 6 8 2 5 3 1)	5	5
9	(9 4 6 8 2 7 5 3 1)	6	6
10	(9 4 6 8 10 2 7 5 3 1)	6	7
11	(11 4 6 8 10 2 9 7 5 3 1)	7	7
12	(11 4 6 8 10 12 2 9 7 5 3 1)	7	8
13	(13 4 6 8 10 12 2 11 9 7 5 3 1)	8	9
14	(13 4 6 8 10 12 14 2 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 30 SbPRPT – Pseudocódigo para ordenar permutações da família 3

```
SbPRPT_Familia_3(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(2, π_n^{-1}) · τ_p(π_2^{-1} + 1, n)
 3      while  π_1 ≠ 3  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(n − 1, n + 1) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_{n−1}^{-1}, π_{n−1}^{-1} + 3) · τ_p(2, 4) · τ_p(π_n^{-1} + 1, n + 1)
 8      while  π_1 ≠ n − 3  do
 9          π ← π · τ_p(2, π_{π_1−1}^{-1})
10      π ← π · ρ_p(π_2^{-1}) · ρ_p(2)
```

Lema 102. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 30. □

Tabela 8: SbPRPT – Família 4

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(7 4 6 2 5 3 1)	5	5
8	(7 6 8 4 2 5 3 1)	5	5
9	(9 6 8 4 2 7 5 3 1)	6	6
10	(9 8 10 6 4 2 7 5 3 1)	6	7
11	(11 8 10 6 4 2 9 7 5 3 1)	7	7
12	(11 10 12 8 6 4 2 9 7 5 3 1)	7	8
13	(13 10 12 8 6 4 2 11 9 7 5 3 1)	8	9
14	(13 12 14 10 8 6 4 2 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n{-}2\ \ n\ \ n{-}4\ \ n{-}6\ \ n{-}8\ \ \ldots\ \ 2\ \ n{-}3\ \ n{-}5\ \ n{-}7\ \ \ldots\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ \ n{-}3\ \ n{-}1\ \ n{-}5\ \ n{-}7\ \ n{-}9\ \ \ldots\ \ 2\ \ n{-}2\ \ n{-}4\ \ n{-}6\ \ \ldots\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 31 SbPRPT – Pseudocódigo para ordenar permutações da família 4

```
SbPRPT_Familia_4(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      π ← π · ρ_p(2) · τ_p(5, π_{n-3}^{-1} + 1)
3  else
4      π ← π · τ_p(3, π_{n-4}^{-1}) · τ_p(2, π_n^{-1})
5  while  π_1 ≠ 2  do
6      π ← π · τ_p(2, π_{π_1 - 1}^{-1})
7  π ← π · τ_p(π_n^{-1} + 1, n + 1) · ρ_p(π_2^{-1}) · ρ_p(2)
```

Lema 103. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ \ n{-}2\ \ n\ \ n{-}4\ \ n{-}6\ \ n{-}8\ \ \ldots\ \ 2\ \ n{-}3\ \ n{-}5\ \ n{-}7\ \ \ldots\ \ 1) & \textit{se } n \textit{ é par} \\ (n\ \ n{-}3\ \ n{-}1\ \ n{-}5\ \ n{-}7\ \ n{-}9\ \ \ldots\ \ 2\ \ n{-}2\ \ n{-}4\ \ n{-}6\ \ \ldots\ \ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n-1$ quando n é par, $b_{\rho_p}(\pi_n) = n$ quando n é ímpar e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Note que $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$ quando n é par. O limitante superior é dado pelo Algoritmo 31. □

Tabela 9: SbPRPT – Família 5

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
12	(1 3 7 5 2 6 4 8 10 12 9 11)	8	8
13	(1 3 7 5 2 6 4 8 10 12 9 13 11)	9	9
14	(1 3 7 5 2 6 4 8 10 12 14 9 13 11)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 7 5 2 6 4 8 10 12 ... n 9 $n-1$ $n-3$ $n-5$ ... 11)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k+1$	(1 3 7 5 2 6 4 8 10 12 ... $n-1$ 9 n $n-2$ $n-4$ ... 11)	$\leq \lceil \frac{n}{2} \rceil + 2$	?

Algoritmo 32 SbPRPT – Pseudocódigo para ordenar permutações da família 5

```
SbPRPT_Familia_5(π ← π_n, n ≥ 12)
1  π ← π · τ_p(π_n^{-1} + 1, n + 1)
2  while π_1 ≠ 1 do
3        π ← π · τ_p(2, π_{π_1+1}^{-1})
4  if n mod 2 = 1 then
5        π ← π · τ_p(n − 1, n) · τ_p(2, π_{10}^{-1})
6  π ← π · τ_p(4, 6) · τ_p(5, 7) · ρ_p(7) · ρ_p(4) · ρ_p(2)
```

Lema 104. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 7\ 5\ 2\ 6\ 4\ 8\ 10\ 12\ \ldots\ n\ 9\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 11) & \textit{se } n \textit{ é par} \\ (1\ 3\ 7\ 5\ 2\ 6\ 4\ 8\ 10\ 12\ \ldots\ n{-}1\ 9\ n\ n{-}2\ n{-}4\ \ldots\ 11) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 32. □

Tabela 10: SbPRPT – Família 6

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(1 5 2 7 4 6 3)	5	5
8	(1 5 2 7 6 4 8 3)	5	5
9	(1 5 7 2 9 6 4 8 3)	6	6
10	(1 5 7 2 9 8 6 4 10 3)	6	7
11	(1 5 7 9 2 11 8 6 4 10 3)	7	7
12	(1 5 7 9 2 11 10 8 6 4 12 3)	7	8
13	(1 5 7 9 11 2 13 10 8 6 4 12 3)	8	9
14	(1 5 7 9 11 2 13 12 10 8 6 4 14 3)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 5 7 9 ... $n-3$ 2 $n-1$ $n-2$ $n-4$ $n-6$ ... 4 n 3)	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k+1$	(1 5 7 9 ... $n-2$ 2 n $n-3$ $n-5$ $n-7$ ... 4 $n-1$ 3)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 33 SbPRPT – Pseudocódigo para ordenar permutações da família 6

```
SbPRPT_Familia_6(π ← π_n, n)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(n, n+1) · ρ_p(n−2) · τ_p(π_2^{-1} + 1, π_1^{-1})
 3      while  π_1 ≠ 5  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(π_2^{-1}, n) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_n^{-1} + 1, n+1) · τ_p(π_5^{-1}, n−1)
 8      while  π_1 ≠ n−2  do
 9          π ← π · τ_p(2, π_{π_1−1}^{-1})
10      π ← π · τ_p(π_{n−1}^{-1}, π_3^{-1}) · ρ_p(n−1) · ρ_p(2)
```

Lema 105. *Seja*

$$\pi_n = \begin{cases} (1\ 5\ 7\ 9\ \ldots\ n{-}3\ 2\ n{-}1\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 4\ n\ 3) & \textit{se } n \textit{ é par} \\ (1\ 5\ 7\ 9\ \ldots\ n{-}2\ 2\ n\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 4\ n{-}1\ 3) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n-1$ quando n é par, $b_{\rho_p}(\pi_n) = n$ quando n é ímpar, e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Note que $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$ quando n é par. O limitante superior é dado pelo Algoritmo 33. □

Tabela 11: SbPRPT – Família 7

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(5 2 7 4 1 6 3)	5	5
8	(5 2 7 4 1 8 6 3)	5	5
9	(5 7 2 9 4 1 8 6 3)	6	6
10	(5 7 2 9 4 1 10 8 6 3)	6	7
11	(5 7 9 2 11 4 1 10 8 6 3)	7	7
12	(5 7 9 2 11 4 1 12 10 8 6 3)	7	8
13	(5 7 9 11 2 13 4 1 12 10 8 6 3)	8	9
14	(5 7 9 11 2 13 4 1 14 12 10 8 6 3)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(5 7 9 ... $n{-}3$ 2 $n{-}1$ 4 1 n $n{-}2$ $n{-}4$ ... 6 3)	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	(5 7 9 ... $n{-}2$ 2 n 4 1 $n{-}1$ $n{-}3$ $n{-}5$ ... 6 3)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 34 SbPRPT – Pseudocódigo para ordenar permutações da família 7

```
SbPRPT_Familia_7(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_2^{-1} + 1, n) · τ_p(2, π_{π_1 − 1}^{-1})
 3      while  π_1 ≠ n − 4  do
 4          π ← π · τ_p(π_{π_1+2}^{-1}, π_{π_1+1}^{-1} + 1)
 5      π ← π · τ_p(π_n^{-1}, n − 1) · ρ_p(n) · ρ_p(3)
 6  else
 7      π ← π · τ_p(π_1^{-1}, n) · τ_p(n − 1, n + 1) · τ_p(π_6^{-1}, π_5^{-1} + 1)
 8      while  π_1 ≠ n − 1  do
 9          π ← π · τ_p(π_{π_1+2}^{-1}, π_{π_1+1}^{-1} + 1)
10      π ← π · ρ_p(n − 1) · ρ_p(2)
```

Lema 106. *Seja*

$$\pi_n = \begin{cases} (5\ 7\ 9\ \ldots\ n{-}3\ 2\ n{-}1\ 4\ 1\ n\ n{-}2\ n{-}4\ \ldots\ 6\ 3) & \textit{se } n \textit{ é par} \\ (5\ 7\ 9\ \ldots\ n{-}2\ 2\ n\ 4\ 1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 6\ 3) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 34. □

Tabela 12: SbPRPT – Família 8

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
9	(5 2 6 1 3 7 4 9 8)	6	6
10	(5 2 6 8 1 3 7 4 10 9)	7	7
11	(5 2 6 8 1 3 9 7 4 11 10)	7	7
12	(5 2 6 8 10 1 3 9 7 4 12 11)	8	8
13	(5 2 6 8 10 1 3 11 9 7 4 13 12)	8	9
14	(5 2 6 8 10 12 1 3 11 9 7 4 14 13)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(5 2 6 8 10 ... $n{-}2$ 1 3 $n{-}3$ $n{-}5$ $n{-}7$ ... 7 4 n $n{-}1$)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	(5 2 6 8 10 ... $n{-}3$ 1 3 $n{-}2$ $n{-}4$ $n{-}6$ ... 7 4 n $n{-}1$)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 35 SbPRPT – Pseudocódigo para ordenar permutações da família 8

```
SbPRPT_Familia_8(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(3, π_1^{-1}) · τ_p(π_3^{-1} + 1, π_4^{-1}) · τ_p(π_8^{-1}, π_5^{-1})
 3      while  π_1 ≠ n − 2  do
 4          π ← π · τ_p(2, π_{π_1 − 1}^{-1})
 5      π ← π · τ_p(n, n + 1) · ρ_p(n − 1) · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · ρ_p(n) · τ_p(π_7^{-1}, π_3^{-1}) · τ_p(π_1^{-1} + 1, π_2^{-1})
 8      while  π_1 ≠ 6  do
 9          π ← π · τ_p(2, π_{π_1 + 1}^{-1})
10      π ← π · τ_p(π_4^{-1}, n + 1) · ρ_p(4) · ρ_p(2)
```

Lema 107. *Seja*

$$\pi_n = \begin{cases} (5\ 2\ 6\ 8\ 10\ \ldots\ n{-}2\ 1\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 7\ 4\ n\ n{-}1) & \textit{se } n \textit{ é par} \\ (5\ 2\ 6\ 8\ 10\ \ldots\ n{-}3\ 1\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 7\ 4\ n\ n{-}1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 35. □

Tabela 13: SbPRPT – Família 9

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
11	(2 8 1 6 3 10 5 7 9 11 4)	7	7
12	(2 10 1 6 8 3 12 5 7 9 11 4)	8	8
13	(2 10 1 6 8 3 12 5 7 9 11 13 4)	8	9
14	(2 12 1 6 8 10 3 14 5 7 9 11 13 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 n−2 1 6 8 10 ... n−4 3 n 5 7 9 ... n−1 4)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k$+1	(2 n−3 1 6 8 10 ... n−5 3 n−1 5 7 9 ... n 4)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 36 SbPRPT – Pseudocódigo para ordenar permutações da família 9

```
SbPRPT_Familia_9(π ← π_n, n ≥ 11)
 1  if  n mod 2 = 0  then
 2      π ← π · ρ_p(π_n^{-1}) · τ_p(π_{n-2}^{-1}, π_4^{-1}) · τ_p(2, π_{π_1+1}^{-1}) · τ_p(π_{n-3}^{-1}, π_3^{-1} + 1)
 3      while  π_1 ≠ 5  do
 4          π ← π · τ_p(π_{π_1-2}^{-1}, π_{π_1-1}^{-1} + 1)
 5      π ← π · τ_p(π_3^{-1}, n + 1) · ρ_p(3)
 6  else
 7      π ← π · τ_p(5, π_7^{-1})
 8      while  π_1 ≠ 3  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(n, n + 1) · τ_p(3, 5) · τ_p(2, n) · τ_p(π_1^{-1}, n − 2) · τ_p(π_5^{-1}, π_2^{-1} + 1) · ρ_p(5)
```

Lema 108. *Seja*

$$\pi_n = \begin{cases} (2\ n{-}2\ 1\ 6\ 8\ 10\ \ldots\ n{-}4\ 3\ n\ 5\ 7\ 9\ \ldots\ n{-}1\ 4) & \textit{se } n \textit{ é par} \\ (2\ n{-}3\ 1\ 6\ 8\ 10\ \ldots\ n{-}5\ 3\ n{-}1\ 5\ 7\ 9\ \ldots\ n\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 36. □

Tabela 14: SbPRPT – Família 10

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
9	(6 2 7 9 4 8 5 1 3)	6	6
10	(10 6 2 7 9 4 8 5 1 3)	7	7
11	(10 6 2 7 9 11 4 8 5 1 3)	7	7
12	(12 10 6 2 7 9 11 4 8 5 1 3)	8	8
13	(12 10 6 2 7 9 11 13 4 8 5 1 3)	8	9
14	(14 12 10 6 2 7 9 11 13 4 8 5 1 3)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ n{-}2\ n{-}4\ \ldots\ 10\ 6\ 2\ 7\ 9\ \ldots\ n{-}1\ 4\ 8\ 5\ 1\ 3)$	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 6\ 2\ 7\ 9\ \ldots\ n\ 4\ 8\ 5\ 1\ 3)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 37 SbPRPT – Pseudocódigo para ordenar permutações da família 10

```
SbPRPT_Familia_10(π ← π_n, n ≥ 9)
1  if n mod 2 = 0 then
2      π ← π · τ_p(π_9^{-1}, π_4^{-1}) · τ_p(π_2^{-1}, π_5^{-1}) · τ_p(π_8^{-1}, n) · τ_p(π_1^{-1}, π_4^{-1}) · τ_p(π_{n+1}^{-1}, n + 1)
3      while π_1 ≠ 6 do
4          π ← π · τ_p(2, π_{π_1+1}^{-1})
5      π ← π · ρ_p(6) · ρ_p(2)
6  else
7      while π_1 ≠ 6 do
8          π ← π · τ_p(2, π_{π_1+1}^{-1})
9      π ← π · τ_p(π_8^{-1}, π_5^{-1} + 1) · τ_p(2, π_9^{-1}) · τ_p(π_4^{-1}, n + 1) · τ_p(2, π_5^{-1}) · τ_p(π_2^{-1}, π_2^{-1} + 1) · ρ_p(2)
```

Lema 109. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 10\ 6\ 2\ 7\ 9\ 11\ \ldots\ n{-}1\ 4\ 8\ 5\ 1\ 3) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 6\ 2\ 7\ 9\ 11\ \ldots\ n\ 4\ 8\ 5\ 1\ 3) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 37. □

Tabela 15: SbPRPT – Família 11

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
9	(8 4 9 5 7 2 6 3 1)	6	6
10	(8 10 4 9 5 7 2 6 3 1)	7	7
11	(8 10 4 11 9 5 7 2 6 3 1)	7	7
12	(8 10 12 4 11 9 5 7 2 6 3 1)	8	8
13	(8 10 12 4 13 11 9 5 7 2 6 3 1)	8	9
14	(8 10 12 14 4 13 11 9 5 7 2 6 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(8 10 12 ... n 4 $n-1$ $n-3$ $n-5$ ... 9 5 7 2 6 3 1)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k+1$	(8 10 12 ... $n-1$ 4 n $n-2$ $n-4$ ... 9 5 7 2 6 3 1)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 38 SbPRPT – Pseudocódigo para ordenar permutações da família 11

```
SbPRPT_Familia_11(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_7^{-1}, π_7^{-1} + 1) · τ_p(π_4^{-1} + 1, π_3^{-1})
 3      while  π_1 ≠ 5  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(2) · τ_p(π_4^{-1}, n + 1) · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_n^{-1}, π_3^{-1}) · τ_p(π_7^{-1}, π_4^{-1}) · τ_p(2, 4) · τ_p(π_n^{-1} + 1, n + 1)
 8      while  π_1 ≠ 5  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · ρ_p(5) · ρ_p(2)
```

Lema 110. *Seja*

$$\pi_n = \begin{cases} (8\ 10\ 12\ \ldots\ n\ 4\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 9\ 5\ 7\ 2\ 6\ 3\ 1) & \textit{se } n \textit{ é par} \\ (8\ 10\ 12\ \ldots\ n{-}1\ 4\ n\ n{-}2\ n{-}4\ \ldots\ 9\ 5\ 7\ 2\ 6\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 38. □

Tabela 16: SbPRPT – Família 12

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
9	(9 5 8 4 6 2 7 3 1)	6	6
10	(9 5 10 4 6 8 2 7 3 1)	7	7
11	(11 5 10 4 6 8 2 7 9 3 1)	7	7
12	(11 5 12 4 6 8 10 2 7 9 3 1)	8	8
13	(13 5 12 4 6 8 10 2 7 9 11 3 1)	8	9
14	(13 5 14 4 6 8 10 12 2 7 9 11 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	$(n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 39 SbPRPT – Pseudocódigo para ordenar permutações da família 12

```
SbPRPT_Familia_12(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(3, π_2^{-1}) · τ_p(4, π_5^{-1})
 3      while  π_1 ≠ n − 2  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(π_6^{-1}, n − 1) · ρ_p(3) · τ_p(n − 2, n + 1) · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · τ_p(3, 4) · τ_p(5, n − 1)
 8      while  π_1 ≠ 2  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(n − 3, n + 1) · ρ_p(5) · ρ_p(2)
```

Lema 111. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1) & \textit{se } n \textit{ é par} \\ (n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 39. □

Tabela 17: SbPRPT – Família 13

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
6	(2 1 4 3 6 5)	3	3
7	(2 1 4 3 6 5 7)	3	5
8	(2 1 4 3 6 5 8 7)	4	5
9	(2 1 4 3 6 5 8 7 9)	4	6
10	(2 1 4 3 6 5 8 7 10 9)	5	7
11	(2 1 4 3 6 5 8 7 10 9 11)	5	7
12	(2 1 4 3 6 5 8 7 10 9 12 11)	6	8
13	(2 1 4 3 6 5 8 7 10 9 12 11 13)	6	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 1 4 3 6 5 ... n $n-1$)	$\leq \lceil \frac{n-1}{2} \rceil$	?
$2k+1$	(2 1 4 3 6 5 ... $n-1$ $n-2$ n)	$\leq \lceil \frac{n-1}{2} \rceil$	?

Algoritmo 40 SbPRPT – Pseudocódigo para ordenar permutações da família 13

SbPRPT_Familia_13($\pi \leftarrow \pi_n$, $\mathtt{n} \geq 6$)

```
1  last_even ← n − 1
2  if  n mod 2 = 0  then
3        last_even ← n
4  π ← π · τ_p(3, π^{-1}_{last_even})
5  while  π_1 ≠ last_even − 2  do
6        π ← π · τ_p(3, π^{-1}_{π_2 − 1})
7  π ← π · τ_p(π^{-1}_{π_1+1}, π^{-1}_{π_1+1} + 1) · ρ_p(π_1^{-1})
```

Lema 112. *Seja*

$$\pi_n = \begin{cases} (2\ 1\ 4\ 3\ 6\ 5\ \ldots\ n\ n{-}1) & \text{se } n \text{ é par} \\ (2\ 1\ 4\ 3\ 6\ 5\ \ldots\ n{-}1\ n{-}2\ n) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-1}{4} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n-1}{2} \rceil$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = \lceil \frac{n-1}{2} \rceil$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 40. □

Tabela 18: SbPRPT – Família 14

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
10	(8 2 7 4 10 6 1 3 9 5)	7	7
11	(8 2 7 4 10 6 1 3 9 11 5)	7	7
12	(10 8 2 7 4 12 6 1 3 9 11 5)	8	8
13	(10 8 2 7 4 12 6 1 3 9 11 13 5)	8	9
14	(12 10 8 2 7 4 14 6 1 3 9 11 13 5)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}2\ n{-}4\ n{-}6\ \ldots\ 8\ 2\ 7\ 4\ n\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n{-}1\ 5)$	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	$(n{-}3\ n{-}5\ n{-}7\ \ldots\ 8\ 2\ 7\ 4\ n{-}1\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n\ 5)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 41 SbPRPT – Pseudocódigo para ordenar permutações da família 14

```
SbPRPT_Familia_14(π ← π_n, n ≥ 10)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_1^{-1}, n) · τ_p(π_2^{-1}, π_2^{-1} + 1) · τ_p(n − 1, n + 1) · τ_p(6, n − 1)
 3      while  π_1 ≠ n − 1  do
 4          π ← π · τ_p(2, π_{π_1 − 1}^{-1})
 5      π ← π · ρ_p(n − 1) · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_6^{-1}, n − 1) · τ_p(4, π_4^{-1})
 8      while  π_1 ≠ n − 2  do
 9          π ← π · τ_p(2, π_{π_1 − 1}^{-1})
10      π ← π · ρ_p(n − 3) · τ_p(n, n + 1) · ρ_p(4) · τ_p(π_2^{-1}, π_2^{-1} + 1) · ρ_p(2)
```

Lema 113. *Seja*

$$\pi_n = \begin{cases} (n{-}2\ n{-}4\ n{-}6\ \ldots\ 8\ 2\ 7\ 4\ n\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n{-}1\ 5) & \textit{se } n \textit{ é par} \\ (n{-}3\ n{-}5\ n{-}7\ \ldots\ 8\ 2\ 7\ 4\ n{-}1\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n\ 5) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 41. □

Tabela 19: SbPRPT – Família 15

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
10	(1 3 10 5 8 6 2 7 9 4)	7	7
11	(1 3 10 5 8 6 2 7 11 9 4)	7	7
12	(1 3 12 10 5 8 6 2 7 11 9 4)	8	8
13	(1 3 12 10 5 8 6 2 7 13 11 9 4)	8	9
14	(1 3 14 12 10 5 8 6 2 7 13 11 9 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 n $n-2$ $n-4$ ... 10 5 8 6 2 7 $n-1$ $n-3$ $n-5$ ... 9 4)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k+1$	(1 3 $n-1$ $n-3$ $n-5$ ... 10 5 8 6 2 7 n $n-2$ $n-4$ ... 9 4)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 42 SbPRPT – Pseudocódigo para ordenar permutações da família 15

```
SbPRPT_Familia_15(π ← π_n, n ≥ 10)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_6^{-1}, π_2^{-1} + 1) · τ_p(π_{n-2}^{-1}, π_7^{-1} + 1)
 3      while  π_1 ≠ 5  do
 4          π ← π · τ_p(2, π_{π_1 - 1}^{-1})
 5      π ← π · τ_p(2, 5) · τ_p(π_n^{-1}, n) · ρ_p(n) · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_6^{-1}, π_2^{-1} + 1) · τ_p(π_{n-1}^{-1}, π_7^{-1} + 1)
 8      while  π_1 ≠ 5  do
 9          π ← π · τ_p(2, π_{π_1 - 1}^{-1})
10      π ← π · τ_p(π_n^{-1} + 1, n + 1)ρ_p(n − 2)τ_p(π_5^{-1}, π_4^{-1} + 1) · ρ_p(5) · ρ_p(2)
```

Lema 114. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ n\ n{-}2\ n{-}4\ \ldots\ 10\ 5\ 8\ 6\ 2\ 7\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 9\ 4) & \text{se } n \text{ é par} \\ (1\ 3\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 5\ 8\ 6\ 2\ 7\ n\ n{-}2\ n{-}4\ \ldots\ 9\ 4) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 42. □

Tabela 20: SbPRPT – Família 16

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
10	(10 5 7 9 3 8 4 2 6 1)	7	7
11	(10 5 7 9 11 3 8 4 2 6 1)	7	7
12	(12 5 7 9 11 3 8 10 4 2 6 1)	8	8
13	(12 5 7 9 11 13 3 8 10 4 2 6 1)	8	9
14	(14 5 7 9 11 13 3 8 10 12 4 2 6 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ 5\ 7\ 9\ \ldots\ n{-}1\ 3\ 8\ 10\ 12\ \ldots\ n{-}2\ 4\ 2\ 6\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	$(n{-}1\ 5\ 7\ 9\ \ldots\ n\ 3\ 8\ 10\ 12\ \ldots\ n{-}3\ 4\ 2\ 6\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 43 SbPRPT – Pseudocódigo para ordenar permutações da família 16

```
SbPRPT_Familia_16(π ← π_n, n ≥ 10)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(4, π_3^{-1})
 3      while  π_1 ≠ n − 1  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(4, π_4^{-1}) · τ_p(3, n) · τ_p(π_2^{-1}, n − 1) · τ_p(n − 3, n + 1) · ρ_p(5) · ρ_p(2)
 6  else
 7      π ← π · τ_p(2, π_n^{-1}) · τ_p(π_8^{-1}, π_2^{-1})
 8      while  π_1 ≠ 4  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(n − 3, n + 1) · τ_p(4, 7) · τ_p(4, 6) · ρ_p(3)
```

Lema 115. *Seja*

$$\pi_n = \begin{cases} (n\ 5\ 7\ 9\ \ldots\ n{-}1\ 3\ 8\ 10\ 12\ \ldots\ n{-}2\ 4\ 2\ 6\ 1) & \textit{se } n \textit{ é par} \\ (n{-}1\ 5\ 7\ 9\ \ldots\ n\ 3\ 8\ 10\ 12\ \ldots\ n{-}3\ 4\ 2\ 6\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 43. □

Tabela 21: SbPRPT – Família 17

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
4	(4 1 3 2)	2	2
5	(5 1 4 2 3)	2	3
6	(6 1 5 2 4 3)	3	4
7	(7 1 6 2 5 3 4)	3	5
8	(8 1 7 2 6 3 5 4)	4	5
9	(9 1 8 2 7 3 6 4 5)	4	6
10	(10 1 9 2 8 3 7 4 6 5)	5	7
11	(11 1 10 2 9 3 8 4 7 5 6)	5	7
12	(12 1 11 2 10 3 9 4 8 5 7 6)	6	8
13	(13 1 12 2 11 3 10 4 9 5 8 6 7)	6	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2})$	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil \frac{n}{2} \rceil{+}1\ \lceil \frac{n}{2} \rceil{-}1\ \lceil \frac{n}{2} \rceil)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 44 SbPRPT – Pseudocódigo para ordenar permutações da família 17

```
SbPRPT_Familia_17(π ← π_n, n ≥ 4)
1  if  n mod 2 = 0  then
2      π ← π · ρ_p(n)
3      for  i ← n/2 − 1 downto 1  do
4          π ← π · τ_p(π_i^{-1}, π_i^{-1} + 1)
5  else
6      for  i ← n − 1 downto ⌈n/2⌉ + 1  do
7          π ← π · τ_p(π_i^{-1}, π_i^{-1} + 1)
8      π ← π · τ_p(π_1^{-1}, n + 1)
```

Lema 116. *Seja*

$$\pi_n = \begin{cases} (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2}) & \text{se } n \text{ é par} \\ (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil \frac{n}{2} \rceil{+}1\ \lceil \frac{n}{2} \rceil{-}1\ \lceil \frac{n}{2} \rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p}(\pi_n) = \lceil \frac{n-1}{2} \rceil$.

