MC-102 — Algoritmos de ordenação

Lehilton Pedrosa

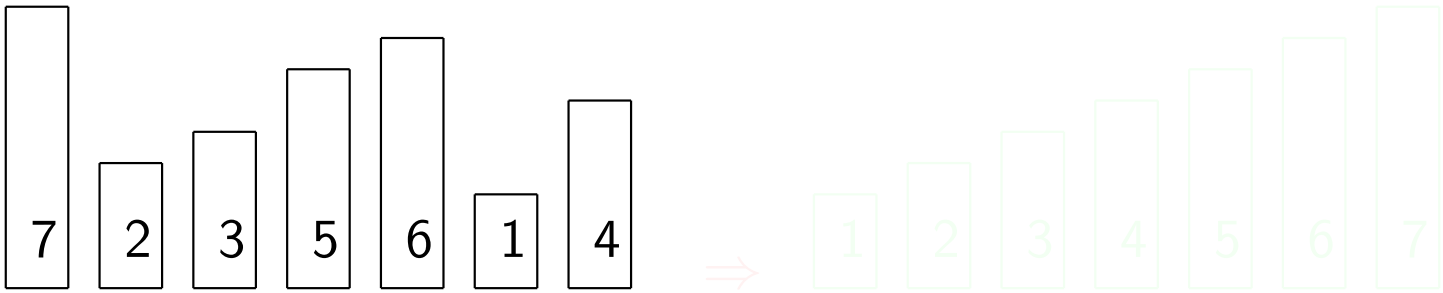
Instituto de Computação – Unicamp

Segundo Semestre de 2012

Roteiro

1. Introdução
2. Ordenação por seleção
3. Ordenação por inserção
4. Ordenação por Intercalação
5. Divisão e conquista
6. Ordenação por Particionamento
7. Divisão e conquista novamente

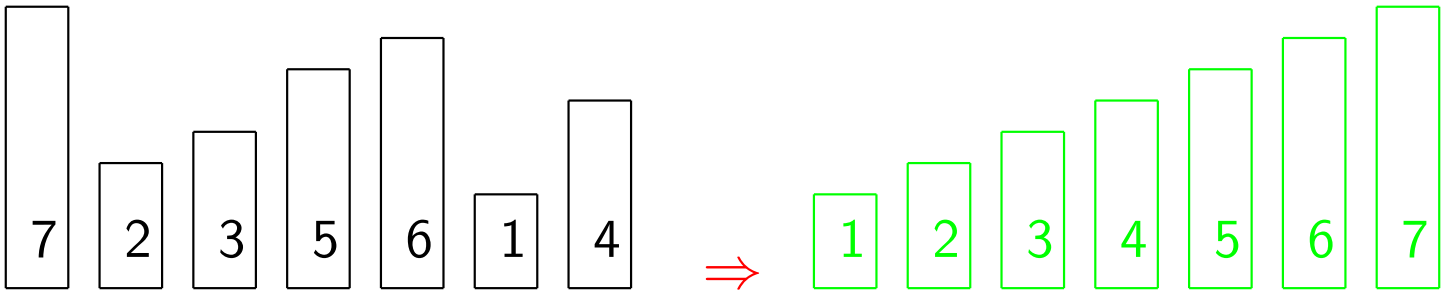
Introdução



Problema

Escreva um programa que recebe uma lista de números inteiros e imprima-os em ordem crescente.

Introdução



Problema

Escreva um programa que recebe uma lista de números inteiros e imprima-os em ordem crescente.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

Ordenação

Ordenação

- Vamos estudar algoritmos para ordenar conjuntos de elementos
- Os elementos podem ser de qualquer tipo que possamos comparar:
  - números inteiros,
  - nomes de pessoas,
  - times de futebol... :)
- Os algoritmos podem ordenar crescente ou decrescentemente, dependendo da direção da comparação.

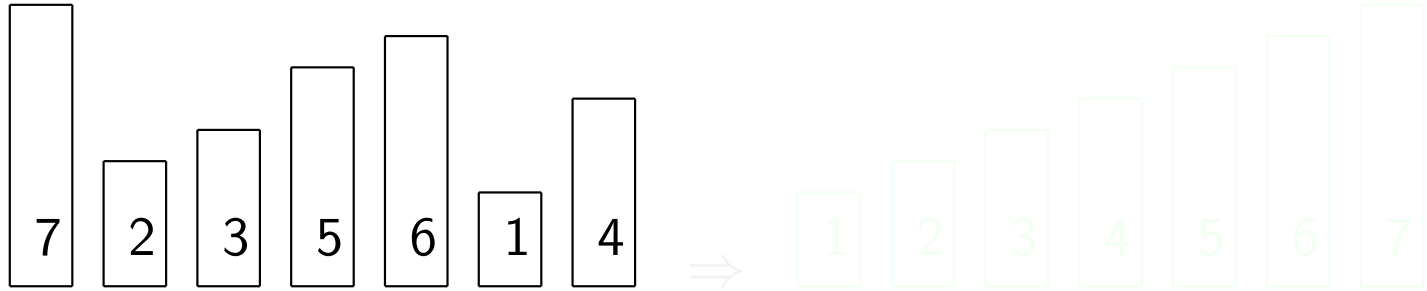
Estratégias

Existem várias estratégias para ordenar:

- Selecionar o menor a cada vez e colocar na ponta
- Trocar itens fora de ordem
- Outras?