Demonstração. Note que $d_{\rho_p\tau_p}(\pi_n) = \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Além disso, o Algoritmo 44 ordena π_n com $\lceil \frac{n-1}{2} \rceil$ operações. □

Tabela 22: SbPRPT – Família 18

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
5	(4 5 2 3 1)	2	3
6	(5 6 3 4 1 2)	2	4
7	(6 7 4 5 2 3 1)	3	5
8	(7 8 5 6 3 4 1 2)	3	5
9	(8 9 6 7 4 5 2 3 1)	4	6
10	(9 10 7 8 5 6 3 4 1 2)	4	7
11	(10 11 8 9 6 7 4 5 2 3 1)	5	7
12	(11 12 9 10 7 8 5 6 3 4 1 2)	5	8
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ n\ n{-}3\ n{-}2\ n{-}5\ n{-}4\ \ldots\ 3\ 4\ 1\ 2)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?
$2k{+}1$	$(n{-}1\ n\ n{-}3\ n{-}2\ n{-}5\ n{-}4\ \ldots\ 4\ 5\ 2\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 45 SbPRPT – Pseudocódigo para ordenar permutações da família 18

```
SbPRPT_Familia_18(π ← π_n, n ≥ 5)
1  π ← π · τ_p(3, n + 1)
2  while  π_1 ≠ 1  do
3        π ← π · τ_p(3, π^{-1}_{π_2+1})
```

Lema 117. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n\ n{-}3\ n{-}2\ n{-}5\ n{-}4\ \ldots\ 3\ 4\ 1\ 2) & \textit{se } n \textit{ é par} \\ (n{-}1\ n\ n{-}3\ n{-}2\ n{-}5\ n{-}4\ \ldots 4\ 5\ 2\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{4} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil - 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = \lceil \frac{n}{2} \rceil$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 45. □

Tabela 23: SbPRPT – Família 19

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
5	(1 5 2 4 3)	3	3
6	(1 6 2 5 3 4)	3	4
7	(1 7 2 6 3 5 4)	4	5
8	(1 8 2 7 3 6 4 5)	4	5
9	(1 9 2 8 3 7 4 6 5)	5	6
10	(1 10 2 9 3 8 4 7 5 6)	5	7
11	(1 11 2 10 3 9 4 8 5 7 6)	6	7
12	(1 12 2 11 3 10 4 9 5 8 6 7)	6	8
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1)$	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	$(1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil \frac{n}{2} \rceil{-}1\ \lceil \frac{n}{2} \rceil{+}1\ \lceil \frac{n}{2} \rceil)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 46 SbPRPT – Pseudocódigo para ordenar permutações da família 19

```
SbPRPT_Familia_19(π ← π_n, n ≥ 5)
1  if  n mod 2 = 0  then
2      for  i ← n − 1 downto n/2 + 2  do
3          π ← π · τ_p(π_i^{-1}, π_i^{-1} + 1)
4      π ← π · τ_p(π_2^{-1}, n + 1) · τ_p(n − 1, n)
5  else
6      π ← π · τ_p(3, n + 1)
7      for  i ← 3 to ⌈n/2⌉ − 1  do
8          π ← π · τ_p(π_i^{-1}, π_i^{-1} + 1)
9      π ← π · τ_p(π_{n-1}^{-1}, n − 1) · ρ_p(n − 1)
```

Lema 118. *Seja*

$$\pi_n = \begin{cases} (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1) & \text{se } n \text{ é par} \\ (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil \frac{n}{2} \rceil{-}1\ \lceil \frac{n}{2} \rceil{+}1\ \lceil \frac{n}{2} \rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 46. □

Tabela 24: SbPRPT – Família 20

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
5	(2 3 5 4 1)	2	3
6	(2 3 5 6 4 1)	2	4
7	(2 3 5 7 6 4 1)	3	5
8	(2 3 5 7 8 6 4 1)	3	5
9	(2 3 5 7 9 8 6 4 1)	4	6
10	(2 3 5 7 9 10 8 6 4 1)	4	7
11	(2 3 5 7 9 11 10 8 6 4 1)	5	7
12	(2 3 5 7 9 11 12 10 8 6 4 1)	5	8
13	(2 3 5 7 9 11 13 12 10 8 6 4 1)	6	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 3 5 7 ... $n{-}1$ n $n{-}2$ $n{-}4$ ... 4 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?
$2k{+}1$	(2 3 5 7 ... n $n{-}1$ $n{-}3$ $n{-}5$ ... 4 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 47 SbPRPT – Pseudocódigo para ordenar permutações da família 20

```
SbPRPT_Familia_20(π ← π_n, n ≥ 5)
1  π ← π · τ_p(π_n^{-1} + 1, n + 1)
2  while  π_1 ≠ 1  do
3        π ← π · τ_p(2, π_{π_1+1}^{-1})
```

Lema 119. *Seja*

$$\pi_n = \begin{cases} (2\ 3\ 5\ 7\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ 4\ 1) & \textit{se } n \textit{ é par} \\ (2\ 3\ 5\ 7\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$.

Demonstração. Note que $d_{\rho_p\tau_p}(\pi) \geq \lceil \frac{n}{2} \rceil - 1$ pois $b_{\rho_p}(\pi_n) = n - 2$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Além disso, o Algoritmo 47 ordena π_n com $\lceil \frac{n}{2} \rceil - 1$ operações. □

Tabela 25: SbPRPT – Família 21

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
8	(7 5 1 2 3 8 6 4)	4	5
9	(9 7 5 1 2 3 8 6 4)	5	6
10	(9 7 5 1 2 3 10 8 6 4)	5	7
11	(11 9 7 5 1 2 3 10 8 6 4)	6	7
12	(11 9 7 5 1 2 3 12 10 8 6 4)	6	8
13	(13 11 9 7 5 1 2 3 12 10 8 6 4)	7	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 5\ 1\ 2\ 3\ n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 4)$	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 5\ 1\ 2\ 3\ n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 4)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 48 SbPRPT – Pseudocódigo para ordenar permutações da família 21

```
SbPRPT_Familia_21(π ← π_n, n ≤ 8)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ 5  do
3          π ← π · τ_p(2, π^{-1}_{π_1 − 1})
4      π ← π · τ_p(5, n) · ρ_p(n) · ρ_p(4)
5  else
6      π ← π · τ_p(3, π^{-1}_{n−1} + 1)
7      while  π_1 ≠ 1  do
8          π ← π · τ_p(2, π^{-1}_{π_1 − 1})
9      π ← π · τ_p(π_n^{-1} + 1, n + 1) · ρ_p(n − 2) · ρ_p(3)
```

Lema 120. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 5\ 1\ 2\ 3\ n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 4) & \textit{se } n \textit{ é par} \\ (n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 5\ 1\ 2\ 3\ n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n - 2$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 48. □

Tabela 26: SbPRPT – Família 22

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
7	(4 6 1 2 7 3 5)	3	5
8	(4 6 8 1 2 7 3 5)	4	5
9	(4 6 8 1 2 9 3 5 7)	4	6
10	(4 6 8 10 1 2 9 3 5 7)	5	7
11	(4 6 8 10 1 2 11 3 5 7 9)	5	7
12	(4 6 8 10 12 1 2 11 3 5 7 9)	6	8
13	(4 6 8 10 12 1 2 13 3 5 7 9 11)	6	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(4 6 8 ... n 1 2 $n-1$ 3 5 7 ... $n-3$)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k+1$	(4 6 8 ... $n-1$ 1 2 n 3 5 7 ... $n-2$)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 49 SbPRPT – Pseudocódigo para ordenar permutações da família 22

```
SbPRPT_Familia_22(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ n − 2  do
3          π ← π · τ_p(2, π^{-1}_{π_1+1})
4      π ← π · τ_p(3, n + 1)
5  else
6      while  π_1 ≠ n − 1  do
7          π ← π · τ_p(2, π^{-1}_{π_1+1})
8  π ← π · τ_p(π_3^{-1}, π^{-1}_{n-2} + 1) · τ_p(π_1^{-1}, π_2^{-1} + 1)
```

Lema 121. *Seja*

$$\pi_n = \begin{cases} (4\ 6\ 8\ \ldots\ n\ 1\ 2\ n{-}1\ 3\ 5\ 7\ \ldots\ n{-}3) & \textit{se } n \textit{ é par} \\ (4\ 6\ 8\ \ldots\ n{-}1\ 1\ 2\ n\ 3\ 5\ 7\ \ldots\ n{-}2) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p}(\pi_n) = \lceil \frac{n-1}{2} \rceil$.

Demonstração. Note que $d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Além disso, o Algoritmo 49 ordena π_n em $\lceil \frac{n-1}{2} \rceil$ passos. □

Tabela 27: SbPRPT – Família 23

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
12	(6 3 7 9 11 8 12 10 5 2 4 1)	8	8
13	(6 3 7 9 11 8 12 10 13 5 2 4 1)	9	9
14	(6 3 7 9 11 8 12 14 10 13 5 2 4 1)	?	?
15	(6 3 7 9 11 8 12 14 10 15 13 5 2 4 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(6 3 7 9 11 8 12 14 16 ... n 10 $n-1$ $n-3$ $n-5$ ... 13 5 2 4 1)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k+1$	(6 3 7 9 11 8 12 14 16 ... $n-1$ 10 n $n-2$ $n-4$ ... 13 5 2 4 1)	$\leq \lceil \frac{n}{2} \rceil + 2$	?

Algoritmo 50 SbPRPT – Pseudocódigo para ordenar permutações da família 23

```
SbPRPT_Familia_23(π ← π_n, n ≥ 12)
1  π ← π · τ_p(π_10^{-1}, n + 1) · τ_p(2, π_11^{-1})
2  while π_1 ≠ 5 do
3        π ← π · τ_p(2, π_{π_1 − 1}^{-1} + 1)
4  π ← π · τ_p(π_8^{-1}, π_8^{-1} + 1) · τ_p(2, π_9^{-1}) · τ_p(3,4) · τ_p(6,7) · τ_p(4,6) · ρ_p(2)
```

Lema 122. *Seja*

$$\pi_n = \begin{cases} (6\ 3\ 7\ 9\ 11\ 8\ 12\ 14\ 16\ \ldots\ n\ 10\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 13\ 5\ 2\ 4\ 1) & \textit{se } n \textit{ é par} \\ (6\ 3\ 7\ 9\ 11\ 8\ 12\ 14\ 16\ \ldots\ n{-}1\ 10\ n\ n{-}2\ n{-}4\ \ldots\ 13\ 5\ 2\ 4\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 50. □

Tabela 28: SbPRPT – Família 24

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
12	(5 3 1 8 4 7 2 6 10 12 9 11)	8	8
13	(5 3 1 8 4 7 2 6 10 12 9 13 11)	9	9
14	(5 3 1 8 4 7 2 6 10 12 14 9 13 11)	?	?
15	(5 3 1 8 4 7 2 6 10 12 14 9 15 13 11)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(5 3 1 8 4 7 2 6 10 12 14 ... n 9 $n-1$ $n-3$ $n-5$ ... 11)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k+1$	(5 3 1 8 4 7 2 6 10 12 14 ... $n-1$ 9 n $n-2$ $n-4$ ... 11)	$\leq \lceil \frac{n}{2} \rceil + 2$	?

Algoritmo 51 SbPRPT – Pseudocódigo para ordenar permutações da família 24

```
SbPRPT_Familia_24(π ← π_n, n ≥ 12)
1  π ← π · τ_p(π_12^{-1}, π_11^{-1}) · τ_p(2, n+1) · τ_p(π_3^{-1}, π_4^{-1}) · τ_p(π_8^{-1}, π_2^{-1}) · τ_p(π_7^{-1}, π_7^{-1}+1) · τ_p(4, π_10^{-1})
2  while π_1 ≠ 5 do
3        π ← π · τ_p(2, π_{π_1-1}^{-1} + 1)
4  π ← π · ρ_p(5) · ρ_p(2)
```

Lema 123. *Seja*

$$\pi_n = \begin{cases} (5\ 3\ 1\ 8\ 4\ 7\ 2\ 6\ 10\ 12\ 14\ \ldots\ n\ 9\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 11) & \textit{se } n \textit{ é par} \\ (5\ 3\ 1\ 8\ 4\ 7\ 2\ 6\ 10\ 12\ 14\ \ldots\ n{-}1\ 9\ n\ n{-}2\ n{-}4\ \ldots\ 11) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 51. □

Tabela 29: SbPRPT – Família 25

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
9	(8 4 9 7 1 3 5 2 6)	6	6
10	(10 8 4 9 7 1 3 5 2 6)	6	7
11	(10 8 4 9 11 7 1 3 5 2 6)	7	7
12	(12 10 8 4 9 11 7 1 3 5 2 6)	7	8
13	(12 10 8 4 9 11 13 7 1 3 5 2 6)	8	9
14	(14 12 10 8 4 9 11 13 7 1 3 5 2 6)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 8\ 4\ 9\ 11\ 13\ \ \ldots\ \ n{-}1\ 7\ 1\ 3\ 5\ 2\ 6)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 8\ 4\ 9\ 11\ 13\ \ \ldots\ \ n\ 7\ 1\ 3\ 5\ 2\ 6)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 52 SbPRPT – Pseudocódigo para ordenar permutações da família 25

```
SbPRPT_Familia_25(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_4^{-1}, π_7^{-1}) · τ_p(π_5^{-1}, π_5^{-1} + 1) · τ_p(π_n^{-1} + 1, n + 1)
 3      while  π_1 ≠ 8  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(8) · τ_p(4, 7) · τ_p(3, 4)
 6  else
 7      π ← π · τ_p(π_4^{-1}, π_7^{-1}) · τ_p(2, π_5^{-1})
 8      while  π_1 ≠ n  do
 9          π ← π · τ_p(2, π_{π_1-1}^{-1})
10      π ← π · ρ_p(n) · ρ_p(6) · τ_p(5, 6) · ρ_p(2)
```

Lema 124. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 8\ 4\ 9\ 11\ 13\ \ldots\ n{-}1\ 7\ 1\ 3\ 5\ 2\ 6) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 8\ 4\ 9\ 11\ 13\ \ldots\ n\ 7\ 1\ 3\ 5\ 2\ 6) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 52. □

Tabela 30: SbPRPT – Família 26

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
12	(8 10 1 4 2 6 3 5 7 12 9 11)	8	8
13	(8 10 1 4 2 6 3 5 7 12 9 13 11)	9	9
14	(8 10 1 12 4 2 6 3 5 7 14 9 13 11)	?	?
15	(8 10 1 12 4 2 6 3 5 7 14 9 15 13 11)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(8 10 1 12 14 16 ... $n{-}2$ 4 2 6 3 5 7 n 9 $n{-}1$ $n{-}3$ $n{-}5$ ... 11)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	(8 10 1 12 14 16 ... $n{-}3$ 4 2 6 3 5 7 $n{-}1$ 9 n $n{-}2$ $n{-}4$ ... 11)	$\leq \lceil \frac{n}{2} \rceil + 2$	?

Algoritmo 53 SbPRPT – Pseudocódigo para ordenar permutações da família 26

```
SbPRPT_Familia_26(π ← π_n, n ≥ 12)
1  π ← π · τ_p(π_4^{-1}, π_7^{-1} + 1) · τ_p(2, 5) · τ_p(π_9^{-1}, n + 1) · τ_p(2, π_10^{-1})
2  while π_1 ≠ 11 do
3        π ← π · τ_p(2, π_{π_1 − 1}^{-1} + 1)
4  π ← π · τ_p(3, π_1^{-1}) · τ_p(2, 5) · τ_p(π_2^{-1}, π_1^{-1} + 1) · ρ_p(2)
```

Lema 125. *Seja*

$$\pi_n = \begin{cases} (8\ 10\ 1\ 12\ 14\ 16\ \ldots\ n{-}2\ 4\ 2\ 6\ 3\ 5\ 7\ n\ 9\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 11) & \textit{se } n \textit{ é par} \\ (8\ 10\ 1\ 12\ 14\ 16\ \ldots\ n{-}3\ 4\ 2\ 6\ 3\ 5\ 7\ n{-}1\ 9\ n\ n{-}2\ n{-}4\ \ldots\ 11) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 53. □

Tabela 31: SbPRPT – Família 27

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
12	(11 9 12 10 2 4 6 3 7 5 8 1)	8	8
13	(11 13 9 12 10 2 4 6 3 7 5 8 1)	9	9
14	(11 13 9 14 12 10 2 4 6 3 7 5 8 1)	?	?
15	(11 13 15 9 14 12 10 2 4 6 3 7 5 8 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(11 13 15 ... $n-1$ 9 n $n-2$ $n-4$ ... 10 2 4 6 3 7 5 8 1)	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k+1$	(11 13 15 ... n 9 $n-1$ $n-3$ $n-5$ ... 10 2 4 6 3 7 5 8 1)	$\leq \lceil \frac{n}{2} \rceil + 2$	?

Algoritmo 54 SbPRPT – Pseudocódigo para ordenar permutações da família 27

```
SbPRPT_Familia_27(π ← π_n, n ≥ 12)
1  while  π_1 ≠ 9  do
2        π ← π · τ_p(2, π^{-1}_{π_1 − 1})
3  π ← π · τ_p(π_7^{-1}, n) · τ_p(π_6^{-1}, π_6^{-1} + 1) · τ_p(3, 4) · ρ_p(π_2^{-1}) · τ_p(π_9^{-1}, n + 1) · ρ_p(9) · ρ_p(2)
```

Lema 126. *Seja*

$$\pi_n = \begin{cases} (11\ 13\ 15\ \ldots\ n{-}1\ 9\ n\ n{-}2\ n{-}4\ \ldots\ 10\ 2\ 4\ 6\ 3\ 7\ 5\ 8\ 1) & \textit{se } n \textit{ é par} \\ (11\ 13\ 15\ \ldots\ n\ 9\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 2\ 4\ 6\ 3\ 7\ 5\ 8\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 54. □

Tabela 32: SbPRPT – Família 28

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
12	(12 6 8 10 3 7 4 9 5 1 11 2)	8	8
13	(13 6 8 10 12 3 7 4 9 5 1 11 2)	9	9
14	(14 6 8 10 12 3 7 4 9 11 5 1 13 2)	?	?
15	(15 6 8 10 12 14 3 7 4 9 11 5 1 13 2)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ 6\ 8\ 10\ \ldots\ n{-}2\ 3\ 7\ 4\ 9\ 11\ 13\ \ldots\ n{-}3\ 5\ 1\ n{-}1\ 2)$	$\leq \lceil \frac{n}{2} \rceil + 2$	?
$2k{+}1$	$(n\ 6\ 8\ 10\ \ldots\ n{-}1\ 3\ 7\ 4\ 9\ 11\ 13\ \ldots\ n{-}4\ 5\ 1\ n{-}2\ 2)$	$\leq \lceil \frac{n}{2} \rceil + 2$	?

Algoritmo 55 SbPRPT – Pseudocódigo para ordenar permutações da família 28

```
SbPRPT_Familia_28(π ← π_n, n ≥ 12)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_7^{-1}, π_2^{-1}) · τ_p(4, π_8^{-1} + 1)
 3      while  π_1 ≠ 5  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(2, π_4^{-1}) · τ_p(4, n + 1) · τ_p(2, 4) · ρ_p(5) · τ_p(π_3^{-1}, π_1^{-1} + 1) · ρ_p(3)
 6  else
 7      π ← π · τ_p(2, n + 1) · τ_p(π_9^{-1}, π_1^{-1})
 8      while  π_1 ≠ 5  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · ρ_p(π_7^{-1}) · τ_p(2, π_6^{-1}) · τ_p(3, n) · τ_p(π_{n-2}^{-1}, π_{n-2}^{-1} + 1) · ρ_p(n − 2) · ρ_p(3)
```

Lema 127. *Seja*

$$\pi_n = \begin{cases} (n\ 6\ 8\ 10\ \ldots\ n{-}2\ 3\ 7\ 4\ 9\ 11\ 13\ \ldots\ n{-}3\ 5\ 1\ n{-}1\ 2) & \textit{se } n \textit{ é par} \\ (n\ 6\ 8\ 10\ \ldots\ n{-}1\ 3\ 7\ 4\ 9\ 11\ 13\ \ldots\ n{-}4\ 5\ 1\ n{-}2\ 2) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 55. □

Tabela 33: SbPRPT – Família 29

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
4	(4 2 1 3)	2	2
5	(4 2 1 3 5)	2	3
6	(6 4 2 1 3 5)	3	4
7	(6 4 2 1 3 5 7)	3	5
8	(8 6 4 2 1 3 5 7)	4	5
9	(8 6 4 2 1 3 5 7 9)	4	6
10	(10 8 6 4 2 1 3 5 7 9)	5	7
11	(10 8 6 4 2 1 3 5 7 9 11)	5	7
12	(12 10 8 6 4 2 1 3 5 7 9 11)	6	8
13	(12 10 8 6 4 2 1 3 5 7 9 11 13)	6	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ n{-}2\ n{-}4\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n{-}1)$	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 56 SbPRPT – Pseudocódigo para ordenar permutações da família 29

```
SbPRPT_Familia_29(π ← π_n, n ≥ 4)
1  while  π_1 ≠ 1  do
2        π ← π · τ_p(2, π^{-1}_{π_1 − 1} + 1)
```

Lema 128. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n{-}1) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\rho_p\tau_p}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Quando n é par, note que $d_{\rho_p\tau_p}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil = \frac{n}{2}$ porque $b_{\rho_p}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. Pelo mesmo motivo, quando n é ímpar, $d_{\rho_p\tau_p}(\pi_n) \geq \lceil \frac{n-2}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$ pois $b_{\rho_p}(\pi_n) = n - 2$. Além disso, o Algoritmo 56 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil \frac{n}{2} \rceil - 1$ quando n é ímpar. □

Tabela 34: SbPRPT – Família 30

n	π_n	$d_{\rho_p\tau_p}(\pi_n)$	$D_{\rho_p\tau_p}(n)$
4	(2 4 3 1)	2	2
5	(2 4 5 3 1)	2	3
6	(2 4 6 5 3 1)	3	4
7	(2 4 6 7 5 3 1)	3	5
8	(2 4 6 8 7 5 3 1)	4	5
9	(2 4 6 8 9 7 5 3 1)	4	6
10	(2 4 6 8 10 9 7 5 3 1)	5	7
11	(2 4 6 8 10 11 9 7 5 3 1)	5	7
12	(2 4 6 8 10 12 11 9 7 5 3 1)	6	8
13	(2 4 6 8 10 12 13 11 9 7 5 3 1)	6	9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 4 6 ... n $n-1$ $n-3$ $n-5$ ... 1)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k+1$	(2 4 6 ... $n-1$ n $n-2$ $n-4$ ... 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 57 SbPRPT – Pseudocódigo para ordenar permutações da família 30

```
SbPRPT_Familia_30(π ← π_n, n ≥ 4)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ n  do
3          π ← π · τ_p(2, π^{-1}_{π_1 − 1})
4      π ← π · ρ_p(n)
5  else
6      π ← π · τ_p(π^{-1}_n + 1, n + 1)
7      while  π_1 ≠ 1  do
8          π ← π · τ_p(2, π^{-1}_{π_1 + 1})
```

Lema 129. *Seja*

$$\pi_n = \begin{cases} (2\ 4\ 6\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (2\ 4\ 6\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p}(\pi_n) = \lceil \frac{n-1}{2} \rceil$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p}(\pi) \geq \left\lceil \frac{b_{\rho_p}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 57. □

Tabela 35: SbPRPT – Resumo das famílias encontradas ($d = d_{\rho_p}(\pi_n)$)

Família	$b_{\rho_p}(\pi_n)$	d se n par	d se n ímpar
1	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
2	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
3	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
4	$n-1$ quando n é par e n quando n é ímpar	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
5	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
6	$n-1$ quando n é par e n quando n é ímpar	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
7	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
8	$n-1$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	$\lceil \frac{n}{2} \rceil - 1 \leq d \leq \lceil \frac{n}{2} \rceil + 1$
9	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$
10	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
11	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
12	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
13	$\lceil \frac{n-1}{2} \rceil$	$\lceil \frac{n-1}{4} \rceil \leq d \leq \lceil \frac{n-1}{2} \rceil$	
14	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$
15	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$
16	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$
17	$n-1$	$d = \lceil \frac{n-1}{2} \rceil$	
18	$\lceil \frac{n}{2} \rceil$	$\lceil \frac{n}{4} \rceil \leq d \leq \lceil \frac{n}{2} \rceil - 1$	
19	$n-1$	$\lceil \frac{n}{2} \rceil - 1 \leq d \leq \lceil \frac{n}{2} \rceil$	
20	$n-2$	$d = \lceil \frac{n}{2} \rceil - 1$	
21	$n-2$	$\lceil \frac{n}{2} \rceil - 1 \leq d \leq \lceil \frac{n}{2} \rceil$	
22	$n-2$	$d = \lceil \frac{n-1}{2} \rceil$	
23	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
24	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
25	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
26	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
27	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
28	n	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 2$	
29	$n-1$ quando n é par e $n-2$ quando n é ímpar	$d = \frac{n}{2}$	$d = \lceil \frac{n}{2} \rceil - 1$
30	$n-1$	$d = \lceil \frac{n-1}{2} \rceil$	

C.2 Famílias para SbPRSR

Tabela 36: SbPRSR – Família 1

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(7 5 3 6 2 4 1)	6	7
8	(7 5 3 8 2 4 6 1)	7	8
9	(9 7 5 3 8 2 4 6 1)	8	9
10	(9 7 5 3 10 2 4 6 8 1)	9	10
11	(11 9 7 5 3 10 2 4 6 8 1)	10	11
12	(11 9 7 5 3 12 2 4 6 8 10 1)	11	12
13	(13 11 9 7 5 3 12 2 4 6 8 10 1)	12	13
14	(13 11 9 7 5 3 14 2 4 6 8 10 12 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 3\ \ n\ \ 2\ \ 4\ \ 6\ \ \ldots\ \ n{-}2\ \ 1)$	$\leq n-1$	?
$2k{+}1$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 3\ \ n{-}1\ \ 2\ \ 4\ \ 6\ \ \ldots\ \ n{-}3\ \ 1)$	$\leq n-1$	?

Algoritmo 58 SbPRSR – Pseudocódigo para ordenar permutações da família 1

```
SbPRSR_Familia_1(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      π ← π · ρ_p(n − 2)ρ_s(π_n^{-1}) · ρ_p(π_{n−1}^{-1}) · ρ_p(n − 1)
3      for  i ← 3 to n − 4  do
4          π ← π · ρ_p(π_{i−1}^{-1} − 1)
5      π ← π · ρ_p(π_1^{-1})
6  else
7      π ← π · ρ_s(π_4^{-1}) · ρ_s(π_3^{-1}) · ρ_s(n − 1) · ρ_p(π_3^{-1}) · ρ_s(n − 1)
8      while  π_1 ≠ 1  do
9          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
```

Lema 130. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 3\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 1$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 58 ordena π_n em $n - 1$ operações. □

Tabela 37: SbPRSR – Família 2

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(1 3 7 5 2 6 4)	6	7
8	(1 3 7 5 2 8 6 4)	7	8
9	(1 3 5 9 7 2 8 6 4)	8	9
10	(1 3 5 9 7 2 10 8 6 4)	9	10
11	(1 3 5 7 11 9 2 10 8 6 4)	10	11
12	(1 3 5 7 11 9 2 12 10 8 6 4)	11	12
13	(1 3 5 7 9 13 11 2 12 10 8 6 4)	12	12
14	(1 3 5 7 9 13 11 2 14 12 10 8 6 4)	?	?
⋮	⋮	⋮	⋮
$2k$	(1 3 5 ... $n-5$ $n-1$ $n-3$ 2 n $n-2$ $n-4$ ... 4)	$\leq n-1$	?
$2k+1$	(1 3 5 ... $n-4$ n $n-2$ 2 $n-1$ $n-3$ $n-5$ ... 4)	$\leq n-1$	?

Algoritmo 59 SbPRSR – Pseudocódigo para ordenar permutações da família 2

```
SbPRSR_Familia_2(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      while  π_n ≠ n − 1  do
3          π ← π · ρ_s(π^{-1}_{π_n − 1} + 1)
4      π ← π · ρ_p(n − 3) · ρ_p(n − 1) · ρ_s(n − 1) · ρ_p(n − 3) · ρ_p(n − 4) · ρ_p(2)
5  else
6      while  π_n ≠ n  do
7          π ← π · ρ_s(π^{-1}_{π_n − 1} + 1)
8      π ← π · ρ_p(n − 3) · ρ_p(n − 1) · ρ_p(n − 2) · ρ_p(n − 3) · ρ_p(2)
```

Lema 131. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 5\ \ldots\ n{-}5\ n{-}1\ n{-}3\ 2\ n\ n{-}2\ n{-}4\ \ldots\ 4) & \textit{se } n \textit{ é par} \\ (1\ 3\ 5\ \ldots\ n{-}4\ n\ n{-}2\ 2\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 1$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 59 ordena π_n em $n-1$ operações. □

Tabela 38: SbPRSR – Família 3

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(7 4 6 2 5 3 1)	6	7
8	(7 4 6 8 2 5 3 1)	7	8
9	(9 4 6 8 2 7 5 3 1)	8	9
10	(9 4 6 8 10 2 7 5 3 1)	9	10
11	(11 4 6 8 10 2 9 7 5 3 1)	10	11
12	(11 4 6 8 10 12 2 9 7 5 3 1)	11	12
13	(13 4 6 8 10 12 2 11 9 7 5 3 1)	12	13
14	(13 4 6 8 10 12 14 2 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1)$	$\leq n-1$	?
$2k{+}1$	$(n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1)$	$\leq n-1$	?

Algoritmo 60 SbPRSR – Pseudocódigo para ordenar permutações da família 3

```
SbPRSR_Familia_3(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      π ← π · ρ_p(π_{n-2}^{-1}) · ρ_p(π_4^{-1}) · ρ_p(n − 2) · ρ_s(π_n^{-1})
3      while  π_1 ≠ 1  do
4          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
5  else
6      π ← π · ρ_s(π_{n-2}^{-1}) · ρ_p(π_{n-3}^{-1}) · ρ_s(2)
7      while  π_n ≠ n  do
8          π ← π · ρ_s(π_{π_n+1}^{-1} + 1)
9      π ← π · ρ_p(2) · ρ_p(n − 2) · ρ_p(2)
```

Lema 132. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 1$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 60 ordena π_n em $n-1$ operações. □

Tabela 39: SbPRSR – Família 4

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(7 4 6 2 5 3 1)	6	7
8	(7 6 8 4 2 5 3 1)	6	8
9	(9 6 8 4 2 7 5 3 1)	8	8
10	(9 8 10 6 4 2 7 5 3 1)	8	10
11	(11 8 10 6 4 2 9 7 5 3 1)	10	11
12	(11 10 12 8 6 4 2 9 7 5 3 1)	10	12
13	(13 10 12 8 6 4 2 11 9 7 5 3 1)	12	13
14	(13 12 14 10 8 6 4 2 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n{-}2\ \ n\ \ n{-}4\ \ n{-}6\ \ n{-}8\ \ \ldots\ \ 2\ \ n{-}3\ \ n{-}5\ \ n{-}7\ \ \ldots\ \ 1)$	$\leq n-2$	?
$2k{+}1$	$(n\ \ n{-}3\ \ n{-}1\ \ n{-}5\ \ n{-}7\ \ n{-}9\ \ \ldots\ \ 2\ \ n{-}2\ \ n{-}4\ \ n{-}6\ \ \ldots\ \ 1)$	$\leq n-1$	?

Algoritmo 61 SbPRSR – Pseudocódigo para ordenar permutações da família 4

```
SbPRSR_Familia_4(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      π ← π · ρ_s(π_{n−3}^{-1}) · ρ_s(π_n^{-1}) · ρ_p(n − 1)
3      while  π_1 ≠ 1  do
4          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
5  else
6      π ← π · ρ_s(π_{n−2}^{-1}) · ρ_p(2) · ρ_s(2) · ρ_p(2)
7      while  π_1 ≠ 1  do
8          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
```

Lema 133. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ \ n{-}2\ \ n\ \ n{-}4\ \ n{-}6\ \ n{-}8\ \ \ldots\ \ 2\ \ n{-}3\ \ n{-}5\ \ n{-}7\ \ \ldots\ \ 1) & \textit{se } n \textit{ é par} \\ (n\ \ n{-}3\ \ n{-}1\ \ n{-}5\ \ n{-}7\ \ n{-}9\ \ \ldots\ \ 2\ \ n{-}2\ \ n{-}4\ \ n{-}6\ \ \ldots\ \ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n-2$ *quando* n *é par e* $d_{\rho_p\rho_s}(\pi_n) = n-1$ *quando* n *é ímpar.*

Demonstração. Note que, quando n é par, $d_{\rho_p\rho_s}(\pi_n) \geq n{-}2$ porque $b_{\rho_p\rho_s}(\pi_n) = n{-}2$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Quando n é ímpar, $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$. Além disso, o Algoritmo 60 ordena π_n em $n{-}2$ operações quando n é par e em $n{-}1$ operações quando n é ímpar. □

Tabela 40: SbPRSR – Família 5

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
12	(1 3 7 5 2 6 4 8 10 12 9 11)	12	12
13	(1 3 7 5 2 6 4 8 10 12 9 13 11)	13	13
14	(1 3 7 5 2 6 4 8 10 12 14 9 13 11)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 7 5 2 6 4 8 10 12 ... n 9 $n-1$ $n-3$ $n-5$ ... 11)	$\leq n$	?
$2k+1$	(1 3 7 5 2 6 4 8 10 12 ... $n-1$ 9 n $n-2$ $n-4$ ... 11)	$\leq n$	?

Algoritmo 62 SbPRSR – Pseudocódigo para ordenar permutações da família 5

```
SbPRSR_Familia_5(π ← π_n, n)
1  while  π_n ≠ 9  do
2        π ← π · ρ_s(π^{-1}_{π_n − 1} + 1)
3  π ← π · ρ_s(π^{-1}_{10}) · ρ_s(π^{-1}_{9} + 1) · ρ_p(4) · ρ_p(6) · ρ_p(5) · ρ_p(7) · ρ_p(3) · ρ_p(6) · ρ_p(3) · ρ_p(2)
```

Lema 134. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 7\ 5\ 2\ 6\ 4\ 8\ 10\ 12\ \ldots\ n\ 9\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 11) & \textit{se } n \textit{ é par} \\ (1\ 3\ 7\ 5\ 2\ 6\ 4\ 8\ 10\ 12\ \ldots\ n{-}1\ 9\ n\ n{-}2\ n{-}4\ \ldots\ 11) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $n - 1 \leq d_{\rho_p\rho_s}(\pi_n) \leq n$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. O limitante superior é dado pelo Algoritmo 62. □

Tabela 41: SbPRSR – Família 6

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
10	(9 5 10 4 6 8 2 7 3 1)	9	10
11	(11 5 10 4 6 8 2 7 9 3 1)	10	11
12	(11 5 12 4 6 8 10 2 7 9 3 1)	11	12
13	(13 5 12 4 6 8 10 2 7 9 11 3 1)	12	13
14	(13 5 14 4 6 8 10 12 2 7 9 11 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1)$	$\leq n-1$	?
$2k{+}1$	$(n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1)$	$\leq n-1$	?

Algoritmo 63 SbPRSR – Pseudocódigo para ordenar permutações da família 6

```
SbPRSR_Familia_6(π ← π_n, n)
1  if  n mod 2 = 0  then
2      π ← π · ρ_p(2) · ρ_p(4) · ρ_p(n − 2) · ρ_s(π_{n−2}^{-1}) · ρ_s(π_5^{-1}) · ρ_s(π_n^{-1})
3      while  π_1 ≠ 1  do
4          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
5  else
6      π ← π · ρ_p(2) · ρ_p(4) · ρ_p(n − 2) · ρ_s(π_{n−3}^{-1}) · ρ_p(n − 1) · ρ_s(n − 1)
7      while  π_1 ≠ 6  do
8          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
9      π ← π · ρ_s(5) · ρ_s(3) · ρ_p(6) · ρ_p(2)
```

Lema 135. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1) & \textit{se } n \textit{ é par} \\ (n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 1$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 63 ordena π_n em $n - 1$ operações. □

Tabela 42: SbPRSR – Família 7

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(7 1 6 2 5 3 4)	5	7
8	(8 1 7 2 6 3 5 4)	6	8
9	(9 1 8 2 7 3 6 4 5)	7	9
10	(10 1 9 2 8 3 7 4 6 5)	8	10
11	(11 1 10 2 9 3 8 4 7 5 6)	9	11
12	(12 1 11 2 10 3 9 4 8 5 7 6)	10	12
13	(13 1 12 2 11 3 10 4 9 5 8 6 7)	11	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2})$	$\leq n-2$	?
$2k{+}1$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil)$	$\leq n-2$	?

Algoritmo 64 SbPRSR – Pseudocódigo para ordenar permutações da família 7

```
SbPRSR_Familia_7(π ← π_n, n)
1  for  i ← n − 1 downto 4  do
2        π ← π · ρ_s(i)
3  π ← π · ρ_p(2) · ρ_s(2)
```

Lema 136. *Seja*

$$\pi_n = \begin{cases} (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2}) & \text{se } n \text{ é par} \\ (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 2$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-2$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 64 ordena π_n em $n-2$ operações. □

Tabela 43: SbPRSR – Família 8

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(6 4 2 1 3 5 7)	5	7
8	(8 6 4 2 1 3 5 7)	6	8
9	(8 6 4 2 1 3 5 7 9)	7	9
10	(10 8 6 4 2 1 3 5 7 9)	8	10
11	(10 8 6 4 2 1 3 5 7 9 11)	9	11
12	(12 10 8 6 4 2 1 3 5 7 9 11)	10	12
13	(12 10 8 6 4 2 1 3 5 7 9 11 13)	11	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ n{-}2\ n{-}4\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n{-}1)$	$\leq n-2$	?
$2k{+}1$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n)$	$\leq n-2$	?

Algoritmo 65 SbPRSR – Pseudocódigo para ordenar permutações da família 8

```
SbPRSR_Familia_8(π ← π_n, n)
1  if  n mod 2 = 0  then
2        π ← π · ρ_p(n − 1) · ρ_s(n − 1)
3  while  π_1 ≠ 1  do
4        π ← π · ρ_p(π^{-1}_{π_1+1} − 1)
```

Lema 137. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n{-}1) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 2$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-2$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 65 ordena π_n em $n-2$ operações. □

Tabela 44: SbPRSR – Família 9

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
9	(2 8 5 7 4 6 3 9 1)	9	9
10	(2 10 5 7 4 6 8 3 9 1)	9	10
11	(2 10 5 7 9 4 6 8 3 11 1)	11	11
12	(2 12 5 7 9 4 6 8 10 3 11 1)	11	12
13	(2 12 5 7 9 11 4 6 8 10 3 13 1)	13	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 n 5 7 9 ... $n-3$ 4 6 8 ... $n-2$ 3 $n-1$ 1)	$\leq n-1$	?
$2k+1$	(2 $n-1$ 5 7 9 ... $n-2$ 4 6 8 ... $n-3$ 3 n 1)	$\leq n$	?

Algoritmo 66 SbPRSR – Pseudocódigo para ordenar permutações da família 9

```
SbPRSR_Familia_9(π ← π_n, n)
 1  if n mod 2 = 0 then
 2      π ← π · ρ_s(2) · ρ_p(3) · ρ_p(n − 1) · ρ_p(π_4^{-1}) · ρ_p(π_{n−2}^{-1}) · ρ_p(π_{n−5}^{-1})
 3      while π_1 ≠ 5 do
 4          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
 5      π ← π · ρ_p(π_4^{-1} − 1) · ρ_p(2) · ρ_p(n − 2)
 6  else
 7      π ← π · ρ_p(n − 1) · ρ_p(π_5^{-1}) · ρ_s(π_n^{-1}) · ρ_p(π_4^{-1} − 1) · ρ_p(π_{n−5}^{-1})
 8      while π_1 ≠ 4 do
 9          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
10      π ← π · ρ_p(π_3^{-1} − 1) · ρ_p(2) · ρ_p(n − 2) · ρ_p(2)
```

Lema 138. *Seja*

$$\pi_n = \begin{cases} (2\ n\ 5\ 7\ 9\ \ldots\ n{-}3\ 4\ 6\ 8\ \ldots\ n{-}2\ 3\ n{-}1\ 1) & \textit{se } n \textit{ é par} \\ (2\ n{-}1\ 5\ 7\ 9\ \ldots\ n{-}2\ 4\ 6\ 8\ \ldots\ n{-}3\ 3\ n\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n-1$ *quando* n *é par e* $n-1 \leq d_{\rho_p\rho_s}(\pi_n) \leq n$ *quando* n *é ímpar.*

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 66 ordena π_n em $n-1$ operações quando n é par e em n operações quando n é ímpar. □

Tabela 45: SbPRSR – Família 10

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
8	(8 1 6 4 2 5 3 7)	8	8
9	(9 1 7 5 3 6 4 2 8)	9	9
10	(10 1 8 6 4 2 7 5 3 9)	10	10
11	(11 1 9 7 5 3 8 6 4 2 10)	11	11
12	(12 1 10 8 6 4 2 9 7 5 3 11)	12	12
13	(13 1 11 9 7 5 3 10 8 6 4 2 12)	13	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ 1\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 3\ n{-}1)$	$\leq n$	?
$2k{+}1$	$(n\ 1\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 2\ n{-}1)$	$\leq n$	?

Algoritmo 67 SbPRSR – Pseudocódigo para ordenar permutações da família 10

```
SbPRSR_Familia_10(π ← π_n, n)
1  π ← π · ρ_p(n − 1) · ρ_p(n − 3) · ρ_s(2)
2  if  n mod 2 = 0  then
3        π ← π · ρ_s(π_2^{-1})
4  else
5        π ← π · ρ_s(π_3^{-1})
6  π ← π · ρ_p(π_{π_1 − 1}^{-1} − 1) · ρ_s(π_n^{-1})
7  while  π_1 ≠ 1  do
8        π ← π · ρ_p(π_{π_1 + 1}^{-1} − 1)
```

Lema 139. *Seja*

$$\pi_n = \begin{cases} (n\ 1\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 3\ n{-}1) & \text{se } n \text{ é par} \\ (n\ 1\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 2\ n{-}1) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $n - 1 \leq d_{\rho_p\rho_s}(\pi_n) \leq n$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. O limitante superior é dado pelo Algoritmo 67. □

Tabela 46: SbPRSR – Família 11

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
8	(1 7 5 3 6 4 2 8)	8	8
9	(1 9 7 5 3 6 4 2 8)	8	9
10	(1 9 7 5 3 8 6 4 2 10)	10	10
11	(1 11 9 7 5 3 8 6 4 2 10)	10	11
12	(1 11 9 7 5 3 10 8 6 4 2 12)	12	12
13	(1 13 11 9 7 5 3 10 8 6 4 2 12)	12	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 $n{-}1$ $n{-}3$ $n{-}5$ ... 3 $n{-}2$ $n{-}4$ $n{-}6$ ... 2 n)	$\leq n$	?
$2k{+}1$	(1 n $n{-}2$ $n{-}4$ ... 3 $n{-}3$ $n{-}5$ $n{-}7$ ... 2 $n{-}1$)	$\leq n-1$	?

Algoritmo 68 SbPRSR – Pseudocódigo para ordenar permutações da família 11

```
SbPRSR_Familia_11(π ← π_n, n)
1  if  n mod 2 = 0  then
2      π ← π · ρ_s(3) · ρ_p(π_{n-2}^{-1}) · ρ_p(n − 1) · ρ_s(n − 1) · ρ_p(π_{π_1−1}^{-1}) · ρ_s(π_1^{-1}) · ρ_s(π_n^{-1})
3      while  π_1 ≠ 1  do
4          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
5  else
6      π ← π · ρ_p(n − 2) · ρ_s(n − 3) · ρ_p(π_3^{-1} − 1)
7      while  π_1 ≠ 3  do
8          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
9      π ← π · ρ_p(n − 3) · ρ_p(n − 1)
```

Lema 140. *Seja*

$$\pi_n = \begin{cases} (1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 2\ n) & \textit{se } n \textit{ é par} \\ (1\ n\ n{-}2\ n{-}4\ \ldots\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 2\ n{-}1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $n-1 \leq d_{\rho_p\rho_s}(\pi_n) \leq n$ *quando* n *é par e* $d_{\rho_p\rho_s}(\pi_n) = n-1$ *quando* n *é ímpar.*

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 68 ordena π_n com n passos quando n é par e com $n-1$ passos quando n é ímpar. □

Tabela 47: SbPRSR – Família 12

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
7	(1 6 4 2 5 3 7)	6	7
8	(1 7 5 3 6 4 2 8)	8	8
9	(1 8 6 4 2 7 5 3 9)	8	9
10	(1 9 7 5 3 8 6 4 2 10)	10	10
11	(1 10 8 6 4 2 9 7 5 3 11)	10	11
12	(1 11 9 7 5 3 10 8 6 4 2 12)	12	12
13	(1 12 10 8 6 4 2 11 9 7 5 3 13)	12	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 2\ n)$	$\leq n$	?
$2k{+}1$	$(1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 3\ n)$	$\leq n-1$	?