Estratégias diferentes levam a algoritmos diferentes.

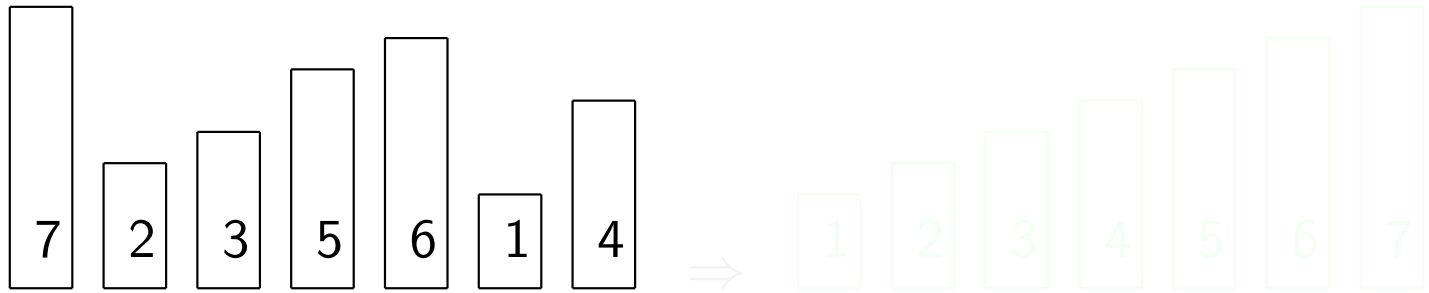
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

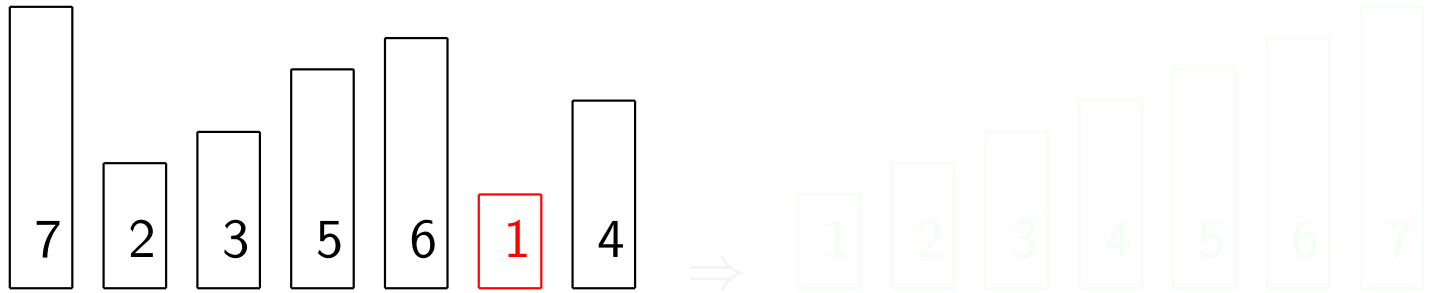
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

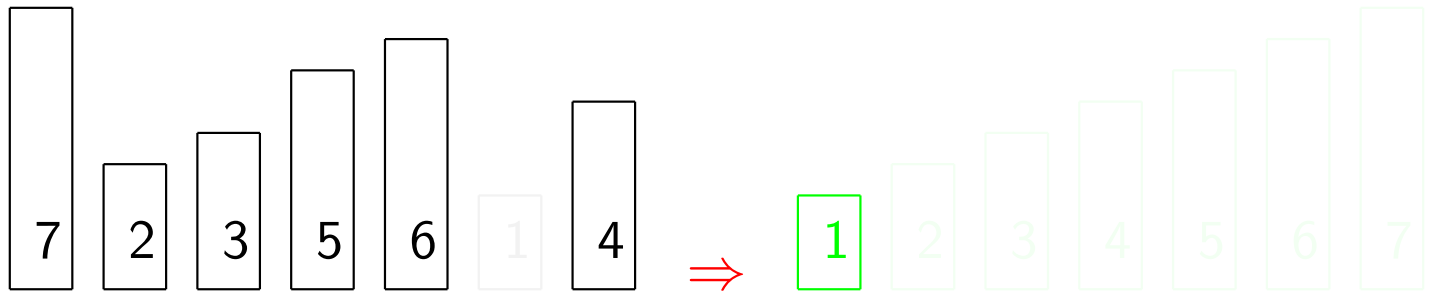
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