Algoritmo 69 SbPRSR – Pseudocódigo para ordenar permutações da família 12

```
SbPRSR_Familia_12(π ← π_n, n)
1  if  n mod 2 = 0  then
2      π ← π · ρ_s(3) · ρ_p(π_{n-2}^{-1}) · ρ_p(n-1) · ρ_s(n-1) · ρ_p(π_{π_1-1}^{-1}) · ρ_s(π_1^{-1}) · ρ_s(π_n^{-1})
3      while  π_1 ≠ 1  do
4          π ← π · ρ_p(π_{π_1+1}^{-1} - 1)
5  else
6      π ← π · ρ_s(3) · ρ_s(π_2^{-1}) · ρ_p(n-1) · ρ_s(n-3)
7      while  π_1 ≠ n-2  do
8          π ← π · ρ_p(π_{π_1-1}^{-1} - 1)
9      π ← π · ρ_p(n-2)
```

Lema 141. *Seja*

$$\pi_n = \begin{cases} (1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 2\ n) & \text{se } n \text{ é par} \\ (1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 3\ n) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $n-1 \leq d_{\rho_p\rho_s}(\pi_n) \leq n$ *quando* n *é par e* $d_{\rho_p\rho_s}(\pi_n) = n-1$ *quando* n *é ímpar.*

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 69 ordena π_n com n passos quando n é par e com $n-1$ passos quando n é ímpar. □

Tabela 48: SbPRSR – Família 13

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
5	(2 4 5 3 1)	3	4
6	(2 4 6 5 3 1)	4	5
7	(2 4 6 7 5 3 1)	5	7
8	(2 4 6 8 7 5 3 1)	6	8
9	(2 4 6 8 9 7 5 3 1)	7	9
10	(2 4 6 8 10 9 7 5 3 1)	8	10
11	(2 4 6 8 10 11 9 7 5 3 1)	9	11
12	(2 4 6 8 10 12 11 9 7 5 3 1)	10	12
13	(2 4 6 8 10 12 13 11 9 7 5 3 1)	11	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 4 6 ... n $n-1$ $n-3$ $n-5$ ... 1)	$\leq n-2$	?
$2k+1$	(2 4 6 ... $n-1$ n $n-2$ $n-4$ ... 1)	$\leq n-2$	?

Algoritmo 70 SbPRSR – Pseudocódigo para ordenar permutações da família 13

```
SbPRSR_Familia_13(π ← π_n, n)
1  for i ← 4 to n do
2        π ← π · ρ_s(π_i^{-1})
3  π ← π · ρ_p(2)
```

Lema 142. *Seja*

$$\pi_n = \begin{cases} (2\ 4\ 6\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (2\ 4\ 6\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 2$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n - 2$ porque $b_{\rho_p\rho_s}(\pi_n) = n - 2$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 70 ordena π_n em $n - 2$ operações. □

Tabela 49: SbPRSR – Família 14

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
11	(1 3 10 5 8 6 2 7 11 9 4)	10	11
12	(1 3 12 10 5 8 6 2 7 11 9 4)	11	12
13	(1 3 12 10 5 8 6 2 7 13 11 9 4)	12	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 n $n-2$ $n-4$ ... 10 5 8 6 2 7 $n-1$ $n-3$ $n-5$ ... 9 4)	$\leq n-1$	?
$2k+1$	(1 3 $n-1$ $n-3$ $n-5$ ... 10 5 8 6 2 7 n $n-2$ $n-4$ ... 9 4)	$\leq n-1$	?

Algoritmo 71 SbPRSR – Pseudocódigo para ordenar permutações da família 14

```
SbPRSR_Familia_14(π ← π_n, n)
 1  if  n mod 2 = 0  then
 2      π ← π · ρ_s(3) · ρ_p(π_7^{-1}) · ρ_s(3) · ρ_p(π_8^{-1})
 3      while  π_1 ≠ n − 2  do
 4          π ← π · ρ_p(π_{π_1+1}^{-1} − 1)
 5      π ← π · ρ_s(2) · ρ_p(n − 2) · ρ_p(7) · ρ_p(5) · ρ_p(2)
 6  else
 7      π ← π · ρ_s(3)
 8      while  π_n ≠ 6  do
 9          π ← π · ρ_s(π_{π_n−1}^{-1} + 1)
10      π ← π · ρ_p(n − 2) · ρ_s(2) · ρ_p(7) · ρ_p(5) · ρ_p(2)
```

Lema 143. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ n\ n{-}2\ n{-}4\ \ldots\ 10\ 5\ 8\ 6\ 2\ 7\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 9\ 4) & \textit{se } n \textit{ é par} \\ (1\ 3\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 5\ 8\ 6\ 2\ 7\ n\ n{-}2\ n{-}4\ \ldots\ 9\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 1$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 71 ordena π_n em $n-1$ operações. □

Tabela 50: SbPRSR – Família 15

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
5	(1 5 2 4 3)	3	4
6	(1 6 2 5 3 4)	4	5
7	(1 7 2 6 3 5 4)	5	7
8	(1 8 2 7 3 6 4 5)	6	8
9	(1 9 2 8 3 7 4 6 5)	7	9
10	(1 10 2 9 3 8 4 7 5 6)	8	10
11	(1 11 2 10 3 9 4 8 5 7 6)	9	11
12	(1 12 2 11 3 10 4 9 5 8 6 7)	10	12
13	(1 13 2 12 3 11 4 10 5 9 6 8 7)	11	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 n 2 $n-1$ 3 $n-2$ ... $\frac{n}{2}$ $\frac{n}{2}+1$)	$\leq n-2$	?
$2k+1$	(1 n 2 $n-1$ 3 $n-2$ ... $\lceil\frac{n}{2}\rceil-1$ $\lceil\frac{n}{2}\rceil+1$ $\lceil\frac{n}{2}\rceil$)	$\leq n-2$	?

Algoritmo 72 SbPRSR – Pseudocódigo para ordenar permutações da família 15

```
SbPRSR_Familia_15(π ← π_n, n)
1  if  n mod 2 = 1  then
2      π ← π · ρ_s(π^{-1}_{π_n − 1} + 1)
3  while  π_n ≠ n  do
4      π ← π · ρ_s(π^{-1}_{π_n + 1} + 1) · ρ_s(π^{-1}_{π_n − 1} + 1)
```

Lema 144. *Seja*

$$\pi_n = \begin{cases} (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1) & \text{se } n \text{ é par} \\ (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 2$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-2$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 72 ordena π_n em $n-2$ operações. □

Tabela 51: SbPRSR – Família 16

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
8	(1 3 4 2 5 7 8 6)	5	8
9	(1 3 4 2 5 7 9 8 6)	6	9
10	(1 3 5 4 2 6 8 10 9 7)	7	10
11	(1 3 5 4 2 6 8 10 11 9 7)	8	11
12	(1 3 5 6 4 2 7 9 11 12 10 8)	9	12
13	(1 3 5 6 4 2 7 9 11 13 12 10 8)	10	13
14	(1 3 5 7 6 4 2 8 10 12 14 13 11 9)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$ (k par)	(1 3 5 ... $k-1$ k $k-2$ $k-4$ ... 2 $k+1$ $k+3$ $k+5$ ... $n-1$ n $n-2$ $n-4$ ... $k+2$)	$\leq n-3$	?
$2k$ (k ímpar)	(1 3 5 ... k $k-1$ $k-3$ $k-5$ ... 2 $k+1$ $k+3$ $k+5$ ... n $n-1$ $n-3$ $n-5$ ... $k+2$)	$\leq n-3$	?
$2k+1$ (k par)	(1 3 5 ... $k-1$ k $k-2$ $k-4$ ... 2 $k+1$ $k+3$ $k+5$ ... n $n-1$ $n-3$ $n-5$ ... $k+2$)	$\leq n-3$	?
$2k+1$ (k ímpar)	(1 3 5 ... k $k-2$ $k-3$ $k-5$ ... 2 $k+1$ $k+3$ $k+5$ ... $n-1$ n $n-2$ $n-4$ ... $k+2$)	$\leq n-3$	?

Algoritmo 73 SbPRSR – Pseudocódigo para ordenar permutações da família 16

```
SbPRSR_Familia_16(π ← π_n, n)
1  while  π_n ≠ n  do
2        π ← π · ρ_s(π^{-1}_{π_n − 1} + 1)
3  π ← π · ρ_p(π_2^{-1} − 1)
4  while  π_1 ≠ ⌊n/2⌋  do
5        π ← π · ρ_p(π^{-1}_{π_1 − 1} − 1)
6  π ← π · ρ_p(π^{-1}_{π_1 + 1} − 1) · ρ_p(2)
```

Lema 145. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 5\ \ldots\ k{-}1\ k\ k{-}2\ k{-}4\ \ldots\ 2\ k{+}1\ k{+}3\ k{+}5\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ k{+}2) \\ \qquad \textit{se } n = 2k \textit{ e } k \textit{ é par} \\ (1\ 3\ 5\ \ldots\ k\ k{-}1\ k{-}3\ k{-}5\ \ldots\ 2\ k{+}1\ k{+}3\ k{+}5\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ k{+}2) \\ \qquad \textit{se } n = 2k \textit{ e } k \textit{ é ímpar} \\ (1\ 3\ 5\ \ldots\ k{-}1\ k\ k{-}2\ k{-}4\ \ldots\ 2\ k{+}1\ k{+}3\ k{+}5\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ k{+}2) \\ \qquad \textit{se } n = 2k{+}1 \textit{ e } k \textit{ é par} \\ (1\ 3\ 5\ \ldots\ k\ k{-}1\ k{-}3\ k{-}5\ \ldots\ 2\ k{+}1\ k{+}3\ k{+}5\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ k{+}2) \\ \qquad \textit{se } n = 2k{+}1 \textit{ e } k \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 3$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-3$ porque $b_{\rho_p\rho_s}(\pi_n) = n-3$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 73 ordena π_n em $n-3$ operações. □

Tabela 52: SbPRSR – Família 17

n	π_n	$d_{\rho_p\rho_s}(\pi_n)$	$D_{\rho_p\rho_s}(n)$
11	(3 1 5 2 7 4 9 6 11 8 10)	10	11
12	(3 1 5 2 7 4 9 6 11 8 12 10)	11	12
13	(3 1 5 2 7 4 9 6 11 13 8 12 10)	12	13
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(3 1 5 2 7 4 9 6 11 13 15 ... $n-1$ 8 n $n-2$ $n-4$ ... 10)	$\leq n-1$	?
$2k+1$	(3 1 5 2 7 4 9 6 11 13 15 ... n 8 $n-1$ $n-3$ $n-5$ ... 10)	$\leq n-1$	?

Algoritmo 74 SbPRSR – Pseudocódigo para ordenar permutações da família 17

```
SbPRSR_Familia_17(π ← π_n, n)
1  π ← π · ρ_s(π_6^{-1}) · ρ_p(5) · ρ_s(2) · ρ_s(n − 1) · ρ_s(3)
2  while π_n ≠ n do
3        π ← π · ρ_s(π_{π_n − 1}^{-1} + 1)
4  π ← π · ρ_p(n − 2) · ρ_p(n − 1) · ρ_p(8) · ρ_p(4) · ρ_p(2)
```

Lema 146. *Seja*

$$\pi_n = \begin{cases} (3\ 1\ 5\ 2\ 7\ 4\ 9\ 6\ 11\ 13\ 15\ \ldots\ n{-}1\ 8\ n\ n{-}2\ n{-}4\ \ldots\ 10) & \textit{se } n \textit{ é par} \\ (3\ 1\ 5\ 2\ 7\ 4\ 9\ 6\ 11\ 13\ 15\ \ldots\ n\ 8\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 10) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\rho_s}(\pi_n) = n - 1$.

Demonstração. Note que $d_{\rho_p\rho_s}(\pi_n) \geq n-1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$ para qualquer π. Além disso, o Algoritmo 74 ordena π_n em $n-1$ operações. □

Tabela 53: SbPRSR – Resumo das famílias encontradas ($d = d_{\rho_p\rho_s}(\pi_n)$)

Família	$b_{\rho_p\rho_s}(\pi_n)$	d se n par	d se n ímpar
1	$n-1$	$d = n-1$	
2	$n-1$	$d = n-1$	
3	$n-1$	$d = n-1$	
4	$n-2$ quando n é par e $n-1$ quando n é ímpar	$d = n-2$	$d = n-1$
5	$n-1$	$n-1 \leq d \leq n$	
6	$n-1$	$d = n-1$	
7	$n-2$	$d = n-2$	
8	$n-2$	$d = n-2$	
9	$n-1$	$d = n-1$	$n-1 \leq d \leq n$
10	$n-1$	$d = n-1$	$n-1 \leq d \leq n$
11	$n-1$	$n-1 \leq d \leq n$	$d = n-1$
12	$n-1$	$n-1 \leq d \leq n$	$d = n-1$
13	$n-2$	$d = n-2$	
14	$n-1$	$d = n-1$	
15	$n-2$	$d = n-2$	
16	$n-3$	$d = n-3$	
17	$n-1$	$d = n-1$	

C.3 Famílias para SbPTST

Tabela 54: SbPTST – Família 1

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
7	(7 5 3 6 2 4 1)	5	5
8	(7 5 3 8 2 4 6 1)	5	6
9	(9 7 5 3 8 2 4 6 1)	6	6
10	(9 7 5 3 10 2 4 6 8 1)	6	7
11	(11 9 7 5 3 10 2 4 6 8 1)	7	8
12	(11 9 7 5 3 12 2 4 6 8 10 1)	7	8
13	(13 11 9 7 5 3 12 2 4 6 8 10 1)	?	?
14	(13 11 9 7 5 3 14 2 4 6 8 10 12 1)	?	?
⋮	⋮	⋮	⋮
$2k$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 1)$	$\leq \frac{n}{2}+1$	?
$2k{+}1$	$(n\ n{-}2\ n{-}4\ \ldots\ 3\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 75 SbPTST – Pseudocódigo para ordenar permutações da família 1

```
SbPTST_Familia_1(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_4^{-1}, n) · τ_p(π_{n-3}^{-1}, π_n^{-1})
 3      while  π_1 ≠ 3  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(n − 1, n + 1) · τ_p(2, 3)
 6  else
 7      π ← π · τ_s(π_{n-1}^{-1}, π_4^{-1}) · τ_p(3, n)τ_s(π_1^{-1}, n − 1)
 8      while  π_1 ≠ 3  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(π_2^{-1}, n − 1) · τ_p(2, 3)
```

Lema 147. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 3\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\tau_p\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\tau_p\tau_s}(\pi_n) = n-1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, tal limite inferior não é justo, pois, para ser, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O limitante superior é dado pelo Algoritmo 75. □

Tabela 55: SbPTST – Família 2

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
7	(1 3 7 5 2 6 4)	4	5
8	(1 3 7 5 2 8 6 4)	4	6
9	(1 3 5 9 7 2 8 6 4)	5	6
10	(1 3 5 9 7 2 10 8 6 4)	5	7
11	(1 3 5 7 11 9 2 10 8 6 4)	6	8
12	(1 3 5 7 11 9 2 12 10 8 6 4)	6	8
13	(1 3 5 7 9 13 11 2 12 10 8 6 4)	?	?
14	(1 3 5 7 9 13 11 2 14 12 10 8 6 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 5 ... $n-5$ $n-1$ $n-3$ 2 n $n-2$ $n-4$ ... 4)	$\leq \frac{n}{2}$	?
$2k+1$	(1 3 5 ... $n-4$ n $n-2$ 2 $n-1$ $n-3$ $n-5$ ... 4)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 76 SbPTST – Pseudocódigo para ordenar permutações da família 2

```
SbPTST_Familia_2(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      while  π_n ≠ n − 4  do
3          π ← π · τ_s(π^{-1}_{π_n+1}, n)
4      π ← π · τ_s(π^{-1}_{n−3}, π^{-1}_n) · τ_s(2, n) · τ_s(n − 2, n − 1) · τ_s(n − 4, n − 2)
5  else
6      while  π_n ≠ n − 1  do
7          π ← π · τ_s(π^{-1}_{π_n+1}, n)
8      π ← π · τ_s(π^{-1}_n, π^{-1}_n + 1) · τ_p(π^{-1}_2, π^{-1}_2 + 1) · τ_p(2, 3)
```

Lema 148. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 5\ \ldots\ n{-}5\ n{-}1\ n{-}3\ 2\ n\ n{-}2\ n{-}4\ \ldots\ 4) & \textit{se } n \textit{ é par} \\ (1\ 3\ 5\ \ldots\ n{-}4\ n\ n{-}2\ 2\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\tau_p\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil$.

Demonstração. Primeiro note que $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\tau_p\tau_s}(\pi_n) = n - 1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$, porém o limite inferior não é justo. Se fosse, sempre seria possível remover dois *breakpoints* a cada operação. Só existe uma possibilidade de fazer isso nos $\lceil \frac{n}{2} \rceil - 4$ primeiros passos: colocando o elemento par i que está na última posição da permutação entre os elementos $i - 1$ e $i + 1$. Depois disso, a permutação estará na forma (1 3 4 5 ... $n{-}6$ $n{-}5$ $n{-}4$ n $n{-}2$ 2 $n{-}1$ $n{-}3$) e são necessárias outras quatro operações para ordená-la. Além disso, o Algoritmo 76 ordena π_n com $\lceil \frac{n}{2} \rceil$ operações sempre. □

Tabela 56: SbPTST – Família 3

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
7	(7 4 6 2 5 3 1)	5	5
8	(7 4 6 8 2 5 3 1)	5	6
9	(9 4 6 8 2 7 5 3 1)	6	6
10	(9 4 6 8 10 2 7 5 3 1)	6	7
11	(11 4 6 8 10 2 9 7 5 3 1)	7	8
12	(11 4 6 8 10 12 2 9 7 5 3 1)	7	8
13	(13 4 6 8 10 12 2 11 9 7 5 3 1)	?	?
14	(13 4 6 8 10 12 14 2 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1)$	$\leq \frac{n}{2}+1$	?
$2k{+}1$	$(n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 77 SbPTST – Pseudocódigo para ordenar permutações da família 3

```
SbPTST_Familia_3(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(2, π_n^{-1}) · τ_p(π_2^{-1} + 1, n)
 3      while  π_1 ≠ 3  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(n − 1, n + 1) · τ_p(2, 3)
 6  else
 7      π ← π · τ_s(3, n − 2)
 8      while  π_n ≠ 2  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_p(π_3^{-1}, n) · τ_s(2, n − 2) · τ_p(4, 6) · τ_p(2, 3)
```

Lema 149. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\tau_p\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\tau_p\tau_s}(\pi_n) = n-1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, tal limite inferior não é justo, pois, para ser, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O limitante superior é dado pelo Algoritmo 77. □

Tabela 57: SbPTST – Família 4

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
5	(2 4 5 3 1)	3	3
6	(2 4 6 5 3 1)	4	4
7	(2 4 6 7 5 3 1)	5	5
8	(2 4 6 8 7 5 3 1)	6	6
9	(2 4 6 8 9 7 5 3 1)	7	6
10	(2 4 6 8 10 9 7 5 3 1)	8	7
11	(2 4 6 8 10 11 9 7 5 3 1)	9	8
12	(2 4 6 8 10 12 11 9 7 5 3 1)	10	8
13	(2 4 6 8 10 12 13 11 9 7 5 3 1)	?	?
14	(2 4 6 8 10 12 14 13 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 4 6 ... n $n-1$ $n-3$ $n-5$ ... 1)	$\leq \frac{n}{2}$	?
$2k+1$	(2 4 6 ... $n-1$ n $n-2$ $n-4$ ... 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 78 SbPTST – Pseudocódigo para ordenar permutações da família 4

```
SbPTST_Familia_4(π ← π_n, n ≥ 5)
1  π ← π · τ_s(2, n − 1)
2  while π_n ≠ n do
3        π ← π · τ_s(π^{-1}_{π_n+1}, n)
4  π ← π · τ_p(3, 4)
```

Lema 150. *Seja*

$$\pi_n = \begin{cases} (2\ 4\ 6\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (2\ 4\ 6\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\tau_p\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\tau_p\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que, quando n e par, $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil = \frac{n}{2}$ porque $b_{\tau_p\tau_s}(\pi_n) = n-1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é ímpar, $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-2}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$ porque $b_{\tau_p\tau_s}(\pi_n) = n-2$. Além disso, o Algoritmo 78 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil \frac{n}{2} \rceil - 1$ quando n é ímpar. □

Tabela 58: SbPTST – Família 5

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
5	(1 5 2 4 3)	2	3
6	(1 6 2 5 3 4)	2	4
7	(1 7 2 6 3 5 4)	3	5
8	(1 8 2 7 3 6 4 5)	3	6
9	(1 9 2 8 3 7 4 6 5)	4	6
10	(1 10 2 9 3 8 4 7 5 6)	4	7
11	(1 11 2 10 3 9 4 8 5 7 6)	5	8
12	(1 12 2 11 3 10 4 9 5 8 6 7)	5	8
13	(1 13 2 12 3 11 4 10 5 9 6 8 7)	?	?
14	(1 14 2 13 3 12 4 11 5 10 6 9 7 8)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1)$	$\leq \frac{n}{2}-1$	?
$2k{+}1$	$(1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil)$	$\leq \lceil\frac{n}{2}\rceil-1$	?

Algoritmo 79 SbPTST – Pseudocódigo para ordenar permutações da família 5

```
SbPTST_Familia_5(π ← π_n, n ≥ 5)
1  while  π_n ≠ n  do
2        π ← π · τ_s(π^{-1}_{π_n+1}, π^{-1}_{π_n+1} + 1)
```

Lema 151. *Seja*

$$\pi_n = \begin{cases} (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1) & \text{se } n \text{ é par} \\ (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\tau_p\tau_s}(\pi_n) = \lceil\frac{n}{2}\rceil - 1$.

Demonstração. Note que, quando n é par, $d_{\tau_p\tau_s}(\pi_n) \geq \lceil\frac{n}{2}\rceil - 1$ porque $b_{\tau_p\tau_s}(\pi_n) = n-2$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é ímpar, $d_{\tau_p\tau_s}(\pi_n) \geq \lceil\frac{n}{2}\rceil - 1$ porque $b_{\tau_p\tau_s}(\pi_n) = n-1$. Além disso, o Algoritmo 79 ordena π com $\lceil\frac{n}{2}\rceil - 1$ operações sempre. □

Tabela 59: SbPTST – Família 6

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
9	(9 5 8 4 6 2 7 3 1)	5	6
10	(9 5 10 4 6 8 2 7 3 1)	7	7
11	(11 5 10 4 6 8 2 7 9 3 1)	6	8
12	(11 5 12 4 6 8 10 2 7 9 3 1)	8	8
13	(13 5 12 4 6 8 10 2 7 9 11 3 1)	?	?
14	(13 5 14 4 6 8 10 12 2 7 9 11 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1)$	$\leq \frac{n}{2}+2$	?
$2k{+}1$	$(n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 80 SbPTST – Pseudocódigo para ordenar permutações da família 6

```
SbPTST_Familia_6(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(π_{n-2}^{-1}, n − 2) · τ_p(π_6^{-1}, π_2^{-1})
 3      while  π_n ≠ 2  do
 4          π ← π · τ_s(π_{π_n+1}^{-1}, n)
 5      π ← π · τ_s(π_1^{-1}, π_4^{-1}) · τ_p(n − 1, n) · τ_p(π_3^{-1}, n − 2) · τ_p(3, 5) · τ_p(2, 3)
 6  else
 7      π ← π · τ_s(π_2^{-1}, n − 2)
 8      while  π_n ≠ 2  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_p(4, n − 1) · τ_s(n − 5, n − 2) · τ_s(2, n) · τ_p(n − 3, n)
```

Lema 152. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1) & \text{se } n \text{ é par} \\ (n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\tau_p\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 2$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\tau_p\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\tau_p\tau_s}(\pi_n) = n{-}1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 80. □

Tabela 60: SbPTST – Família 7

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
4	(4 1 3 2)	2	3
5	(5 1 4 2 3)	2	3
6	(6 1 5 2 4 3)	3	4
7	(7 1 6 2 5 3 4)	3	5
8	(8 1 7 2 6 3 5 4)	4	6
9	(9 1 8 2 7 3 6 4 5)	4	6
10	(10 1 9 2 8 3 7 4 6 5)	5	7
11	(11 1 10 2 9 3 8 4 7 5 6)	5	8
12	(12 1 11 2 10 3 9 4 8 5 7 6)	6	8
13	(13 1 12 2 11 3 10 4 9 5 8 6 7)	?	?
14	(14 1 13 2 12 3 11 4 10 5 9 6 8 7)	?	?
⋮	⋮	⋮	⋮
$2k$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2})$	$\leq \frac{n}{2}$	?
$2k{+}1$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil)$	$\leq \lceil\frac{n}{2}\rceil - 1$	?

Algoritmo 81 SbPTST – Pseudocódigo para ordenar permutações da família 7

SbPTST_Familia_7($\pi \leftarrow \pi_{\mathtt{n}}, \mathtt{n} \geq 4$)
1 **while** $\pi_1 \neq \lceil \frac{\mathtt{n}}{2} \rceil + 1$ **do**
2 $\quad \pi \leftarrow \pi \cdot \tau_p(\pi^{-1}_{\pi_1 - 1}, \pi^{-1}_{\pi_1 - 1} + 1)$
3 $\pi \leftarrow \pi \cdot \tau_p(\pi_1^{-1}, \mathtt{n} + 1)$

Lema 153. *Seja*

$$\pi_n = \begin{cases} (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2}) & \text{se } n \text{ é par} \\ (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil\frac{n}{2}\rceil{+}1\ \lceil\frac{n}{2}\rceil{-}1\ \lceil\frac{n}{2}\rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\tau_p\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\tau_p\tau_s}(\pi_n) = \lceil\frac{n}{2}\rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que, quando n é par, $d_{\tau_p\tau_s}(\pi_n) \geq \lceil\frac{n-1}{2}\rceil = \frac{n}{2}$ porque $b_{\tau_p\tau_s}(\pi_n) = n - 1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é ímpar, $d_{\tau_p\tau_s}(\pi_n) \geq \lceil\frac{n-2}{2}\rceil = \lceil\frac{n}{2}\rceil - 1$ porque $b_{\tau_p\tau_s}(\pi_n) = n - 2$. Além disso, o Algoritmo 81 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil\frac{n}{2}\rceil - 1$ quando n é ímpar. □

Tabela 61: SbPTST – Família 8

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
4	(4 2 1 3)	2	3
5	(4 2 1 3 5)	2	3
6	(6 4 2 1 3 5)	3	4
7	(6 4 2 1 3 5 7)	3	5
8	(8 6 4 2 1 3 5 7)	4	6
9	(8 6 4 2 1 3 5 7 9)	4	6
10	(10 8 6 4 2 1 3 5 7 9)	5	7
11	(10 8 6 4 2 1 3 5 7 9 11)	5	8
12	(12 10 8 6 4 2 1 3 5 7 9 11)	6	8
13	(12 10 8 6 4 2 1 3 5 7 9 11 13)	?	?
14	(14 12 10 8 6 4 2 1 3 5 7 9 11 13)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 2\ \ 1\ \ 3\ \ 5\ \ \ldots\ \ n{-}1)$	$\leq \frac{n}{2}$	?
$2k{+}1$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 2\ \ 1\ \ 3\ \ 5\ \ \ldots\ \ n)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 82 SbPTST – Pseudocódigo para ordenar permutações da família 8

```
SbPTST_Familia_8(π ← π_n, n ≥ 4)
1  while  π_1 ≠ 1  do
2        π ← π · τ_p(2, π^{-1}_{π_1+1})
```

Lema 154. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n{-}1) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\tau_p\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\tau_p\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\tau_p\tau_s}(\pi_n) = n - 1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \frac{n}{2}$ e quando n é ímpar, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$. Além disso, o Algoritmo 82 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil \frac{n}{2} \rceil - 1$ quando n é ímpar. □

Tabela 62: SBPTST – Família 9

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
9	(2 8 5 7 4 6 3 9 1)	6	6
10	(2 10 5 7 4 6 8 3 9 1)	5	7
11	(2 10 5 7 9 4 6 8 3 11 1)	7	8
12	(2 12 5 7 9 4 6 8 10 3 11 1)	6	8
13	(2 12 5 7 9 11 4 6 8 10 3 13 1)	?	?
14	(2 14 5 7 9 11 4 6 8 10 12 3 13 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 n 5 7 9 ... $n-3$ 4 6 8 ... $n-2$ 3 $n-1$ 1)	$\leq \frac{n}{2}$	?
$2k+1$	(2 $n-1$ 5 7 9 ... $n-2$ 4 6 8 ... $n-3$ 3 n 1)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 83 SBPTST – Pseudocódigo para ordenar permutações da família 9

```
SBPTST_FAMILIA_9(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(2, n − 2) · τ_s(3, π_4^{-1})
 3      while  π_n ≠ n  do
 4          π ← π · τ_s(π^{-1}_{π_n+1}, n)
 5      π ← π · τ_p(n − 1, n)
 6  else
 7      π ← π · τ_s(n − 2, n − 1) · τ_s(π_4^{-1}, π_n^{-1})
 8      while  π_n ≠ 4  do
 9          π ← π · τ_s(π^{-1}_{π_n+1}, n)
10      π ← π · τ_s(π_5^{-1}, π_n^{-1}) · τ_s(2, 4) · τ_p(2, 3)
```

Lema 155. *Seja*

$$\pi_n = \begin{cases} (2\ n\ 5\ 7\ 9\ \ldots\ n{-}3\ 4\ 6\ 8\ \ldots\ n{-}2\ 3\ n{-}1\ 1) & \textit{se } n \textit{ é par} \\ (2\ n{-}1\ 5\ 7\ 9\ \ldots\ n{-}2\ 4\ 6\ 8\ \ldots\ n{-}3\ 3\ n\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\tau_p\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil \leq d_{\tau_p\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\tau_p\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\tau_p\tau_s}(\pi_n) = n - 1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \frac{n}{2}$. Quando n é ímpar, esse limitante não é justo, pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível na primeira operação. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. Além disso, o Algoritmo 83 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil \frac{n}{2} \rceil + 1$ quando n é ímpar. □

Tabela 63: SbPTST – Família 10

n	π_n	$d_{\tau_p\tau_s}(\pi_n)$	$D_{\tau_p\tau_s}(n)$
11	(1 3 10 5 8 6 2 7 11 9 4)	6	8
12	(1 3 12 10 5 8 6 2 7 11 9 4)	7	8
13	(1 3 12 10 5 8 6 2 7 13 11 9 4)	?	?
14	(1 3 14 12 10 5 8 6 2 7 13 11 9 4)	?	?
⋮	⋮	⋮	⋮
$2k$	(1 3 n $n-2$ $n-4$ ... 10 5 8 6 2 7 $n-1$ $n-3$ $n-5$ ... 9 4)	$\leq n-5$	?
$2k+1$	(1 3 $n-1$ $n-3$ $n-5$ ... 10 5 8 6 2 7 n $n-2$ $n-4$ ... 9 4)	$\leq n-5$	?

Algoritmo 84 SbPTST – Pseudocódigo para ordenar permutações da família 10

```
SbPTST_Familia_10(π ← π_n, n)
1  π ← π · τ_s(π_5^{-1}, π_11^{-1}) · τ_s(π_8^{-1}, π_8^{-1} + 1) · τ_s(π_9^{-1}, π_9^{-1} + 1) · τ_s(π_10^{-1}, π_4^{-1})
2  for i ← 12 to n do
3        π ← π · τ_s(π_i^{-1}, π_i^{-1} + 1)
4  π ← π · τ_p(6, 7) · τ_p(2, 3)
```

Lema 156. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ n\ n{-}2\ n{-}4\ \ldots\ 10\ 5\ 8\ 6\ 2\ 7\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 9\ 4) & \textit{se } n \textit{ é par} \\ (1\ 3\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 5\ 8\ 6\ 2\ 7\ n\ n{-}2\ n{-}4\ \ldots\ 9\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\left\lceil \frac{n}{2} \right\rceil \leq d_{\tau_p\tau_s}(\pi_n) \leq n-5$.

Demonstração. Primeiro note que $d_{\tau_p\tau_s}(\pi_n) \geq \left\lceil \frac{n-1}{2} \right\rceil$ porque $b_{\tau_p\tau_s}(\pi_n) = n-1$ e $d_{\tau_p\tau_s}(\pi) \geq \left\lceil \frac{b_{\tau_p\tau_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\left\lceil \frac{n-1}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil$. Quando n é ímpar, esse limitante não é justo, pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível na primeira operação. Portanto, a distância deve ser pelo menos $\left\lceil \frac{n-1}{2} \right\rceil + 1 = \left\lceil \frac{n}{2} \right\rceil - 1 + 1 = \left\lceil \frac{n}{2} \right\rceil$. O limitante superior é dado pelo Algoritmo 84. □

Tabela 64: SbPTST – Resumo das famílias encontradas ($d = d_{\tau_p\tau_s}(\pi_n)$)

Família	$b_{\tau_p\tau_s}(\pi_n)$	d se n par	d se n ímpar
1	$n-1$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
2	$n-1$	$d = \lceil \frac{n}{2} \rceil$	
3	$n-1$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$	
4	$n-1$ se n par e $n-2$ se n ímpar	$d = \frac{n}{2}$	$d = \lceil \frac{n}{2} \rceil - 1$
5	$n-2$ se n par e $n-1$ se n ímpar	$d = \lceil \frac{n}{2} \rceil - 1$	
6	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil \frac{n}{2} \rceil - 1 \leq d \leq \lceil \frac{n}{2} \rceil$
7	$n-1$ se n par e $n-2$ se n ímpar	$d = \frac{n}{2}$	$d = \lceil \frac{n}{2} \rceil - 1$
8	$n-1$	$d = \frac{n}{2}$	$d = \lceil \frac{n}{2} \rceil - 1$
9	$n-1$	$d = \frac{n}{2}$	$\lceil \frac{n}{2} \rceil \leq d \leq \lceil \frac{n}{2} \rceil + 1$
10	$n-1$	$\lceil \frac{n}{2} \rceil \leq d \leq n-5$	

C.4 Famílias para SbPRPTSRST

Tabela 65: SbPRPTSRST – Família 1

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
7	(7 5 3 6 2 4 1)	5	5
8	(7 5 3 8 2 4 6 1)	5	5
9	(9 7 5 3 8 2 4 6 1)	6	6
10	(9 7 5 3 10 2 4 6 8 1)	6	6
11	(11 9 7 5 3 10 2 4 6 8 1)	7	7
12	(11 9 7 5 3 12 2 4 6 8 10 1)	7	7
13	(13 11 9 7 5 3 12 2 4 6 8 10 1)	?	?
14	(13 11 9 7 5 3 14 2 4 6 8 10 12 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 3\ \ n\ \ 2\ \ 4\ \ 6\ \ \ldots\ \ n{-}2\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 3\ \ n{-}1\ \ 2\ \ 4\ \ 6\ \ \ldots\ \ n{-}3\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 85 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 1

```
SbPRPTSRST_Familia_1(π ← π_n, n ≥ 7)
 1  if n mod 2 = 0 then
 2      π ← π · τ_p(π_4^{-1}, n) · τ_p(π_{n-3}^{-1}, π_n^{-1})
 3      while π_1 ≠ 3 do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(n − 1, n + 1) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_3^{-1}, π_4^{-1}) · τ_s(π_2^{-1}, π_2^{-1} + 2)
 8      while π_1 ≠ n − 4 do
 9          π ← π · τ_p(π_{π_1+2}^{-1}, π_{π_1+2}^{-1} + 2)
10      π ← π · τ_p(π_{n-1}^{-1}, π_1^{-1}) · ρ_p(n − 1) · ρ_p(2)
```

Lema 157. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 3\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-1}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 85. □

Tabela 66: SbPRPTSRST – Família 2

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
7	(1 3 7 5 2 6 4)	4	5
8	(1 3 7 5 2 8 6 4)	4	5
9	(1 3 5 9 7 2 8 6 4)	5	6
10	(1 3 5 9 7 2 10 8 6 4)	5	6
11	(1 3 5 7 11 9 2 10 8 6 4)	6	7
12	(1 3 5 7 11 9 2 12 10 8 6 4)	6	7
13	(1 3 5 7 9 13 11 2 12 10 8 6 4)	?	?
14	(1 3 5 7 9 13 11 2 14 12 10 8 6 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 3 5 ... $n-5$ $n-1$ $n-3$ 2 n $n-2$ $n-4$ ... 4)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k+1$	(1 3 5 ... $n-4$ n $n-2$ 2 $n-1$ $n-3$ $n-5$ ... 4)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 86 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 2

```
SbPRPTSRST_Familia_2(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      π ← π · τ_s(2, π_2^{-1})
3      while  π_n ≠ 3  do
4          π ← π · τ_s(π_{π_n - 1}^{-1}, n)
5      π ← π · ρ_s(3)
6  else
7      while  π_{n-2} ≠ 2  do
8          π ← π · τ_s(π_{π_n + 1}^{-1}, n)
9      π ← π · τ_p(π_2^{-1} − 1, π_2^{-1} + 1) · τ_p(2, n) · ρ_s(n − 3) · ρ_p(2)
```

Lema 158. *Seja*

$$\pi_n = \begin{cases} (1\ 3\ 5\ \ldots\ n{-}5\ n{-}1\ n{-}3\ 2\ n\ n{-}2\ n{-}4\ \ldots\ 4) & \textit{se } n \textit{ é par} \\ (1\ 3\ 5\ \ldots\ n{-}4\ n\ n{-}2\ 2\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \frac{n}{2}$. Além disso, o Algoritmo 86 ordena π_n com $\lceil \frac{n}{2} \rceil$ operações. □

Tabela 67: SbPRPTSRST – Família 3

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
7	(7 4 6 2 5 3 1)	5	5
8	(7 4 6 8 2 5 3 1)	5	5
9	(9 4 6 8 2 7 5 3 1)	6	6
10	(9 4 6 8 10 2 7 5 3 1)	6	6
11	(11 4 6 8 10 2 9 7 5 3 1)	7	7
12	(11 4 6 8 10 12 2 9 7 5 3 1)	7	7
13	(13 4 6 8 10 12 2 11 9 7 5 3 1)	?	?
14	(13 4 6 8 10 12 14 2 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 87 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 3

```
SbPRPTSRST_Familia_3(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(π_2^{-1}, π_2^{-1} + 2) · τ_s(π_{n-2}^{-1}, π_n^{-1} + 1) · τ_p(2, n)
 3      while  π_1 ≠ n − 4  do
 4          π ← π · τ_p(2, π_{π_1 − 1}^{-1})
 5      π ← π · ρ_p(π_2^{-1}) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_{n-1}^{-1}, π_{n-1}^{-1} + 3) · τ_p(2, 4) · τ_p(π_n^{-1} + 1, n + 1)
 8      while  π_1 ≠ n − 3  do
 9          π ← π · τ_p(2, π_{π_1 − 1}^{-1})
10      π ← π · ρ_p(π_2^{-1}) · ρ_p(2)
```

Lema 159. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 4\ 6\ 8\ \ldots\ n\ 2\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (n\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, esse limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O limitante superior é dado pelo Algoritmo 87. □

Tabela 68: SbPRPTSRST – Família 4

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
8	(7 4 6 8 2 5 3 1)	5	5
9	(9 7 4 6 8 2 5 3 1)	6	6
10	(9 7 4 6 8 10 2 5 3 1)	6	6
11	(11 9 7 4 6 8 10 2 5 3 1)	7	7
12	(11 9 7 4 6 8 10 12 2 5 3 1)	7	7
13	(13 11 9 7 4 6 8 10 12 2 5 3 1)	?	?
14	(13 11 9 7 4 6 8 10 12 14 2 5 3 1)	?	?
⋮	⋮	⋮	⋮
$2k$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 7\ 4\ 6\ 8\ \ldots\ n\ 2\ 5\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ n{-}2\ n{-}4\ \ldots\ 7\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ 5\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 88 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 4

```
SbPRPTSRST_Familia_4(π ← π_n, n ≥ 8)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(n − 3, n − 1) · τ_s(π_6^{-1}, π_3^{-1})
 3      while  π_1 ≠ 4  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · τ_p(π_6^{-1}, π_2^{-1}) · τ_s(π_8^{-1}, π_7^{-1})
 8      while  π_n ≠ n  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_p(4, 6) · τ_p(5, 8) · ρ_p(4) · ρ_p(2)
```

Lema 160. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 7\ 4\ 6\ 8\ \ldots\ n\ 2\ 5\ 3\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 7\ 4\ 6\ 8\ \ldots\ n{-}1\ 2\ 5\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, esse limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O limitante superior é dado pelo Algoritmo 88. □

Tabela 69: SbPRPTSRST – Família 5

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(9 5 8 6 3 7 2 4 1)	6	6
10	(9 5 10 8 6 3 7 2 4 1)	6	6
11	(11 5 10 8 6 3 7 9 2 3 1)	7	7
12	(11 5 12 10 8 6 3 7 9 2 4 1)	7	7
13	(13 5 12 10 8 6 3 7 9 11 2 4 1)	?	?
14	(13 5 14 12 10 8 6 3 7 9 11 2 4 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 5\ n\ n{-}2\ n{-}4\ \ldots\ 6\ 3\ 7\ 9\ 11\ \ldots\ n{-}3\ 2\ 4\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 5\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 6\ 3\ 7\ 9\ 11\ \ldots\ n{-}2\ 2\ 4\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 89 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 5

```
SbPRPTSRST_Familia_5(π ← π_n, n ≥ 9)
 1  if n mod 2 = 0 then
 2      π ← π · τ_p(3, n − 1) · τ_p(π_3^{-1}, π_5^{-1}) · τ_s(π_2^{-1}, π_{n−2}^{-1}) · τ_p(π_{n−4}^{-1}, n − 1)
 3      while π_1 ≠ 6 do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(6) · ρ_p(3)
 6  else
 7      π ← π · τ_p(3, n − 1) · τ_p(π_3^{-1}, π_5^{-1})
 8      while π_1 ≠ 2 do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1} + 1)
10      π ← π · ρ_s(2) · ρ_p(2)
```

Lema 161. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 5\ n\ n{-}2\ n{-}4\ \ldots\ 6\ 3\ 7\ 9\ 11\ \ldots\ n{-}3\ 2\ 4\ 1) & \textit{se } n \textit{ é par} \\ (n\ 5\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 6\ 3\ 7\ 9\ 11\ \ldots\ n{-}2\ 2\ 4\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, esse limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O limitante superior é dado pelo Algoritmo 89. □

Tabela 70: SbPRPTSRST – Família 6

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
7	(1 5 2 7 4 6 3)	4	5
8	(1 5 2 7 6 4 8 3)	4	5
9	(1 5 7 2 9 6 4 8 3)	5	6
10	(1 5 7 2 9 8 6 4 10 3)	5	6
11	(1 5 7 9 2 11 8 6 4 10 3)	6	7
12	(1 5 7 9 2 11 10 8 6 4 12 3)	6	7
13	(1 5 7 9 11 2 13 10 8 6 4 12 3)	?	?
14	(1 5 7 9 11 2 13 12 10 8 6 4 14 3)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(1 5 7 9 ... $n{-}3$ 2 $n{-}1$ $n{-}2$ $n{-}4$ $n{-}6$ ... 4 n 3)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	(1 5 7 9 ... $n{-}2$ 2 n $n{-}3$ $n{-}5$ $n{-}7$ ... 4 $n{-}1$ 3)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 90 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 6

```
SbPRPTSRST_Familia_6(π ← π_n, n ≥ 7)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(π_2^{-1}, π_{n-2}^{-1}) · τ_s(π_n^{-1}, π_n^{-1} + 1) · τ_p(π_{n-4}^{-1}, n − 1)
 3      while  π_1 ≠ 4  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(4)
 6  else
 7      π ← π · τ_s(π_2^{-1}, n − 1)
 8      while  π_n ≠ n  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_p(π_3^{-1}, n) · ρ_p(3)
```

Lema 162. *Seja*

$$\pi_n = \begin{cases} (1\ 5\ 7\ 9\ \ldots\ n{-}3\ 2\ n{-}1\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 4\ n\ 3) & \textit{se } n \textit{ é par} \\ (1\ 5\ 7\ 9\ \ldots\ n{-}2\ 2\ n\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 4\ n{-}1\ 3) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Primeiro note que, quando n é par, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n - 2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. No entanto, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser no mínimo $\lceil \frac{n}{2} \rceil - 1 + 1 = \frac{n}{2}$. Quando n é ímpar, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$. Além disso, o Algoritmo 90 ordena π_n com $\lceil \frac{n}{2} \rceil$ operações. □

Tabela 71: SbPRPTSRST – Família 7

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(5 7 2 9 4 1 8 6 3)	5	6
10	(5 7 2 9 4 1 10 8 6 3)	5	6
11	(5 7 9 2 11 4 1 10 8 6 3)	6	7
12	(5 7 9 2 11 4 1 12 10 8 6 3)	6	7
13	(5 7 9 11 2 13 4 1 12 10 8 6 3)	?	?
14	(5 7 9 11 2 13 4 1 14 12 10 8 6 3)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(5 7 9 ... $n-3$ 2 $n-1$ 4 1 n $n-2$ $n-4$ ... 6 3)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k+1$	(5 7 9 ... $n-2$ 2 n 4 1 $n-1$ $n-3$ $n-5$ ... 6 3)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 91 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 7

```
SbPRPTSRST_Familia_7(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(π_2^{-1}, π_{n-4}^{-1})
 3      while  π_3 ≠ 6  do
 4          π ← π · τ_s(π_{π_n - 1}^{-1}, π_{π_n - 1}^{-1} + 2)
 5      π ← π · τ_s(2, π_4^{-1}) · τ_s(π_{n-2}^{-1}, n) · τ_p(3, n − 1) · ρ_s(2)
 6  else
 7      π ← π · τ_s(π_2^{-1}, π_{n-1}^{-1}) · ρ_s(n − 2) · τ_p(π_{n-1}^{-1} + 1, n)
 8      while  π_1 ≠ 3  do
 9          π ← π · τ_p(2, π_{π_1 + 1}^{-1})
10      π ← π · ρ_p(3)
```

Lema 163. *Seja*

$$\pi_n = \begin{cases} (5\ 7\ 9\ \ldots\ n{-}3\ 2\ n{-}1\ 4\ 1\ n\ n{-}2\ n{-}4\ \ldots\ 6\ 3) & \text{se } n \text{ é par} \\ (5\ 7\ 9\ \ldots\ n{-}2\ 2\ n\ 4\ 1\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 6\ 3) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \frac{n}{2}$. Quando n é ímpar, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$. Além disso, o Algoritmo 91 ordena π_n com $\lceil \frac{n}{2} \rceil$ operações. □

Tabela 72: SbPRPTSRST – Família 8

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(5 2 6 1 3 7 4 9 8)	5	6
10	(5 2 6 8 1 3 7 4 10 9)	5	6
11	(5 2 6 8 1 3 9 7 4 11 10)	6	7
12	(5 2 6 8 10 1 3 9 7 4 12 11)	6	7
13	(5 2 6 8 10 1 3 11 9 7 4 13 12)	?	?
14	(5 2 6 8 10 12 1 3 11 9 7 4 14 13)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(5 2 6 8 10 ... n−2 1 3 n−3 n−5 n−7 ... 7 4 n n−1)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k$+1	(5 2 6 8 10 ... n−3 1 3 n−2 n−4 n−6 ... 7 4 n n−1)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 92 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 8

```
SbPRPTSRST_Familia_8(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(π_{n−2}^{-1}, π_3^{-1} + 1) · τ_s(2, π_4^{-1})
 3      while  π_n ≠ n − 3  do
 4          π ← π · τ_s(π_{π_n+1}^{-1}, n)
 5      π ← π · ρ_p(π_2^{-1}) · τ_s(π_{n−2}^{-1}, π_4^{-1}) · τ_p(3, 4)
 6  else
 7      π ← π · τ_s(π_3^{-1} + 1, π_4^{-1})
 8      while  π_n ≠ n − 2  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · ρ_s(π_1^{-1}) · τ_s(2, n − 2) · ρ_p(5) · ρ_p(2)
```

Lema 164. *Seja*

$$\pi_n = \begin{cases} (5\ 2\ 6\ 8\ 10\ \ldots\ n{-}2\ 1\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 7\ 4\ n\ n{-}1) & \textit{se } n \textit{ é par} \\ (5\ 2\ 6\ 8\ 10\ \ldots\ n{-}3\ 1\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 7\ 4\ n\ n{-}1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser no mínimo $\lceil \frac{n}{2} \rceil - 1 + 1 = \frac{n}{2}$. Além disso, o Algoritmo 92 ordena π_n com $\lceil \frac{n}{2} \rceil$ operações. □

Tabela 73: SbPRPTSRST – Família 9

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
11	(2 8 1 6 3 10 5 7 9 11 4)	6	7
12	(2 10 1 6 8 3 12 5 7 9 11 4)	7	7
13	(2 10 1 6 8 3 12 5 7 9 11 13 4)	?	?
14	(2 12 1 6 8 10 3 14 5 7 9 11 13 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 n−2 1 6 8 10 ... n−4 3 n 5 7 9 ... n−1 4)	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k$+1	(2 n−3 1 6 8 10 ... n−5 3 n−1 5 7 9 ... n 4)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 93 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 9

```
SbPRPTSRST_Familia_9(π ← π_n, n ≥ 11)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(π_3^{-1}, n − 2)
 3      while  π_n ≠ n  do
 4          π ← π · τ_s(π_{π_n+1}^{-1}, n)
 5      π ← π · τ_p(3, π_{π_n−1}^{-1}) · τ_p(π_2^{-1}, π_2^{-1} + 1) · τ_p(π_4^{-1}, n) · ρ_p(4)
 6  else
 7      π ← π · τ_s(3, n − 2) · τ_s(2, π_3^{-1})
 8      while  π_n ≠ 1  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_s(π_5^{-1}, π_n^{-1}) · τ_s(3, 5) · τ_p(4, 5)
```