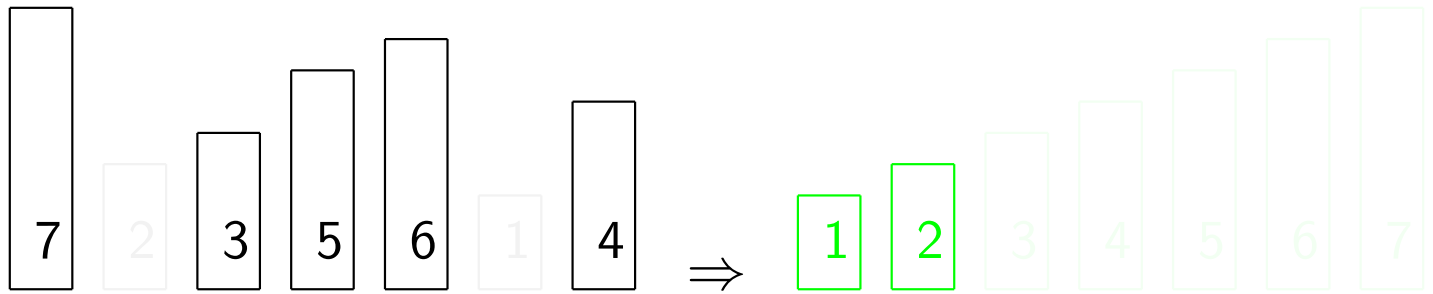
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

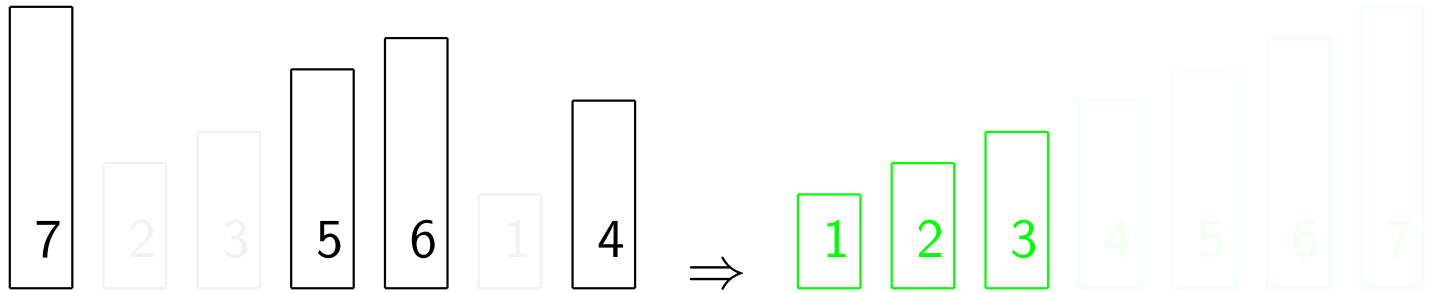
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

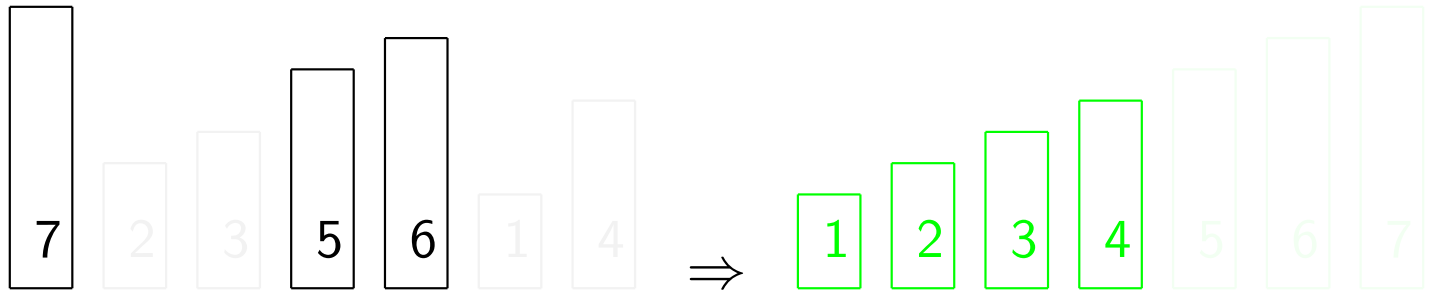
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