Lema 165. *Seja*

$$\pi_n = \begin{cases} (2\ n{-}2\ 1\ 6\ 8\ 10\ \ldots\ n{-}4\ 3\ n\ 5\ 7\ 9\ \ldots\ n{-}1\ 4) & \textit{se } n \textit{ é par} \\ (2\ n{-}3\ 1\ 6\ 8\ 10\ \ldots\ n{-}5\ 3\ n{-}1\ 5\ 7\ 9\ \ldots\ n\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\frac{n}{2} \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \frac{n}{2} + 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 93. □

Tabela 74: SbPRPTSRST – Família 10

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(6 2 7 9 4 8 5 1 3)	5	6
10	(10 6 2 7 9 4 8 5 1 3)	6	6
11	(10 6 2 7 9 11 4 8 5 1 3)	6	7
12	(12 10 6 2 7 9 11 4 8 5 1 3)	7	7
13	(12 10 6 2 7 9 11 13 4 8 5 1 3)	?	?
14	(14 12 10 6 2 7 9 11 13 4 8 5 1 3)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 10\ 6\ 2\ 7\ 9\ 11\ \ \ldots\ \ n{-}1\ 4\ 8\ 5\ 1\ 3)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 10\ 6\ 2\ 7\ 9\ 11\ \ \ldots\ \ n\ 4\ 8\ 5\ 1\ 3)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 94 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 10

```
SbPRPTSRST_Familia_10(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_9^{-1}, π_4^{-1}) · τ_p(π_2^{-1}, π_5^{-1}) · τ_s(π_4^{-1}, π_6^{-1})
 3      while  π_n ≠ n  do
 4          π ← π · τ_s(π_{π_n+1}^{-1}, n)
 5      π ← π · ρ_p(4) · τ_p(π_2^{-1}, π_4^{-1} + 1) · ρ_p(2)
 6  else
 7      π ← π · τ_s(π_2^{-1}, π_2^{-1} + 1)
 8      while  π_1 ≠ 6  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(π_8^{-1}, π_3^{-1}) · τ_s(π_1^{-1}, π_4^{-1}) · τ_p(2, π_9^{-1}) · ρ_p(5)
```

Lema 166. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 10\ 6\ 2\ 7\ 9\ 11\ \ldots\ n{-}1\ 4\ 8\ 5\ 1\ 3) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 6\ 2\ 7\ 9\ 11\ \ldots\ n\ 4\ 8\ 5\ 1\ 3) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\frac{n}{2} \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \frac{n}{2} + 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 94. □

Tabela 75: SbPRPTSRST – Família 11

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
10	(8 10 4 9 5 7 2 6 3 1)	6	6
11	(8 10 4 11 9 5 7 2 6 3 1)	6	7
12	(8 10 12 4 11 9 5 7 2 6 3 1)	7	7
13	(8 10 12 4 13 11 9 5 7 2 6 3 1)	?	?
14	(8 10 12 14 4 13 11 9 5 7 2 6 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(8 10 12 ... n 4 $n-1$ $n-3$ $n-5$ ... 9 5 7 2 6 3 1)	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k+1$	(8 10 12 ... $n-1$ 4 n $n-2$ $n-4$ ... 9 5 7 2 6 3 1)	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 95 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 11

```
SbPRPTSRST_Familia_11(π ← π_n, n ≥ 10)
 1  if n mod 2 = 0 then
 2      π ← π · τ_p(π_7^{-1}, π_7^{-1} + 1) · τ_p(π_{n-1}^{-1}, π_3^{-1}) · τ_s(π_2^{-1}, π_4^{-1})
 3      while π_1 ≠ 5 do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(5) · ρ_p(2)
 6  else
 7      π ← π · τ_s(π_4^{-1}, π_{n-2}^{-1})
 8      while π_3 ≠ 9 do
 9          π ← π · τ_s(π_{π_n-1}^{-1}, π_{π_n-1}^{-1} + 2)
10      π ← π · τ_s(2, 5) · τ_p(5, n) · τ_p(π_2^{-1}, π_2^{-1} + 1) · τ_p(3, 4) · ρ_s(5)
```

Lema 167. *Seja*

$$\pi_n = \begin{cases} (8\ 10\ 12\ \ldots\ n\ 4\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 9\ 5\ 7\ 2\ 6\ 3\ 1) & \textit{se } n \textit{ é par} \\ (8\ 10\ 12\ \ldots\ n{-}1\ 4\ n\ n{-}2\ n{-}4\ \ldots\ 9\ 5\ 7\ 2\ 6\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\frac{n}{2} \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \frac{n}{2} + 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 95. □

Tabela 76: SbPRPTSRST – Família 12

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
10	(9 5 10 4 6 8 2 7 3 1)	6	6
11	(11 5 10 4 6 8 2 7 9 3 1)	6	7
12	(11 5 12 4 6 8 10 2 7 9 3 1)	7	7
13	(13 5 12 4 6 8 10 2 7 9 11 3 1)	?	?
14	(13 5 14 4 6 8 10 12 2 7 9 11 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 96 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 12

```
SbPRPTSRST_Familia_12(π ← π_n, n ≥ 10)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(3, π_2^{-1}) τ_p(π_8^{-1}, π_5^{-1})
 3      while  π_1 ≠ n − 2  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · τ_p(π_6^{-1}, π_3^{-1}) · τ_s(π_2^{-1}, π_4^{-1}) · ρ_p(6) · ρ_p(2)
 6  else
 7      π ← π · τ_s(π_2^{-1}, π_{n-2}^{-1})
 8      while  π_n ≠ 7  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_p(3, 4) · τ_p(π_6^{-1}, π_3^{-1}) · τ_s(2, π_5^{-1}) · ρ_p(6) · ρ_p(2)
```

Lema 168. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 5\ n\ 4\ 6\ 8\ \ldots\ n{-}2\ 2\ 7\ 9\ 11\ \ldots\ n{-}3\ 3\ 1) & \text{se } n \text{ é par} \\ (n\ 5\ n{-}1\ 4\ 6\ 8\ \ldots\ n{-}3\ 2\ 7\ 9\ 11\ \ldots\ n{-}2\ 3\ 1) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\frac{n}{2} \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \frac{n}{2} + 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 96. □

Tabela 77: SbPRPTSRST – Família 13

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
8	(7 4 8 3 5 2 6 1)	5	5
9	(9 7 4 8 3 5 2 6 1)	6	6
10	(9 7 4 8 10 3 5 2 6 1)	6	6
11	(11 9 7 4 8 10 3 5 2 6 1)	7	7
12	(11 9 7 4 8 10 12 3 5 2 6 1)	7	7
13	(13 11 9 7 4 8 10 12 3 5 2 6 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 7\ 4\ 8\ 10\ 12\ \ldots\ n\ 3\ 5\ 2\ 6\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ n{-}2\ n{-}4\ \ldots\ 7\ 4\ 8\ 10\ 12\ \ldots\ n{-}1\ 3\ 5\ 2\ 6\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 97 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 13

```
SbPRPTSRST_Familia_13(π ← π_n, n ≥ 8)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ 7  do
3          π ← π · τ_p(2, π^{-1}_{π_1+1})
4      π ← π · τ_p(π_5^{-1}, n) · τ_s(2,5) · τ_s(3, π_3^{-1}) · ρ_p(5) · ρ_p(2)
5  else
6      π ← π · τ_p(π_8^{-1}, π_3^{-1}) · τ_p(π^{-1}_{n-2}, π_4^{-1})
7      while  π_1 ≠ 7  do
8          π ← π · τ_p(2, π^{-1}_{π_1+1})
9      π ← π · τ_p(n − 3, n) · τ_s(2, n − 2) · ρ_p(5) · ρ_p(2)
```

Lema 169. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 7\ 4\ 8\ 10\ 12\ \ldots\ n\ 3\ 5\ 2\ 6\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 7\ 4\ 8\ 10\ 12\ \ldots\ n{-}1\ 3\ 5\ 2\ 6\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-1}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 97. □

Tabela 78: SbPRPTSRST – Família 14

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
10	(8 2 7 4 10 6 1 3 9 5)	6	6
11	(8 2 7 4 10 6 1 3 9 11 5)	6	7
12	(10 8 2 7 4 12 6 1 3 9 11 5)	7	7
13	(10 8 2 7 4 12 6 1 3 9 11 13 5)	?	?
14	(12 10 8 2 7 4 14 6 1 3 9 11 13 5)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}2\ n{-}4\ n{-}6\ \ldots\ 8\ 2\ 7\ 4\ n\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n{-}1\ 5)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n{-}3\ n{-}5\ n{-}7\ \ldots\ 8\ 2\ 7\ 4\ n{-}1\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n\ 5)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 98 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 14

```
SbPRPTSRST_Familia_14(π ← π_n, n ≥ 10)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_n^{-1}, n) · ρ_s(3)
 3      while  π_1 ≠ 10  do
 4          π ← π · τ_p(π_{π_1-2}^{-1}, π_{π_1-2}^{-1} + 2)
 5      π ← π · τ_p(π_7^{-1}, π_3^{-1})τ_s(2, π_6^{-1}) · ρ_p(7) · ρ_p(2)
 6  else
 7      π ← π · τ_s(π_6^{-1}, π_n^{-1}) · τ_s(π_{n-1}^{-1}, π_5^{-1})
 8      while  π_1 ≠ 8  do
 9          π ← π · τ_p(2, π_{π_1+1}^{-1})
10      π ← π · τ_p(3,7) · τ_p(2,5) · τ_p(6,9) · ρ_p(2)
```

Lema 170. *Seja*

$$\pi_n = \begin{cases} (n{-}2\ n{-}4\ n{-}6\ \ldots\ 8\ 2\ 7\ 4\ n\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n{-}1\ 5) & \textit{se } n \textit{ é par} \\ (n{-}3\ n{-}5\ n{-}7\ \ldots\ 8\ 2\ 7\ 4\ n{-}1\ 6\ 1\ 3\ 9\ 11\ 13\ \ldots\ n\ 5) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\frac{n}{2} \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \frac{n}{2} + 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 98. □

Tabela 79: SbPRPTSRST – Família 15

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
6	(5 3 6 2 4 1)	4	4
7	(7 5 3 6 2 4 1)	5	5
8	(7 5 3 6 8 2 4 1)	5	5
9	(9 7 5 3 6 8 2 4 1)	6	6
10	(9 7 5 3 6 8 10 2 4 1)	6	6
11	(11 9 7 5 3 6 8 10 2 4 1)	7	7
12	(11 9 7 5 3 6 8 10 12 2 4 1)	7	7
13	(13 11 9 7 5 3 6 8 10 12 2 4 1)	?	?
14	(13 11 9 7 5 3 6 8 10 12 14 2 4 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 3\ \ 6\ \ 8\ \ 10\ \ \ldots\ \ n\ \ 2\ \ 4\ \ 1)$	$\leq \left\lceil \frac{n}{2} \right\rceil + 1$	?
$2k{+}1$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 3\ \ 6\ \ 8\ \ 10\ \ \ldots\ \ n{-}1\ \ 2\ \ 4\ \ 1)$	$\leq \left\lceil \frac{n}{2} \right\rceil + 1$	?

Algoritmo 99 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 15

```
SbPRPTSRST_Familia_15(π ← π_n, n ≥ 6)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ 5  do
3          π ← π · τ_p(2, π^{-1}_{π_1+1})
4      π ← π · τ_p(n − 1, n) · τ_p(3, 4) · τ_p(n − 1, n + 1) · ρ_p(2)
5  else
6      π ← π · τ_p(π_3^{-1}, π_4^{-1}) · τ_s(π_2^{-1}, π_n^{-1} − 2) · τ_p(2, π_1^{-1})
7      while  π_1 ≠ n − 1  do
8          π ← π · τ_p(2, π^{-1}_{π_1−1})
9      π ← π · ρ_p(n − 1) · ρ_p(2)
```

Lema 171. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ 6\ 8\ 10\ \ldots\ n\ 2\ 4\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 3\ 6\ 8\ 10\ \ldots\ n{-}1\ 2\ 4\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\left\lceil \frac{n-1}{2} \right\rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \left\lceil \frac{n}{2} \right\rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 99. □

Tabela 80: SbPRPTSRST – Família 16

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(9 6 8 3 7 4 2 5 1)	6	6
10	(9 6 8 10 3 7 4 2 5 1)	6	6
11	(11 6 8 10 3 9 7 4 2 5 1)	7	7
12	(11 6 8 10 12 3 9 7 4 2 5 1)	7	7
13	(13 6 8 10 12 3 11 9 7 4 2 5 1)	?	?
14	(13 6 8 10 12 14 3 11 9 7 4 2 5 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 6\ 8\ 10\ \ldots\ n\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 7\ 4\ 2\ 5\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 6\ 8\ 10\ \ldots\ n{-}1\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 7\ 4\ 2\ 5\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 100 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 16

```
SbPRPTSRST_Familia_16(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(2, π_n^{-1}) τ_p(π_3^{-1} + 1, n) · τ_s(π_2^{-1}, π_3^{-1})
 3      while  π_1 ≠ 4  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(4) · ρ_p(2)
 6  else
 7      π ← π · τ_s(3, π_7^{-1})
 8      while  π_n ≠ 3  do
 9          π ← π · τ_s(π_{π_n+1}^{-1}, n)
10      π ← π · τ_p(π_2^{-1}, n) · τ_s(π_1^{-1}, π_6^{-1}) · τ_p(5, 8) · ρ_p(4) · ρ_p(2)
```

Lema 172. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 6\ 8\ 10\ \ldots\ n\ 3\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 7\ 4\ 2\ 5\ 1) & \textit{se } n \textit{ é par} \\ (n\ 6\ 8\ 10\ \ldots\ n{-}1\ 3\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 7\ 4\ 2\ 5\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-1}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 100. □

Tabela 81: SbPRPTSRST – Família 17

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
4	(4 1 3 2)	2	2
5	(5 1 4 2 3)	2	3
6	(6 1 5 2 4 3)	3	4
7	(7 1 6 2 5 3 4)	3	5
8	(8 1 7 2 6 3 5 4)	4	5
9	(9 1 8 2 7 3 6 4 5)	4	6
10	(10 1 9 2 8 3 7 4 6 5)	5	6
11	(11 1 10 2 9 3 8 4 7 5 6)	5	7
12	(12 1 11 2 10 3 9 4 8 5 7 6)	6	7
13	(13 1 12 2 11 3 10 4 9 5 8 6 7)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2})$	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	$(n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil \frac{n}{2} \rceil{+}1\ \lceil \frac{n}{2} \rceil{-}1\ \lceil \frac{n}{2} \rceil)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 101 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 17

```
SbPRPTSRST_Familia_17(π ← π_n, n ≥ 4)
1  if  n mod 2 = 0  then
2      for  i ← n/2 − 1 downto 1  do
3          π ← π · τ_s(π_i^{-1}, π_i^{-1} + 1)
4      π ← π · ρ_s(1)
5  else
6      for  i ← n − 1 downto ⌈n/2⌉ + 1  do
7          π ← π · τ_p(π_i^{-1}, π_i^{-1} + 1)
8      π ← π · τ_p(π_1^{-1}, n + 1)
```

Lema 173. *Seja*

$$\pi_n = \begin{cases} (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \frac{n}{2}{+}1\ \frac{n}{2}) & \text{se } n \text{ é par} \\ (n\ 1\ n{-}1\ 2\ n{-}2\ 3\ \ldots\ \lceil \frac{n}{2} \rceil{+}1\ \lceil \frac{n}{2} \rceil{-}1\ \lceil \frac{n}{2} \rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, tal limitante inferior não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre. No entanto, isso só é possível até que a permutação esteja nas formas $(i\ i{+}1\ i{+}2\ \ldots\ n\ 1\ 2\ 3\ \ldots\ i{-}1)$, $(n\ n{-}1\ n{-}2\ \ldots\ i\ 1\ 2\ 3\ \ldots\ i{-}1)$ ou η_n, das quais só se pode remover zero ou um *breakpoint*. Portanto, a distância deve ser pelo menos $\lceil \frac{n}{2} \rceil - 1 + 1 = \frac{n}{2}$. Além disso, o Algoritmo 101 ordena π_n com $\frac{n}{2}$ operações quando n é par e $\lceil \frac{n}{2} \rceil - 1$ operações quando n é ímpar. □

Tabela 82: SbPRPTSRST – Família 18

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(8 9 6 7 4 5 2 3 1)	3	6
10	(9 10 7 8 5 6 3 4 1 2)	3	6
11	(10 11 8 9 6 7 4 5 2 3 1)	4	7
12	(11 12 9 10 7 8 5 6 3 4 1 2)	4	7
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ \ n\ \ n{-}3\ \ n{-}2\ \ n{-}5\ \ n{-}4\ \ \ldots\ \ 3\ \ 4\ \ 1\ \ 2)$	$\leq \lceil \frac{n}{2} \rceil - 2$	?
$2k{+}1$	$(n{-}1\ \ n\ \ n{-}3\ \ n{-}2\ \ n{-}5\ \ n{-}4\ \ \ldots\ \ 4\ \ 5\ \ 2\ \ 3\ \ 1)$	$\leq \lceil \frac{n}{2} \rceil - 2$	?

Algoritmo 102 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 18

```
SbPRPTSRST_Familia_18(π ← π_n, n ≥ 9)
1  π ← π · τ_p(5, 9) · τ_s(3, 7) · τ_p(5, n − 1)
2  while π_1 ≠ 1 do
3        π ← π · τ_p(3, π^{-1}_{π_2+1})
```

Lema 174. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n\ n{-}3\ n{-}2\ n{-}5\ n{-}4\ \ldots\ 3\ 4\ 1\ 2) & \text{se } n \text{ é par} \\ (n{-}1\ n\ n{-}3\ n{-}2\ n{-}5\ n{-}4\ \ldots\ 4\ 5\ 2\ 3\ 1) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-2}{4} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil - 2$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 102. □

Tabela 83: SbPRPTSRST – Família 19

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
5	(1 5 2 4 3)	2	3
6	(1 6 2 5 3 4)	2	4
7	(1 7 2 6 3 5 4)	3	5
8	(1 8 2 7 3 6 4 5)	3	5
9	(1 9 2 8 3 7 4 6 5)	4	6
10	(1 10 2 9 3 8 4 7 5 6)	4	6
11	(1 11 2 10 3 9 4 8 5 7 6)	5	7
12	(1 12 2 11 3 10 4 9 5 8 6 7)	5	7
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?
$2k{+}1$	$(1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil \frac{n}{2} \rceil {-}1\ \lceil \frac{n}{2} \rceil {+}1\ \lceil \frac{n}{2} \rceil)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 103 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 19

```
SbPRPTSRST_Familia_19(π ← π_n, n ≥ 5)
1  if n mod 2 = 0 then
2      for i ← n/2 + 2 to n do
3          π ← π · τ_s(π_i^{-1}, π_i^{-1} + 1)
4  else
5      for i ← ⌈n/2⌉ − 1 downto 2 do
6          π ← π · τ_s(π_i^{-1}, π_i^{-1} + 1)
7      π ← π · ρ_s(2)
```

Lema 175. *Seja*

$$\pi_n = \begin{cases} (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \frac{n}{2}\ \frac{n}{2}{+}1) & \text{se } n \text{ é par} \\ (1\ n\ 2\ n{-}1\ 3\ n{-}2\ \ldots\ \lceil \frac{n}{2} \rceil {-}1\ \lceil \frac{n}{2} \rceil {+}1\ \lceil \frac{n}{2} \rceil) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$.

Demonstração. Note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n - 2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Além disso, o Algoritmo 103 ordena π_n em $\lceil \frac{n}{2} \rceil - 1$ operações. □

Tabela 84: SbPRPTSRST – Família 20

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
5	(2 3 5 4 1)	2	3
6	(2 3 5 6 4 1)	2	4
7	(2 3 5 7 6 4 1)	3	5
8	(2 3 5 7 8 6 4 1)	3	5
9	(2 3 5 7 9 8 6 4 1)	4	6
10	(2 3 5 7 9 10 8 6 4 1)	4	6
11	(2 3 5 7 9 11 10 8 6 4 1)	5	7
12	(2 3 5 7 9 11 12 10 8 6 4 1)	5	7
13	(2 3 5 7 9 11 13 12 10 8 6 4 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 3 5 7 ... $n{-}1$ n $n{-}2$ $n{-}4$ ... 4 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?
$2k{+}1$	(2 3 5 7 ... n $n{-}1$ $n{-}3$ $n{-}5$ ... 4 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 104 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 20

```
SbPRPTSRST_Familia_20(π ← π_n, n ≥ 5)
1  π ← π · τ_p(π_n^{-1} + 1, n + 1)
2  while  π_1 ≠ 1  do
3        π ← π · τ_p(2, π_{π_1+1}^{-1})
```

Lema 176. *Seja*

$$\pi_n = \begin{cases} (2\ 3\ 5\ 7\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ 4\ 1) & \textit{se } n \textit{ é par} \\ (2\ 3\ 5\ 7\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$.