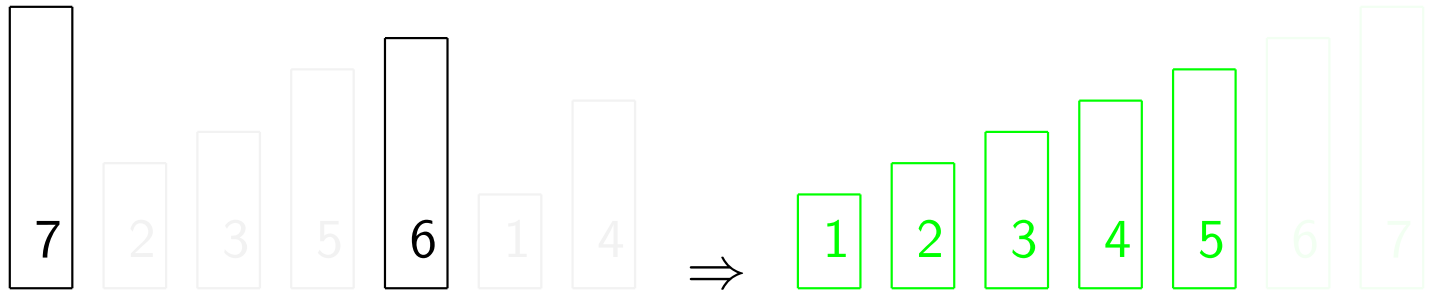
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

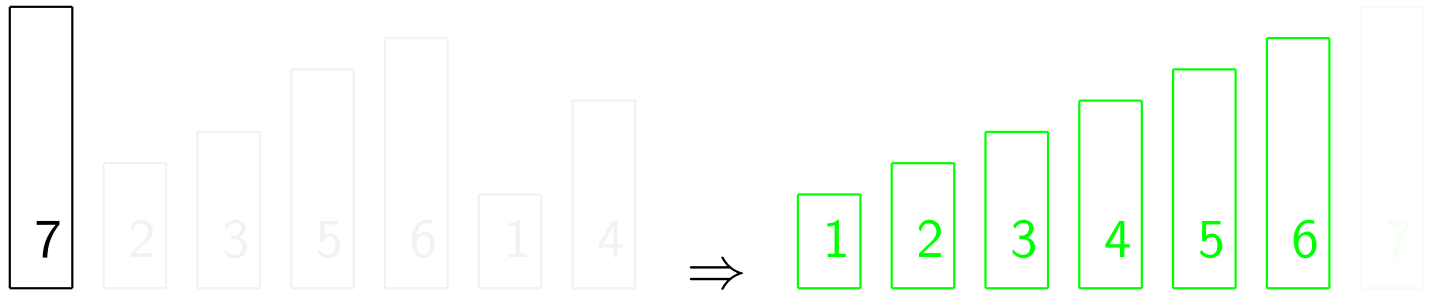
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

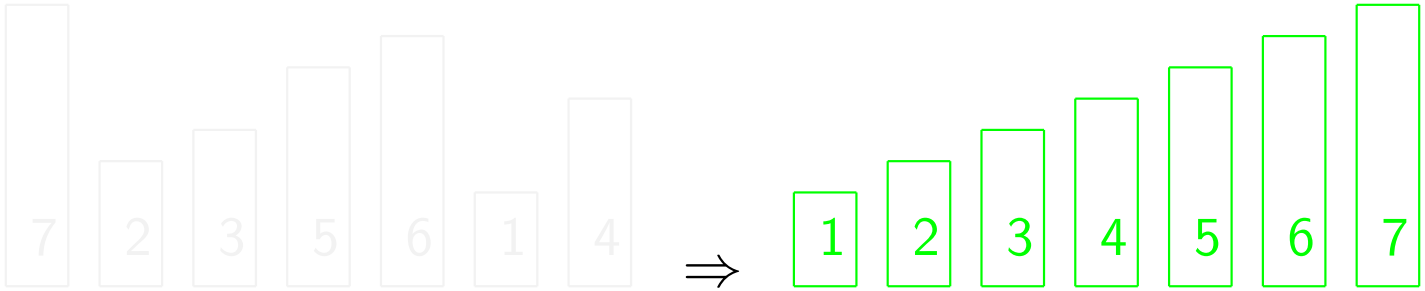
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

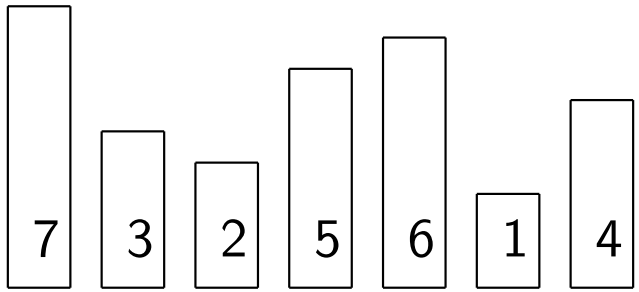
Ordenação por seleção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Selecionamos o menor elemento
- Movemos o item para uma nova lista
- Repetimos tudo com a lista restante (preta)

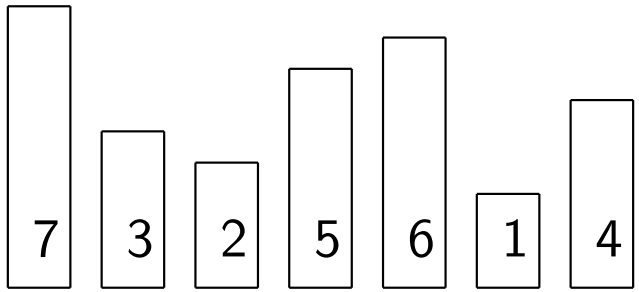
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

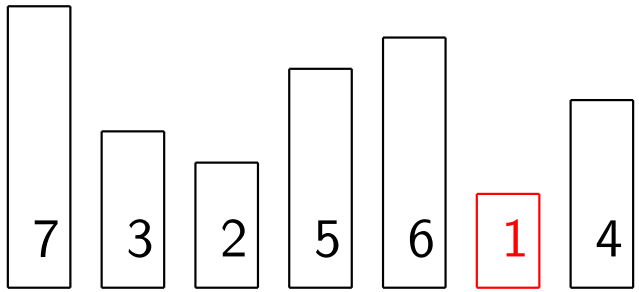
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

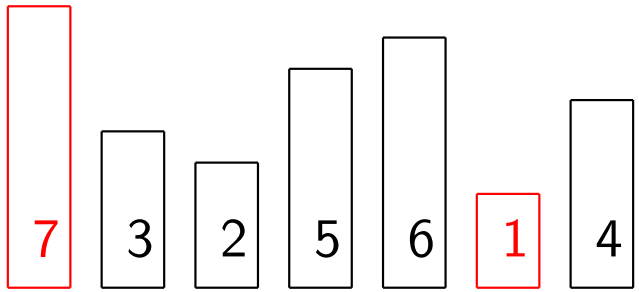
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

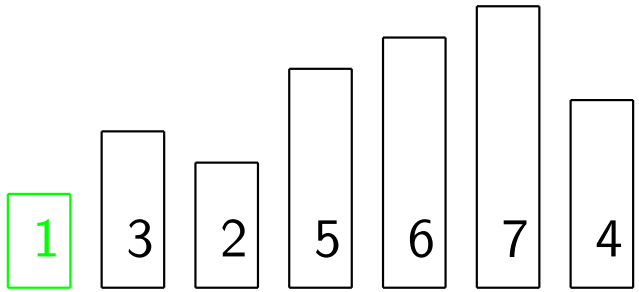
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

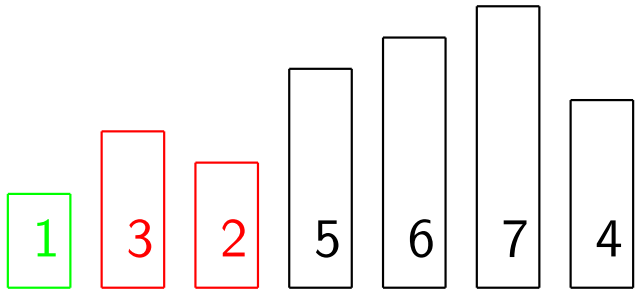
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

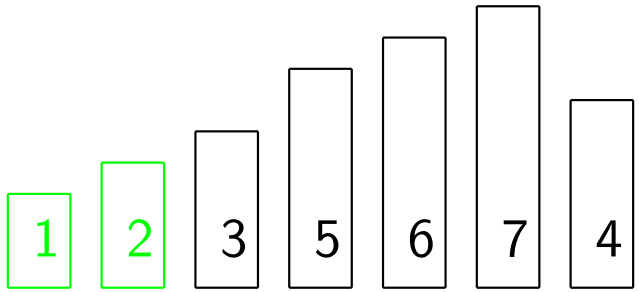
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

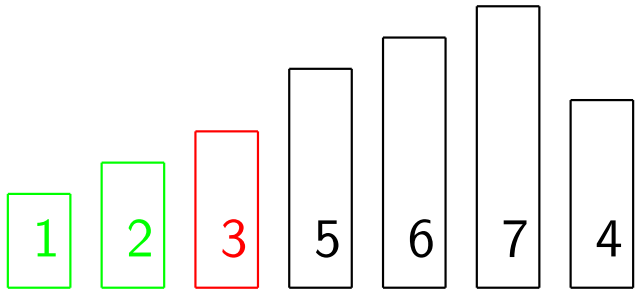
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