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-3$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil - 1 = \lceil \frac{n}{2} \rceil - 1$. Quando n é ímpar, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância deve ser pelo menos $\lceil \frac{n-1}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil - 1$. Além disso, o Algoritmo 104 ordena π_n com $\lceil \frac{n}{2} \rceil - 1$ operações sempre. □

Tabela 85: SbPRPTSRST – Família 21

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
8	(7 5 1 2 3 8 6 4)	3	5
9	(9 7 5 1 2 3 8 6 4)	4	6
10	(9 7 5 1 2 3 10 8 6 4)	4	6
11	(11 9 7 5 1 2 3 10 8 6 4)	5	7
12	(11 9 7 5 1 2 3 12 10 8 6 4)	5	7
13	(13 11 9 7 5 1 2 3 12 10 8 6 4)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 5\ 1\ 2\ 3\ n\ n{-}2\ n{-}4\ \ldots\ 4)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?
$2k{+}1$	$(n\ n{-}2\ n{-}4\ \ldots\ 5\ 1\ 2\ 3\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 105 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 21

```
SbPRPTSRST_Familia_21(π ← π_n, n ≥ 8)
1  while  π_1 ≠ 1  do
2        π ← π · τ_p(2, π^{-1}_{π_1 − 1})
3  π ← π · ρ_s(4)
```

Lema 177. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 5\ 1\ 2\ 3\ n\ n{-}2\ n{-}4\ \ldots\ 4). & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 5\ 1\ 2\ 3\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 4) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2} - 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 2 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n{-}3$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil - 1 = \frac{n}{2} - 1$. Quando n é ímpar, $\lceil \frac{n-1}{2} \rceil - 1 = \lceil \frac{n}{2} \rceil - 2$. Além disso, o Algoritmo 105 ordena π_n com $\lceil \frac{n}{2} \rceil - 1$ operações sempre. □

Tabela 86: SbPRPTSRST – Família 22

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
7	(4 6 1 2 7 3 5)	3	5
8	(4 6 8 1 2 7 3 5)	4	5
9	(4 6 8 1 2 9 3 5 7)	4	6
10	(4 6 8 10 1 2 9 3 5 7)	5	6
11	(4 6 8 10 1 2 11 3 5 7 9)	5	7
12	(4 6 8 10 12 1 2 11 3 5 7 9)	6	7
13	(4 6 8 10 12 1 2 13 3 5 7 9 11)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(4 6 8 ... n 1 2 $n-1$ 3 5 7 ... $n-3$)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k+1$	(4 6 8 ... $n-1$ 1 2 n 3 5 7 ... $n-2$)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 106 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 22

```
SbPRPTSRST_Familia_22(π ← π_n, n ≥ 7)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ n − 2  do
3          π ← π · τ_p(2, π^{-1}_{π_1+1})
4      π ← π · τ_p(3, n + 1)
5  else
6      while  π_1 ≠ n − 1  do
7          π ← π · τ_p(2, π^{-1}_{π_1+1})
8  π ← π · τ_p(π_3^{-1}, π^{-1}_{n-2} + 1) · τ_p(π_1^{-1}, π_2^{-1} + 1)
```

Lema 178. *Seja*

$$\pi_n = \begin{cases} (4\ 6\ 8\ \ldots\ n\ 1\ 2\ n{-}1\ 3\ 5\ 7\ \ldots\ n{-}3) & \textit{se } n \textit{ é par} \\ (4\ 6\ 8\ \ldots\ n{-}1\ 1\ 2\ n\ 3\ 5\ 7\ \ldots\ n{-}2) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é par e* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Além disso, o Algoritmo 106 ordena π_n com $\lceil \frac{n}{2} \rceil$ operações quando n é par e com $\lceil \frac{n}{2} \rceil - 1$ operações quando n é ímpar. □

Tabela 87: SbPRPTSRST – Família 23

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(9 7 5 8 2 4 6 3 1)	6	6
10	(9 7 5 10 2 4 6 8 3 1)	6	6
11	(11 9 7 5 10 2 4 6 8 3 1)	7	7
12	(11 9 7 5 12 2 4 6 8 10 3 1)	7	7
13	(13 11 9 7 5 12 2 4 6 8 10 3 1)	?	?
14	(13 11 9 7 5 14 2 4 6 8 10 12 3 1)	?	?
⋮	⋮	⋮	⋮
$2k$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 5\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ n{-}2\ n{-}4\ \ldots\ 5\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 107 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 23

```
SbPRPTSRST_Familia_23(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_p(π_6^{-1}, n − 1) · τ_p(π_{n−3}^{-1}, π_n^{-1})
 3      while  π_1 ≠ 5  do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(π_2^{-1}) · ρ_s(2) · ρ_p(2)
 6  else
 7      π ← π · ρ_p(n − 2) · τ_s(π_2^{-1}, π_3^{-1}) · τ_p(π_5^{-1}, n)
 8      while  π_1 ≠ n − 2  do
 9          π ← π · τ_p(2, π_{π_1−1}^{-1})
10      π ← π · ρ_p(n − 2) · ρ_p(2)
```

Lema 179. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 5\ n\ 2\ 4\ 6\ \ldots\ n{-}2\ 3\ 1) & \textit{se } n \textit{ é par} \\ (n\ n{-}2\ n{-}4\ \ldots\ 5\ n{-}1\ 2\ 4\ 6\ \ldots\ n{-}3\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância é pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O O limitante superior é dado pelo Algoritmo 107. □

Tabela 88: SbPRPTSRST – Família 24

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(9 6 8 2 4 7 5 3 1)	6	6
10	(9 6 8 10 2 4 7 5 3 1)	6	6
11	(11 6 8 10 2 4 9 7 5 3 1)	7	7
12	(11 6 8 10 12 2 4 9 7 5 3 1)	7	7
13	(13 6 8 10 12 2 4 11 9 7 5 3 1)	?	?
14	(13 6 8 10 12 14 2 4 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n{-}1\ 6\ 8\ 10\ \ldots\ n\ 2\ 4\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 6\ 8\ 10\ \ldots\ n{-}1\ 2\ 4\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 108 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 24

```
SbPRPTSRST_Familia_24(π ← π_n, n ≥ 9)
 1  if n mod 2 = 0 then
 2      π ← π · τ_p(2, π_n^{-1}) · τ_p(π_{n-3}^{-1}, n − 1)
 3      while π_1 ≠ 5 do
 4          π ← π · τ_p(2, π_{π_1+1}^{-1})
 5      π ← π · ρ_p(n − 3) · ρ_s(2) · ρ_p(2)
 6  else
 7      π ← π · ρ_p(n − 2) · τ_s(π_2^{-1}, π_3^{-1}) · τ_p(π_{n-3}^{-1}, n) · τ_p(π_7^{-1}, π_4^{-1})
 8      while π_1 ≠ n − 2 do
 9          π ← π · τ_p(2, π_{π_1-1}^{-1})
10      π ← π · ρ_p(n − 2) · ρ_p(2)
```

Lema 180. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 6\ 8\ 10\ \ldots\ n\ 2\ 4\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (n\ 6\ 8\ 10\ \ldots\ n{-}1\ 2\ 4\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n-1}{2} \rceil$ porque $b_{\rho_p\rho_s}(\pi_n) = n-1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, $\lceil \frac{n-1}{2} \rceil = \lceil \frac{n}{2} \rceil$. Quando n é ímpar, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que já não é possível no primeiro passo. Portanto, a distância é pelo menos $\lceil \frac{n-1}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil - 1 + 1 = \lceil \frac{n}{2} \rceil$. O O limitante superior é dado pelo Algoritmo 108. □

Tabela 89: SbPRPTSRST – Família 25

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(8 4 9 7 1 3 5 2 6)	5	6
10	(10 8 4 9 7 1 3 5 2 6)	6	6
11	(10 8 4 9 11 7 1 3 5 2 6)	6	7
12	(12 10 8 4 9 11 7 1 3 5 2 6)	7	7
13	(12 10 8 4 9 11 13 7 1 3 5 2 6)	?	?
14	(14 12 10 8 4 9 11 13 7 1 3 5 2 6)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ n{-}2\ n{-}4\ \ldots\ 8\ 4\ 9\ 11\ 13\ \ldots\ n{-}1\ 7\ 1\ 3\ 5\ 2\ 6)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n{-}1\ n{-}3\ n{-}5\ \ldots\ 8\ 4\ 9\ 11\ 13\ \ldots\ n\ 7\ 1\ 3\ 5\ 2\ 6)$	$\leq \lceil \frac{n}{2} \rceil$	?

Algoritmo 109 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 25

```
SbPRPTSRST_Familia_25(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(n − 2, n − 1) · τ_s(π_4^{-1}, π_4^{-1} + 1) · τ_p(π_1^{-1}, π_2^{-1} + 1) · τ_s(2, π_8^{-1})
 3      while  π_n ≠ n  do
 4          π ← π · τ_s(π_{π_n+1}^{-1}, n)
 5      π ← π · τ_p(π_7^{-1}, n) · ρ_p(7)
 6  else
 7      π ← π · τ_p(π_4^{-1}, π_7^{-1}) · τ_p(π_3^{-1}, π_2^{-1}) · τ_s(2, π_n^{-1})
 8      while  π_n ≠ 4  do
 9          π ← π · τ_s(π_{π_{n-1}}^{-1}, n)
10      π ← π · τ_p(π_1^{-1}, π_2^{-1} + 1) · ρ_s(4)
```

Lema 181. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 8\ 4\ 9\ 11\ 13\ \ldots\ n{-}1\ 7\ 1\ 3\ 5\ 2\ 6) & \textit{se } n \textit{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 8\ 4\ 9\ 11\ 13\ \ldots\ n\ 7\ 1\ 3\ 5\ 2\ 6) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$ *quando* n *é par e* $\lceil \frac{n}{2} \rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 109. □

Tabela 90: SbPRPTSRST – Família 26

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
9	(9 4 8 5 7 2 6 3 1)	6	6
10	(9 4 8 10 5 7 2 6 3 1)	6	6
11	(11 4 8 10 5 9 7 2 6 3 1)	7	7
12	(11 4 8 10 12 5 9 7 2 6 3 1)	7	7
13	(13 4 8 10 12 5 11 9 7 2 6 3 1)	?	?
14	(13 4 8 10 12 14 5 11 9 7 2 6 3 1)	?	?
⋮	⋮	⋮	⋮
$2k$	$(n{-}1\ 4\ 8\ 10\ 12\ \ldots\ n\ 5\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 7\ 2\ 6\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k{+}1$	$(n\ 4\ 8\ 10\ 12\ \ldots\ n{-}1\ 5\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 7\ 2\ 6\ 3\ 1)$	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 110 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 26

```
SbPRPTSRST_Familia_26(π ← π_n, n ≥ 9)
 1  if  n mod 2 = 0  then
 2      π ← π · τ_s(n − 3, n − 2) · τ_s(π_n^{-1}, π_n^{-1} + 2) · τ_s(π_4^{-1}, π_{n-2}^{-1})
 3      while  π_n ≠ 4  do
 4          π ← π · τ_s(π_{π_n − 1}^{-1}, n)
 5      π ← π · τ_s(π_3^{-1}, π_5^{-1}) · ρ_p(n − 1) · ρ_p(2)
 6  else
 7      π ← π · τ_p(3, n − 1) · τ_s(π_2^{-1}, π_4^{-1}) · τ_p(π_5^{-1} + 1, π_4^{-1})
 8      while  π_1 ≠ 7  do
 9          π ← π · τ_p(2, π_{π_1 + 1}^{-1})
10      π ← π · τ_p(π_5^{-1}, n) · ρ_p(5) · ρ_p(2)
```

Lema 182. *Seja*

$$\pi_n = \begin{cases} (n{-}1\ 4\ 8\ 10\ 12\ \ldots\ n\ 5\ n{-}3\ n{-}5\ n{-}7\ \ldots\ 7\ 2\ 6\ 3\ 1) & \textit{se } n \textit{ é par} \\ (n\ 4\ 8\ 10\ 12\ \ldots\ n{-}1\ 5\ n{-}2\ n{-}4\ n{-}6\ \ldots\ 7\ 2\ 6\ 3\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-1}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 110. □

Tabela 91: SbPRPTSRST – Família 27

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
12	(11 9 12 10 2 4 6 3 7 5 8 1)	7	7
13	(11 13 9 12 10 2 4 6 3 7 5 8 1)	?	?
14	(11 13 9 14 12 10 2 4 6 3 7 5 8 1)	?	?
15	(11 13 15 9 14 12 10 2 4 6 3 7 5 8 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(11 13 15 ... $n-1$ 9 n $n-2$ $n-4$ ... 10 2 4 6 3 7 5 8 1)	$\leq \lceil \frac{n}{2} \rceil + 1$	?
$2k+1$	(11 13 15 ... n 9 $n-1$ $n-3$ $n-5$ ... 10 2 4 6 3 7 5 8 1)	$\leq \lceil \frac{n}{2} \rceil + 1$	?

Algoritmo 111 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 27

```
SbPRPTSRST_Familia_27(π ← π_n, n ≥ 12)
1  π ← π · τ_s(n − 2, n − 1)τ_s(n − 5, n − 4)
2  while π_1 ≠ 9 do
3        π ← π · τ_p(2, π^{-1}_{π_1 − 1})
4  π ← π · τ_p(π_4^{-1}, π_1^{-1}) · τ_s(π_7^{-1}, π_2^{-1}) · ρ_s(π_n^{-1}) · ρ_p(4)
```

Lema 183. *Seja*

$$\pi_n = \begin{cases} (11\ 13\ 15\ \ldots\ n{-}1\ 9\ n\ n{-}2\ n{-}4\ \ldots\ 10\ 2\ 4\ 6\ 3\ 7\ 5\ 8\ 1) & \textit{se } n \textit{ é par} \\ (11\ 13\ 15\ \ldots\ n\ 9\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 10\ 2\ 4\ 6\ 3\ 7\ 5\ 8\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-1}{2} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n}{2} \rceil + 1$.

Demonstração. O limitante inferior é válido porque $b_{\rho_p\rho_s}(\pi_n) = n - 1$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. O limitante superior é dado pelo Algoritmo 111. □

Tabela 92: SbPRPTSRST – Família 28

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
10	(2 1 4 3 6 5 8 7 10 9)	4	6
11	(2 1 4 3 6 5 8 7 11 10 9)	4	7
12	(2 1 4 3 6 5 8 7 10 9 12 11)	5	7
13	(2 1 4 3 6 5 8 7 10 9 13 12 11)	?	?
14	(2 1 4 3 6 5 8 7 10 9 12 11 14 13)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 1 4 3 6 5 ... n $n-1$)	$\leq \lceil \frac{n}{2} \rceil - 1$	?
$2k+1$	(2 1 4 3 6 5 ... n $n-1$ $n-2$)	$\leq \lceil \frac{n}{2} \rceil - 2$	?

Algoritmo 112 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 28

```
SbPRPTSRST_Familia_28(π ← π_n, n ≥ 10)
1  π ← π · τ_p(5, 9) · τ_s(3, 7) · τ_p(5, n − 1)
2  while  π_1 ≠ n  do
3        π ← π · τ_p(3, π^{-1}_{π_2 − 1})
4  π ← π · ρ_p(n)
```

Lema 184. *Seja*

$$\pi_n = \begin{cases} (2\ 1\ 4\ 3\ 6\ 5\ \ldots\ n\ n{-}1) & \text{se } n \text{ é par} \\ (2\ 1\ 4\ 3\ 6\ 5\ \ldots\ n\ n{-}1\ n{-}2) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $\lceil \frac{n-2}{4} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n-2}{2} \rceil$ *quando* n *é par e* $\lceil \frac{n-3}{4} \rceil \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \leq \lceil \frac{n-3}{2} \rceil$ *quando* n *é ímpar.*

Demonstração. Os limitantes inferiores são válidos porque $b_{\rho_p\rho_s}(\pi_n) = \lceil \frac{n-2}{2} \rceil$ quando n é par, $b_{\rho_p\rho_s}(\pi_n) = \lceil \frac{n-3}{2} \rceil$ quando n é ímpar e e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Os limitantes superiores são dados pelo Algoritmo 112. □

Tabela 93: SbPRPTSRST – Família 29

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
4	(4 2 1 3)	2	2
5	(4 2 1 3 5)	2	3
6	(6 4 2 1 3 5)	3	4
7	(6 4 2 1 3 5 7)	3	5
8	(8 6 4 2 1 3 5 7)	4	5
9	(8 6 4 2 1 3 5 7 9)	4	6
10	(10 8 6 4 2 1 3 5 7 9)	5	6
11	(10 8 6 4 2 1 3 5 7 9 11)	5	7
12	(12 10 8 6 4 2 1 3 5 7 9 11)	6	7
13	(12 10 8 6 4 2 1 3 5 7 9 11 13)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	$(n\ \ n{-}2\ \ n{-}4\ \ \ldots\ \ 2\ \ 1\ \ 3\ \ 5\ \ \ldots\ \ n{-}1)$	$\leq \lceil \frac{n}{2} \rceil$	?
$2k{+}1$	$(n{-}1\ \ n{-}3\ \ n{-}5\ \ \ldots\ \ 2\ \ 1\ \ 3\ \ 5\ \ \ldots\ \ n)$	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 113 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 29

```
SbPRPTSRST_Familia_29(π ← π_n, n ≥ 4)
1  while  π_1 ≠ 1  do
2        π ← π · τ_p(2, π^{-1}_{π_1 − 1} + 1)
```

Lema 185. *Seja*

$$\pi_n = \begin{cases} (n\ n{-}2\ n{-}4\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n{-}1) & \text{se } n \text{ é par} \\ (n{-}1\ n{-}3\ n{-}5\ \ldots\ 2\ 1\ 3\ 5\ \ldots\ n) & \text{se } n \text{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que só é possível até que a permutação esteja na forma η_n, da qual nenhum *breakpoint* pode ser removido. Portanto, a distância é pelo menos $\lceil \frac{n}{2} \rceil - 1 + 1 = \frac{n}{2}$. Além disso, o Algoritmo 113 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil \frac{n}{2} \rceil - 1$ operações quando n é ímpar. □

Tabela 94: SbPRPTSRST – Família 30

n	π	$d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$	$D_{\rho_p\tau_p\rho_s\tau_s}(n)$
4	(2 4 3 1)	2	2
5	(2 4 5 3 1)	2	3
6	(2 4 6 5 3 1)	3	4
7	(2 4 6 7 5 3 1)	3	5
8	(2 4 6 8 7 5 3 1)	4	5
9	(2 4 6 8 9 7 5 3 1)	4	6
10	(2 4 6 8 10 9 7 5 3 1)	5	6
11	(2 4 6 8 10 11 9 7 5 3 1)	5	7
12	(2 4 6 8 10 12 11 9 7 5 3 1)	6	7
13	(2 4 6 8 10 12 13 11 9 7 5 3 1)	?	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k$	(2 4 6 ... n $n-1$ $n-3$ $n-5$ ... 1)	$\leq \lceil \frac{n}{2} \rceil$	?
$2k+1$	(2 4 6 ... $n-1$ n $n-2$ $n-4$ ... 1)	$\leq \lceil \frac{n}{2} \rceil - 1$	?

Algoritmo 114 SbPRPTSRST – Pseudocódigo para ordenar permutações da família 30

```
SbPRPTSRST_Familia_30(π ← π_n, n ≥ 4)
1  if  n mod 2 = 0  then
2      while  π_1 ≠ n  do
3          π ← π · τ_p(2, π^{-1}_{π_1 − 1})
4      π ← π · ρ_p(n)
5  else
6      π ← π · τ_p(π_n^{-1} + 1, n + 1)
7      while  π_1 ≠ 1  do
8          π ← π · τ_p(2, π^{-1}_{π_1 + 1})
```

Lema 186. *Seja*

$$\pi_n = \begin{cases} (2\ 4\ 6\ \ldots\ n\ n{-}1\ n{-}3\ n{-}5\ \ldots\ 1) & \textit{se } n \textit{ é par} \\ (2\ 4\ 6\ \ldots\ n{-}1\ n\ n{-}2\ n{-}4\ \ldots\ 1) & \textit{se } n \textit{ é ímpar} \end{cases}.$$

Então, $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \frac{n}{2}$ *quando* n *é par e* $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) = \lceil \frac{n}{2} \rceil - 1$ *quando* n *é ímpar.*

Demonstração. Primeiro note que $d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n) \geq \lceil \frac{n}{2} \rceil - 1$ porque $b_{\rho_p\rho_s}(\pi_n) = n-2$ e $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \left\lceil \frac{b_{\rho_p\rho_s}(\pi)}{2} \right\rceil$ para qualquer π. Quando n é par, tal limitante não é justo pois, se fosse, seria necessário remover dois *breakpoints* sempre, o que só é possível até que a permutação esteja na forma η_n, da qual nenhum *breakpoint* pode ser removido. Portanto, a distância é pelo menos $\lceil \frac{n}{2} \rceil - 1 + 1 = \frac{n}{2}$. Além disso, o Algoritmo 114 ordena π_n com $\frac{n}{2}$ operações quando n é par e com $\lceil \frac{n}{2} \rceil - 1$ operações quando n é ímpar. □

Tabela 95: SbPRPTSRST – Resumo das famílias encontradas ($d = d_{\rho_p\tau_p\rho_s\tau_s}(\pi_n)$)

Família	$b_{\rho_p\rho_s}(\pi_n)$	d se n par	d se n ímpar
1	$n-1$	$\lceil\frac{n-1}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
2	$n-1$	$d = \frac{n}{2}$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
3	$n-1$	$\lceil\frac{n}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
4	$n-1$	$\lceil\frac{n}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
5	$n-1$	$\lceil\frac{n}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
6	$n-2$ quando n é par e $n-1$ quando n é ímpar	$d = \frac{n}{2}$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
7	$n-1$	$d = \frac{n}{2}$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
8	$n-2$	$d = \frac{n}{2}$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
9	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
10	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
11	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
12	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
13	$n-1$	$\lceil\frac{n-1}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
14	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
15	$n-1$	$\lceil\frac{n-1}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
16	$n-1$	$\lceil\frac{n-1}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
17	$n-2$	$d = \frac{n}{2}$	$d = \lceil\frac{n}{2}\rceil - 1$
18	$\lceil\frac{n}{2}\rceil - 1$	$\lceil\frac{n-2}{4}\rceil \leq d \leq \lceil\frac{n}{2}\rceil - 2$	
19	$n-2$	$d = \lceil\frac{n}{2}\rceil - 1$	
20	$n-3$	$d = \lceil\frac{n}{2}\rceil - 1$	
21	$n-3$	$d = \frac{n}{2} - 1$	$\lceil\frac{n}{2}\rceil - 2 \leq d \leq \lceil\frac{n}{2}\rceil - 1$
22	$n-3$	$\frac{n}{2} - 1 \leq d \leq \frac{n}{2}$	$d = \lceil\frac{n}{2}\rceil - 1$
23	$n-1$	$\lceil\frac{n}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
24	$n-1$	$\lceil\frac{n}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
25	$n-1$	$\frac{n}{2} \leq d \leq \frac{n}{2} + 1$	$\lceil\frac{n}{2}\rceil - 1 \leq d \leq \lceil\frac{n}{2}\rceil$
26	$n-1$	$\lceil\frac{n-1}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
27	$n-1$	$\lceil\frac{n-1}{2}\rceil \leq d \leq \lceil\frac{n}{2}\rceil + 1$	
28	$\lceil\frac{n-2}{2}\rceil$ quando n é par e $\lceil\frac{n-3}{2}\rceil$ quando n é ímpar	$\frac{n-2}{4} \leq d \leq \frac{n-2}{2}$	$\lceil\frac{n-3}{4}\rceil \leq d \leq \lceil\frac{n-3}{2}\rceil$
29	$n-2$	$d = \frac{n}{2}$	$d = \lceil\frac{n}{2}\rceil - 1$
30	$n-2$	$d = \frac{n}{2}$	$d = \lceil\frac{n}{2}\rceil - 1$

D Classes de Permutações

Esta seção apresenta algumas classes de permutações para alguns dos problemas estudados bem como algoritmos simples que podem ordená-las.

D.1 Classes para SbPRPT

Lema 187. *Seja $\pi \cdot \rho(i,j) = \iota_n$. Então,*

$$d_{\rho_p\tau_p}(\pi) = \begin{cases} 1 & \text{se } i = 1 \\ 2 & \text{se } i = 2 \text{ ou } j - i = 1 \\ 3 & \text{caso contrário} \end{cases}.$$

Demonstração. Quando $i = 1$, $\rho_p(j)$ basta. Quando $i = 2$ então $\tau_p(3, j+1) \cdot \rho_p(j-1)$ basta. Quando $j - i = 1$, $\tau_p(j, j+1) \cdot \tau_p(2, j)$ basta. Caso contrário, $\tau_p(i+1, j+1) \cdot \rho_p(j-1) \cdot \rho_p(i-1)$ ordena π. □

Lema 188. *Seja $\pi \cdot \tau(i,j,k) = \iota_n$. Então,*

$$d_{\rho_p\tau_p}(\pi) = \begin{cases} 1 & \text{se } i = 1 \\ 2 & \text{caso contrário} \end{cases}.$$

Demonstração. Quando $i = 1$, $\tau_p(j,k)$ basta. Caso contrário, $\tau_p(i+k-j, k) \cdot \tau_p(j-i+1, j)$ ordena π. □

Lema 189. *Seja $\pi \cdot \rho(i,j) = \eta_n$. Então,*

$$d_{\rho_p\tau_p}(\pi) = \begin{cases} 2 & \text{se } i = 1 \text{ ou } j = n \\ 3 & \text{caso contrário} \end{cases}.$$

Demonstração. Quando $i = 1$ ou $j = n$ então $\rho_p(j) \cdot \rho_p(n - i + 1)$ basta. Caso contrário, $\tau_p(j+1, n+1) \cdot \rho_p(i - 1 + n - j) \cdot \tau_p(i, n+1)$ ordena π. □

Lema 190. *Seja $\pi \cdot \rho(i,j) \cdot \rho(k,l) = \iota_n$, com $i < j < k < l$. Então,*

$$d_{\rho_p\tau_p}(\pi) = \begin{cases} 2 & \text{se } i = 1 \text{ e } k - j = 1 \\ 3 & \text{se } i = 2 \text{ e } k - j = 1 \\ 4 & \text{caso contrário} \end{cases}.$$

Demonstração. Quando $i = 1$ e $k - j = 1$ então $\tau_p(k+1, l+1) \cdot \rho_p(l-1)$ basta. Quando $i = 2$ e $k - j = 1$ então $\tau_p(2, k) \cdot \tau_p(k+1, l+1) \cdot \rho_p(l-1)$ basta. Caso contrário, $\tau_p(j+1, l+1) \cdot \rho_p(l - j + i - 1) \cdot \tau_p(l - k + i, l + 1) \cdot \rho_p(k - 1)$ ordena π. □

Lema 191. *Seja $\pi \cdot \tau(i, j, k) \cdot \rho(l, m) = \iota_n$ com $i \leq l < m \leq k$. Então,*

$$d_{\rho_p\tau_p}(\pi) \leq 4.$$

Demonstração. A sequência $\tau_p(i, k) \cdot \rho_p(k - i + 1) \cdot \tau_p(k - j + 1, k) \cdot \tau_p(j - i + 1, j)$ consegue ordenar π sempre. □

Lema 192. *Seja $\pi = (\text{números pares}, \text{números ímpares em ordem})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \left\lceil \frac{n-1}{2} \right\rceil = \begin{cases} \left\lceil \frac{n}{2} \right\rceil & \text{se } n \text{ é par} \\ \\ \left\lceil \frac{n}{2} \right\rceil - 1 & \text{se } n \text{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n - 2 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que coloca cada um dos pares no devido lugar com transposições de prefixo. □

Lema 193. *Seja $\pi = (\text{números pares em ordem}, \text{números ímpares})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \left\lceil \frac{n}{2} \right\rceil + 1.$$

Demonstração. O limitante inferior se dá pelo fato de que $n - 2 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que traz os ímpares para o começo da permutação com uma transposição de prefixo e depois coloca cada um no devido lugar também com transposições de prefixo. □

Lema 194. *Seja $\pi = (\text{números pares}, \text{números ímpares})$. Então.*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \begin{cases} 2\left\lceil \frac{n-1}{2} \right\rceil - 1 & = 2\frac{n}{2} - 1 \quad = n - 1 \quad \text{se } n \text{ é par} \\ \\ 2\left\lceil \frac{n-1}{2} \right\rceil & = 2\frac{n-1}{2} \quad = n - 1 \quad \text{se } n \text{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n - 2 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que move cada um dos $\left\lceil \frac{n-1}{2} \right\rceil$ pares i para depois de $i - 1$ e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento.