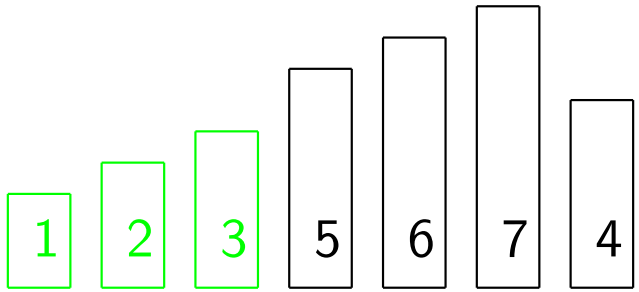
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

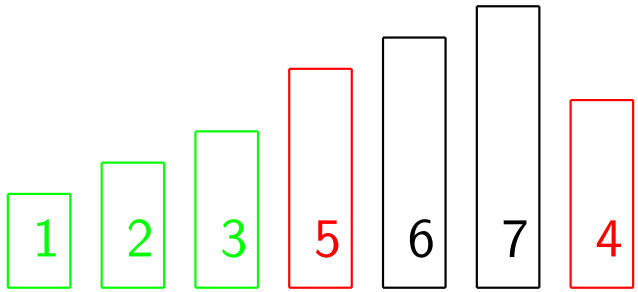
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

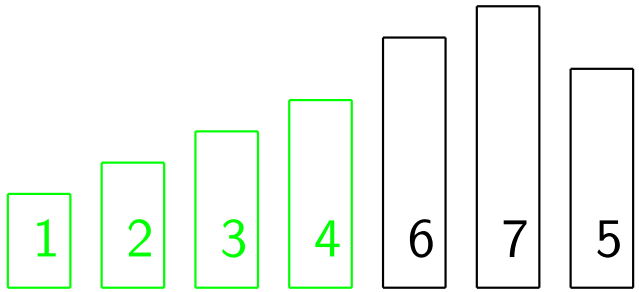
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

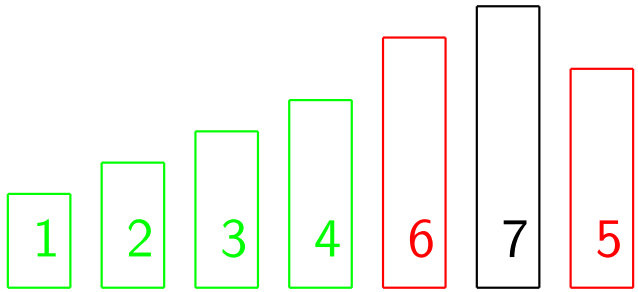
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

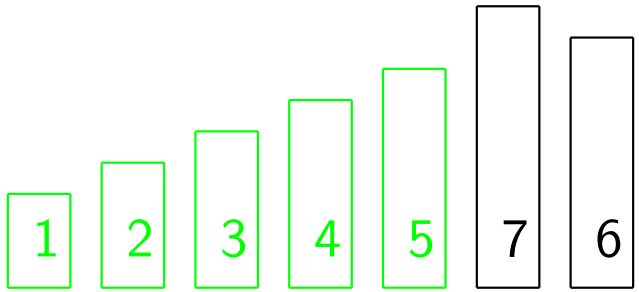
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

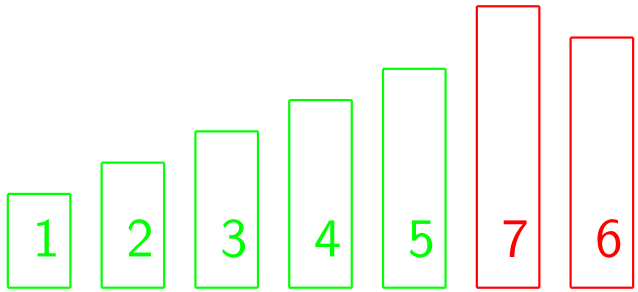
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

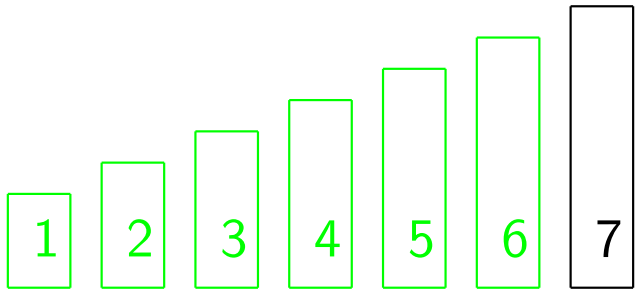
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

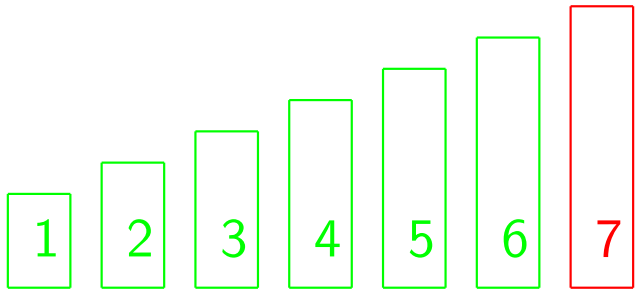
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

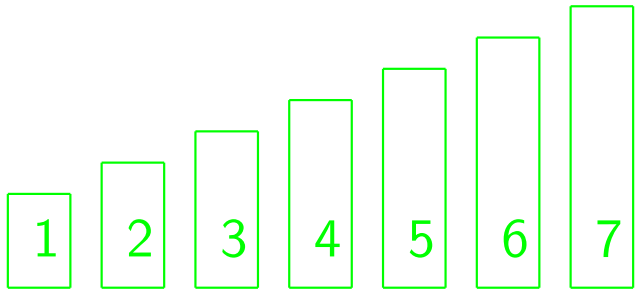
Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

Ordenação por seleção - Melhorando



Ideia

- Não precisamos de uma nova lista! Basta:
  1. Selecionar o menor elemento
  2. Trocar com o primeiro elemento da lista
  3. Continuar com a lista restante (preta)

Convenções

Algoritmo de ordenação

- Vamos implementar uma função para ordenar inteiros
- A função terá os seguintes parâmetros:
  - `int vetor[]`: vetor de inteiros onde os elementos estão
  - `int n`: o número de elemento do vetor
- A função deverá ordenar o vetor passado crescentemente

Para trocar dois valores, vamos sempre usar a seguinte função:

Trocar dois valores inteiros

```
void trocar(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Convenções

Algoritmo de ordenação

- Vamos implementar uma função para ordenar inteiros
- A função terá os seguintes parâmetros:
  - `int vetor[]`: vetor de inteiros onde os elementos estão
  - `int n`: o número de elemento do vetor
- A função deverá ordenar o vetor passado crescentemente

Para trocar dois valores, vamos sempre usar a seguinte função:

Trocar dois valores inteiros

```
void trocar(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Convenções

Algoritmo de ordenação

- Vamos implementar uma função para ordenar inteiros
- A função terá os seguintes parâmetros:
  - `int vetor[]`: vetor de inteiros onde os elementos estão
  - `int n`: o número de elemento do vetor
- A função deverá ordenar o vetor passado crescentemente

Para trocar dois valores, vamos sempre usar a seguinte função:

Trocar dois valores inteiros

```
void trocar(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Convenções

Algoritmo de ordenação

- Vamos implementar uma função para ordenar inteiros
- A função terá os seguintes parâmetros:
  - `int vetor[]`: vetor de inteiros onde os elementos estão
  - `int n`: o número de elemento do vetor
- A função deverá ordenar o vetor passado crescentemente

Para trocar dois valores, vamos sempre usar a seguinte função:

Trocar dois valores inteiros

```
void trocar(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Convenções

Algoritmo de ordenação

- Vamos implementar uma função para ordenar inteiros
- A função terá os seguintes parâmetros:
  - `int vetor[]`: vetor de inteiros onde os elementos estão
  - `int n`: o número de elemento do vetor
- A função deverá ordenar o vetor passado crescentemente

Para trocar dois valores, vamos sempre usar a seguinte função:

Trocar dois valores inteiros

```
void trocar(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Convenções

Algoritmo de ordenação

- Vamos implementar uma função para ordenar inteiros
- A função terá os seguintes parâmetros:
  - `int vetor[]`: vetor de inteiros onde os elementos estão
  - `int n`: o número de elemento do vetor
- A função deverá ordenar o vetor passado crescentemente

Para trocar dois valores, vamos sempre usar a seguinte função:

Trocar dois valores inteiros

```
void trocar(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Algoritmo de ordenação por seleção (*Selection-Sort*)

Menor elemento não ordenado (na lista preta)

```
int menor_elemento(int vetor[], int n, int primeiro) {
    int i, menor = primeiro;
    for (i = primeiro + 1; i < n; i++) {
        if (vetor[i] < vetor[menor])
            menor = i;
    }
    return menor;
}
```

Ordenação por seleção

```
int ordenar_selecao(int vetor[], int n) {
    int i, menor;
    for (i = 0; i < n; i++) {
        menor = menor_elemento(vetor, n, i);
        trocar(&vetor[i], &vetor[menor]);
    }
}
```