Note que se n é par, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2. Se n é ímpar, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2 mais uma de tamanho 1. □

Lema 195. *Seja $\pi = (alguns\ pares,\ ímpares\ em\ ordem,\ outros\ pares)$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \left\lceil \frac{n-1}{2} \right\rceil + 1.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um simples algoritmo que traz os pares que estão no final para o começo da permutação e então coloca cada um dos números pares no devido lugar com transposições de prefixo. □

Lema 196. *Seja $\pi = (alguns\ pares,\ ímpares,\ outros\ pares)$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \begin{cases} 2\left\lceil \frac{n-1}{2} \right\rceil & = 2\frac{n}{2} & = n \quad se\ n\ é\ par \\ \\ 2\left\lceil \frac{n-1}{2} \right\rceil + 1 & = 2\frac{n-1}{2} + 1 & = n \quad se\ n\ é\ ímpar \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que move os números pares que estão no final para o começo da permutação, move cada um dos $\left\lceil \frac{n-1}{2} \right\rceil$ pares i para depois de $i-1$ e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que se n é par, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2. Se n é ímpar, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2 mais uma de tamanho 1. □

Lema 197. *Seja $\pi = (números\ ímpares,\ números\ pares\ em\ ordem)$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \left\lceil \frac{n}{2} \right\rceil.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que coloca cada um dos números ímpares no devido lugar com transposições de prefixo. □

Lema 198. *Seja $\pi = (números\ ímpares\ em\ ordem,\ números\ pares)$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \left\lceil \frac{n-1}{2} \right\rceil + 1 = \begin{cases} \left\lceil \frac{n}{2} \right\rceil + 1 & se\ n\ é\ par \\ \\ \left\lceil \frac{n}{2} \right\rceil & se\ n\ é\ ímpar \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que traz os números pares para o começo da permutação com uma transposição de prefixo e depois coloca cada um no devido lugar também com transposições de prefixo. □

Lema 199. *Seja $\pi = (\text{números ímpares}, \text{números pares})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq 2\left\lceil \frac{n}{2} \right\rceil - 1 = \begin{cases} 2\frac{n}{2} - 1 & = n-1 \quad \text{se } n \text{ é par} \\ \\ 2\frac{n+1}{2} - 1 & = n \qquad \text{se } n \text{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que move cada um dos $\left\lceil \frac{n}{2} \right\rceil$ ímpares i para antes de $i+1$ e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que são no máximo $\left\lceil \frac{n}{2} \right\rceil$ *strips* de tamanho 2 sendo que uma delas tem tamanho 1 se n for ímpar. □

Lema 200. *Seja $\pi = (\text{alguns ímpares}, \text{números pares em ordem}, \text{outros ímpares})$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq \left\lceil \frac{n}{2} \right\rceil + 1.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que traz os números ímpares que estão no final para o começo da permutação e então coloca cada um dos números ímpares no devido lugar com transposições de prefixo. □

Lema 201. *Seja $\pi = (\text{alguns ímpares}, \text{números pares}, \text{outros ímpares})$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq 2\left\lceil \frac{n}{2} \right\rceil = \begin{cases} 2\frac{n}{2} & = n \qquad \text{se } n \text{ é par} \\ \\ 2\frac{n+1}{2} & = n+1 \quad \text{se } n \text{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um algoritmo simples que move os números ímpares que estão no final para o começo da permutação, move cada um dos $\left\lceil \frac{n}{2} \right\rceil$ ímpares i para antes de $i+1$ e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que são no máximo $\left\lceil \frac{n}{2} \right\rceil$ *strips* de tamanho 2 sendo que uma delas tem tamanho 1 se n for ímpar. □

Lema 202. *Seja $\pi = \pi' = (par\ ímpar\ par\ ímpar\ \ldots\ par\ ímpar)$ ou $\pi = \pi'' = (ímpar\ par\ ímpar\ par\ \ldots\ ímpar\ par)$ permutações de Z_n com n par. Então*

$$\begin{array}{rcl} d_{\rho_p\tau_p}(\pi) & \leq & n-1 \\ d_{\rho_p\tau_p}(\pi') & \geq & 1 \\ d_{\rho_p\tau_p}(\pi'') & \geq & 0 \end{array} .$$

Demonstração. O limitante inferior de π' se dá pelo fato de que esta pode ser a permutação reversa. O limitante inferior de π'' se dá porque esta pode ser a permutação identidade. O limitante superior se dá por um simples algoritmo move cada elemento para o lugar certo, do maior para o menor. Note que os dois últimos elementos são ordenados com um único movimento. □

Lema 203. *Seja $\pi = \pi' = (par\ ímpar\ par\ ímpar\ \ldots\ par\ ímpar\ ímpar\ \ldots\ par\ ímpar)$ ou $\pi = \pi'' = (ímpar\ par\ ímpar\ par\ \ldots\ ímpar\ par\ ímpar)$, ambas com n ímpar. Então*

$$\begin{array}{rcl} d_{\rho_p\tau_p}(\pi) & \leq & n-1 \\ d_{\rho_p\tau_p}(\pi') & \geq & 1 \\ d_{\rho_p\tau_p}(\pi'') & \geq & 0 \end{array} .$$

Demonstração. O limitante inferior de π' se dá pelo fato de que esta pode ser formada por duas *strips* crescentes. O limitante inferior de π'' se dá por que esta pode ser a permutação identidade. O limitante superior se dá por um simples algoritmo move cada elemento para o lugar certo, do maior para o menor. Note que os dois últimos elementos são ordenados com um único movimento. □

Lema 204. *Seja $\pi = (múltiplos\ de\ 3\ em\ ordem,\ não\ múltiplos\ de\ 3)$. Então,*

$$\left\lceil \frac{n}{3} \right\rceil - 1 \leq d_{\rho_p\tau_p}(\pi) \leq 2 \left\lceil \frac{n-2}{3} \right\rceil + 3.$$

Demonstração. O limitante inferior se dá pelo fato de que $2\left\lceil \frac{n-2}{3} \right\rceil - 1 \leq b_{\rho_p}(\pi) \leq n$. O limitante superior se dá por um simples algoritmo que traz os não múltiplos de 3, que são no máximo $2\left\lceil \frac{n-2}{3} \right\rceil + 2$, para o começo da permutação e coloca cada um no lugar certo com transposições de prefixo. □

Definição 24. *Um **frame** de tamanho k é uma subsequência $[\pi_i, \ldots, \pi_{i+k-1}]$ de uma permutação π na qual os elementos pertencentes possuem valores que variam entre i e $i+k-1$, não necessariamente nos locais corretos.*

Lema 205. *Seja π uma permutação de Z_n dividida em $\left\lceil \frac{n}{k} \right\rceil$ frames com $k \geq 1$ elementos cada,*

exceto pelo último se n não é múltiplo de k. Então,

$$\begin{aligned} d_{\rho_p\tau_p}(\pi) &\leq (k-1)\left\lceil \tfrac{n}{k}\right\rceil + \left\lceil \tfrac{n}{k}\right\rceil = k\left\lceil \tfrac{n}{k}\right\rceil \\ d_{\rho_p\tau_p}(\pi) &\leq D_{\rho_p\tau_p}(k)\left\lceil \tfrac{n}{k}\right\rceil + \left\lceil \tfrac{n}{k}\right\rceil = (D_{\rho_p\tau_p}(k)+1)\left\lceil \tfrac{n}{k}\right\rceil \\ d_{\rho_p\tau_p}(\pi) &\geq 0 \end{aligned}$$

Demonstração. O limitante inferior se dá pelo fato de que π é a permutação identidade quando $k = 1$. O primeiro limitante superior se dá por um simples algoritmo que ordena o primeiro *frame* (o que leva no máximo $k-1$ passos), move tal *frame* para o final e faz isso para todos os *frames* seguintes. O segundo limitante superior apenas se utiliza do fato de que ordenar um *frame* leva no máximo $D_{\rho_p\tau_p}(k)$ passos, valor que sabemos para $k \leq 13$. □

Lema 206. *Seja* $\pi = (\underbrace{p_1 \ \ldots \ 1}_{l_1} \ \underbrace{p_2 \ \ldots \ p_1+1}_{l_2} \ \ldots\ldots \ \underbrace{n \ \ldots \ p_{b_{\rho_p}(\pi)-1}+1)}_{l_{b_{\rho_p}(\pi)}}$ *com* $b_{\rho_p}(\pi) \geq 1$ *e* $l_i \geq 1$ *para* $1 \leq i \leq b_{\rho_p}(\pi)$. *Então*

$$1 \leq d_{\rho_p\tau_p}(\pi) \leq b_{\rho_p}(\pi).$$

Demonstração. O limitante inferior se dá pelo fato de que π pode ser a permutação reversa. O limitante superior se dá pelo fato de que a sequência $\tau_p\left(l_1+1, n+1\right) \cdot \tau_p\left(l_2+1, n-l_1+1\right) \cdot \tau_p\left(l_3+1, n-(l_1+l_2)+1\right) \cdot \ldots \cdot \tau_p\left(l_{b_{\rho_p}(\pi)-1}+1, n-(\sum_{i=1}^{b_{\rho_p}(\pi)-2} l_i)+1\right) \cdot \rho_p(n)$ ordena π. □

D.2 Classes para SbPRPTSRST

Lema 207. *Seja* $\pi \cdot \rho(i,j) = \iota_n$. *Então,*

$$d_{\rho_p\tau_p\rho_s\tau_s}(\pi) = \begin{cases} 1 & \text{se } i = 1 \text{ ou } j = n \\ 2 & \text{se } i = 2 \text{ ou } j - i = 1 \text{ ou } j = n-1 \\ 3 & \text{caso contrário} \end{cases}.$$

Demonstração. Quando $i = 1$, $\rho_p(j)$ basta. Quando $j = n$, $\rho_s(i)$ basta. Quando $i = 2$ então $\tau_p(3, j+1) \cdot \rho_p(j-1)$ basta. Quando $j - i = 1$, $\tau_p(j, j+1) \cdot \tau_p(2, j)$ basta. Quando $j = n-1$, $\tau_s(i, n-1) \cdot \rho_s(i+1)$ basta. Caso contrário, $\tau_p(i+1, j+1) \cdot \rho_p(j-1) \cdot \rho_p(i-1)$ ordena π. □

Lema 208. *Seja* $\pi \cdot \tau(i,j,k) = \iota_n$. *Então,*

$$d_{\rho_p\tau_p\rho_s\tau_s}(\pi) = \begin{cases} 1 & \text{se } i = 1 \text{ ou } k = n+1 \\ 2 & \text{caso contrário} \end{cases}$$

Demonstração. Quando $i = 1$, $\tau_p(j,k)$ basta. Quando $k = n+1$, $\tau_s(i,j)$ basta. Caso contrário, $\tau_p(i+k-j,k) \cdot \tau_p(j-i+1,j)$ ordena π. □

Lema 209. *Seja $\pi \cdot \rho(i,j) = \eta_n$. Então.*

$$d_{\rho_p\tau_p\rho_s\tau_s}(\pi) = \begin{cases} 2 & \text{se } i = 1 \text{ ou } j = n \\ 3 & \text{caso contrário} \end{cases}$$

Demonstração. Quando $i = 1$ ou $j = n$ então $\rho_p(j) \cdot \rho_p(n-i+1)$ basta. Caso contrário, $\tau_p(j+1,n+1) \cdot \rho_p(i-1+n-j) \cdot \tau_p(i,n+1)$ ordena π. □

Lema 210. *Seja $\pi \cdot \rho(i,j) \cdot \rho(k,l) = \iota_n$, com $i < j < k < l$. Então,*

$$d_{\rho_p\tau_p\rho_s\tau_s}(\pi) = \begin{cases} 2 & \text{se } i = 1 \text{ e } k-j = 1 \\ 3 & \text{se } i = 2 \text{ e } k-j = 1 \\ 4 & \text{caso contrário} \end{cases}.$$

Demonstração. Quando $i = 1$ e $k-j = 1$ então $\tau_p(k+1,l+1) \cdot \rho_p(l-1)$ basta. Quando $i = 2$ e $k-j = 1$ então $\tau_p(2,k) \cdot \tau_p(k+1,l+1) \cdot \rho_p(l-1)$ basta. Caso contrário, $\tau_p(j+1,l+1) \cdot \rho_p(l-j+i-1) \cdot \tau_p(l-k+i,l+1) \cdot \rho_p(k-1)$ ordena π. □

Lema 211. *Seja $\pi \cdot \tau(i,j,k) \cdot \rho(l,m) = \iota_n$ com $i \leq l < m \leq k$. Então,*

$$d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq 4.$$

Demonstração. A sequência $\tau_p(i,k) \cdot \rho_p(k-i+1) \cdot \tau_p(k-j+1,k) \cdot \tau_p(j-i+1,j)$ consegue ordenar π sempre. □

Lema 212. *Seja $\pi = (\text{números pares}, \text{números ímpares em ordem})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \left\lceil \frac{n-1}{2} \right\rceil = \begin{cases} \left\lceil \frac{n}{2} \right\rceil = \frac{n}{2} & \text{se } n \text{ é par} \\ \left\lceil \frac{n}{2} \right\rceil - 1 = \frac{n-1}{2} & \text{se } n \text{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que coloca cada um dos pares no devido lugar com transposições de prefixo. □

Lema 213. *Seja $\pi = (\text{números pares em ordem}, \text{números ímpares})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \left\lceil \frac{n}{2} \right\rceil.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que coloca cada um dos ímpares no devido lugar com transposições de sufixo. □

Lema 214. *Seja $\pi = (números\ pares,\ números\ ímpares)$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \begin{cases} 2\left\lceil \frac{n-1}{2} \right\rceil - 1 & = 2\frac{n}{2} - 1 \quad = n-1 \quad \textit{se } n \textit{ é par} \\ \\ 2\left\lceil \frac{n-1}{2} \right\rceil & = 2\frac{n-1}{2} \quad = n-1 \quad \textit{se } n \textit{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que move cada um dos $\left\lceil \frac{n-1}{2} \right\rceil$ pares i para depois de $i-1$ e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que se n é par, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2. Se n é ímpar, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2 mais uma de tamanho 1. □

Lema 215. *Seja $\pi = (alguns\ pares,\ ímpares\ em\ ordem,\ outros\ pares)$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \left\lceil \frac{n-1}{2} \right\rceil.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um simples algoritmo move cada um dos pares que estão no final para o lugar certo com transposições de sufixo e move cada um dos pares que estão no começo da permutação para o lugar certo com transposições de prefixo. □

Lema 216. *Seja $\pi = (alguns\ pares,\ ímpares,\ outros\ pares)$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq \begin{cases} 2\left\lceil \frac{n-1}{2} \right\rceil - 1 & = 2\frac{n}{2} - 1 \quad = n-1 \quad \textit{se } n \textit{ é par} \\ \\ 2\left\lceil \frac{n-1}{2} \right\rceil & = 2\frac{n-1}{2} \quad = n-1 \quad \textit{se } n \textit{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que move cada um dos $\left\lceil \frac{n-1}{2} \right\rceil$ pares i para depois de $i-1$ com transposições de prefixo e sufixo depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que se n é par, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2. Se n é ímpar, então existem no máximo $\left\lceil \frac{n-1}{2} \right\rceil$ *strips* de tamanho 2 mais uma de tamanho 1. □

Lema 217. *Seja $\pi = (\text{números ímpares}, \text{números pares em ordem})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p \tau_p \rho_s \tau_s}(\pi) \leq \left\lceil \frac{n}{2} \right\rceil .$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p \rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que coloca cada um dos números ímpares no devido lugar com transposições de prefixo. □

Lema 218. *Seja $\pi = (\text{números ímpares em ordem}, \text{números pares})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p \tau_p \rho_s \tau_s}(\pi) \leq \left\lceil \frac{n-1}{2} \right\rceil = \begin{cases} \left\lceil \frac{n}{2} \right\rceil & \text{se } n \text{ é par} \\ \\ \left\lceil \frac{n}{2} \right\rceil - 1 & \text{se } n \text{ é ímpar} \end{cases} .$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p \rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que depois coloca cada um dos números pares no devido lugar com transposições de sufixo. □

Lema 219. *Seja $\pi = (\text{números ímpares}, \text{números pares})$. Então,*

$$\left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1 \leq d_{\rho_p \tau_p \rho_s \tau_s}(\pi) \leq 2\left\lceil \frac{n}{2} \right\rceil - 1 = \begin{cases} 2\frac{n}{2} - 1 = n-1 & \text{se } n \text{ é par} \\ \\ 2\frac{n+1}{2} - 1 = n & \text{se } n \text{ é ímpar} \end{cases} .$$

Demonstração. O limitante inferior se dá pelo fato de que $n-2 \leq b_{\rho_p \rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que move cada um dos $\left\lceil \frac{n}{2} \right\rceil$ ímpares i para antes de $i+1$ e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que são no máximo $\left\lceil \frac{n}{2} \right\rceil$ *strips* de tamanho 2 sendo que uma delas tem tamanho 1 se n for ímpar. □

Lema 220. *Seja $\pi = (\text{alguns ímpares}, \text{números pares em ordem}, \text{outros ímpares})$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p \tau_p \rho_s \tau_s}(\pi) \leq \left\lceil \frac{n}{2} \right\rceil .$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p \rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que coloca cada um dos números ímpares no devido lugar com transposições de prefixo e sufixo. □

Lema 221. *Seja $\pi = (\text{alguns ímpares, números pares, outros ímpares})$. Então,*

$$\left\lceil \frac{n-3}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq 2\left\lceil \frac{n}{2} \right\rceil - 1 = \begin{cases} 2\frac{n}{2} - 1 & = n-1 \quad \textit{se } n \textit{ é par} \\ \\ 2\frac{n+1}{2} - 1 & = n \qquad \textit{se } n \textit{ é ímpar} \end{cases}.$$

Demonstração. O limitante inferior se dá pelo fato de que $n-3 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um algoritmo simples que move cada um dos $\left\lceil \frac{n}{2} \right\rceil$ ímpares i para antes de $i+1$ com transposições de prefixo e sufixo e depois move cada *strip* crescente para o lugar correto com transposições de prefixo da maior para a menor. Nesse caso, as duas últimas *strips* são ordenadas com um único movimento. Note que são no máximo $\left\lceil \frac{n}{2} \right\rceil$ *strips* de tamanho 2 sendo que uma delas tem tamanho 1 se n for ímpar. □

Lema 222. *Seja $\pi = \pi' = (\text{par ímpar par ímpar } \ldots \text{ par ímpar})$ ou $\pi = \pi'' = (\text{ímpar par ímpar par } \ldots \text{ ímpar par})$ permutações de Z_n com n par. Então*

$$\begin{aligned} d_{\rho_p\tau_p\rho_s\tau_s}(\pi) &\leq n-1 \\ d_{\rho_p\tau_p\rho_s\tau_s}(\pi') &\geq 1 \\ d_{\rho_p\tau_p\rho_s\tau_s}(\pi'') &\geq 0 \end{aligned}.$$

Demonstração. O limitante inferior de π' se dá pelo fato de que esta pode ser a permutação reversa. O limitante inferior de π'' se dá porque esta pode ser a permutação identidade. O limitante superior se dá por um simples algoritmo move cada elemento para o lugar certo, do maior para o menor. Note que os dois últimos elementos são ordenados com um único movimento. □

Lema 223. *Seja $\pi = \pi' = (\text{par ímpar par ímpar } \ldots \text{ par ímpar ímpar } \ldots \text{ par ímpar})$ ou $\pi = \pi'' = (\text{ímpar par ímpar par } \ldots \text{ ímpar par ímpar})$, ambas com n ímpar. Então,*

$$\begin{aligned} d_{\rho_p\tau_p\rho_s\tau_s}(\pi) &\leq n-1 \\ d_{\rho_p\tau_p\rho_s\tau_s}(\pi') &\geq 1 \\ d_{\rho_p\tau_p\rho_s\tau_s}(\pi'') &\geq 0 \end{aligned}.$$

Demonstração. O limitante inferior de π' se dá pelo fato de que esta pode ser formada por duas *strips* crescentes. O limitante inferior de π'' se dá por que esta pode ser a permutação identidade. O limitante superior se dá por um simples algoritmo move cada elemento para o lugar certo, do maior para o menor. Note que os dois últimos elementos são ordenados com um único movimento. □

Lema 224. *Seja $\pi = ($múltiplos de 3 em ordem, não múltiplos de 3$)$. Então,*

$$\left\lceil \frac{n}{3} \right\rceil - 1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq 2 \left\lceil \frac{n-2}{3} \right\rceil + 2.$$

Demonstração. O limitante inferior se dá pelo fato de que $2\left\lceil \frac{n-2}{3} \right\rceil - 1 \leq b_{\rho_p\rho_s}(\pi) \leq n-1$. O limitante superior se dá por um simples algoritmo que coloca cada um dos não múltiplos de 3, que são no máximo $2\left\lceil \frac{n-2}{3} \right\rceil + 2$, no lugar certo com transposições de sufixo. □

Lema 225. *Seja π uma permutação de Z_n dividida em $\left\lceil \frac{n}{k} \right\rceil$ frames com $k \geq 1$ elementos cada, exceto pelo último se n não é múltiplo de k. Então,*

$$\begin{aligned}
d_{\rho_p\tau_p\rho_s\tau_s}(\pi) &\leq (k-1)\left\lceil \tfrac{n}{k} \right\rceil + \left\lceil \tfrac{n}{k} \right\rceil = k\left\lceil \tfrac{n}{k} \right\rceil \\
d_{\rho_p\tau_p\rho_s\tau_s}(\pi) &\leq d_{\rho_p\tau_p\rho_s\tau_s}(k)\left\lceil \tfrac{n}{k} \right\rceil + \left\lceil \tfrac{n}{k} \right\rceil = (d_{\rho_p\tau_p\rho_s\tau_s}(k)+1)\left\lceil \tfrac{n}{k} \right\rceil \\
d_{\rho_p\tau_p\rho_s\tau_s}(\pi) &\geq 0
\end{aligned}$$

Demonstração. O limitante inferior se dá pelo fato de que π é a permutação identidade quando $k = 1$. O primeiro limitante superior se dá por um simples algoritmo que ordena o primeiro *frame* (o que leva no máximo $k-1$ passos), move tal *frame* para o final e faz isso para todos os *frames* seguintes. O segundo limitante superior apenas se utiliza do fato de que ordenar um *frame* leva no máximo $d_{\rho_p\tau_p\rho_s\tau_s}(k)$ passos, valor que sabemos para $k \leq 13$. □

Lema 226. *Seja* $\pi = (\underbrace{p_1 \ \ldots \ 1}_{l_1} \ \underbrace{p_2 \ \ldots \ p_1+1}_{l_2} \ \ldots\ldots \ \underbrace{n \ \ldots \ p_{b_{\rho_p\rho_s}(\pi)}+1)}_{l_{b_{\rho_p\rho_s}(\pi)+1}}$ *com* $b_{\rho_p\rho_s}(\pi) \geq 1$ *e* $l_i \geq 1$ *para* $1 \leq i \leq b_{\rho_p\rho_s}(\pi)+1$. *Entãom*

$$1 \leq d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq b_{\rho_p\rho_s}(\pi) + 1.$$

Demonstração. O limitante inferior se dá pelo fato de que π pode ser a permutação reversa. O limitante superior se dá pelo fato de que a sequência $\tau_p\left(l_1+1, n+1\right) \cdot \tau_p\left(l_2+1, n-l_1+1\right) \cdot \tau_p\left(l_3+1, n-(l_1+l_2)+1\right) \cdot \ldots \cdot \tau_p\left(l_{b_{\rho_p\rho_s}(\pi)}+1, n-(\sum_{i=1}^{b_{\rho_p\rho_s}(\pi)-1} l_i)+1\right) \cdot \rho_p(n)$ ordena π. □