Algoritmo de ordenação por seleção (*Selection-Sort*)

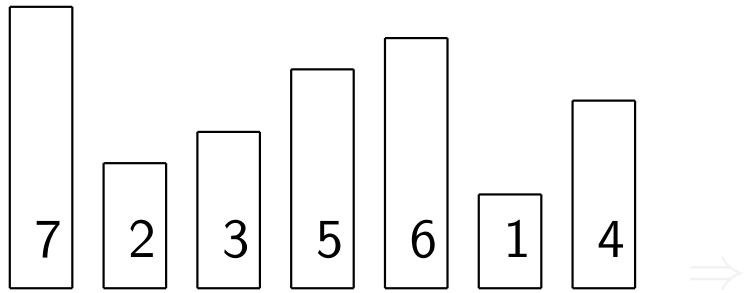
Menor elemento não ordenado (na lista preta)

```
int menor_elemento(int vetor[], int n, int primeiro) {
    int i, menor = primeiro;
    for (i = primeiro + 1; i < n; i++) {
        if (vetor[i] < vetor[menor])
            menor = i;
    }
    return menor;
}
```

Ordenação por seleção

```
int ordenar_selecao(int vetor[], int n) {
    int i, menor;
    for (i = 0; i < n; i++) {
        menor = menor_elemento(vetor, n, i);
        trocar(&vetor[i], &vetor[menor]);
    }
}
```

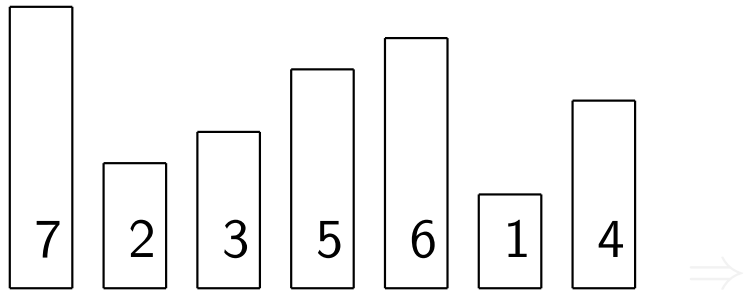
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

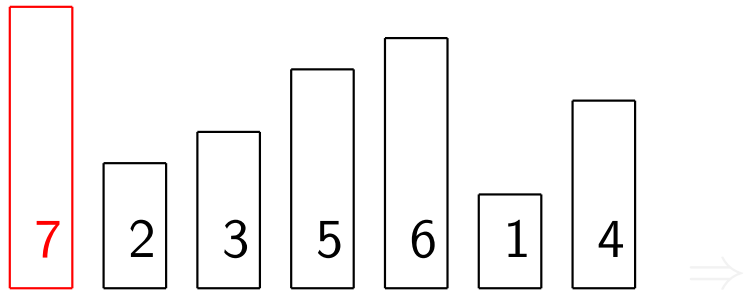
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

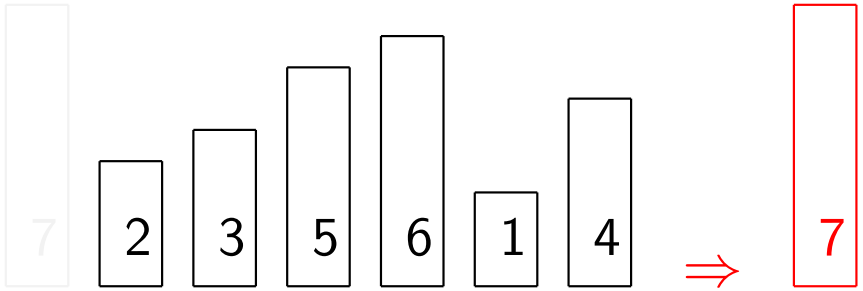
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

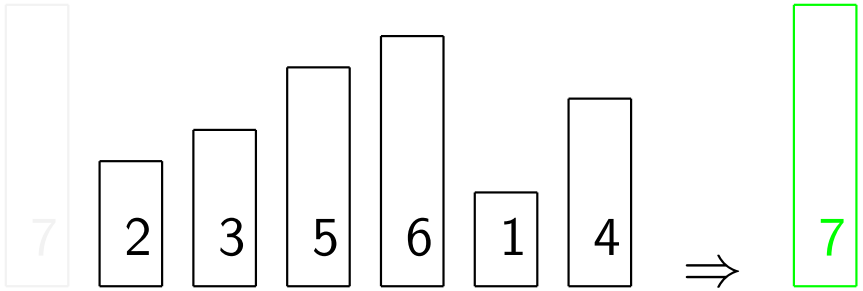
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

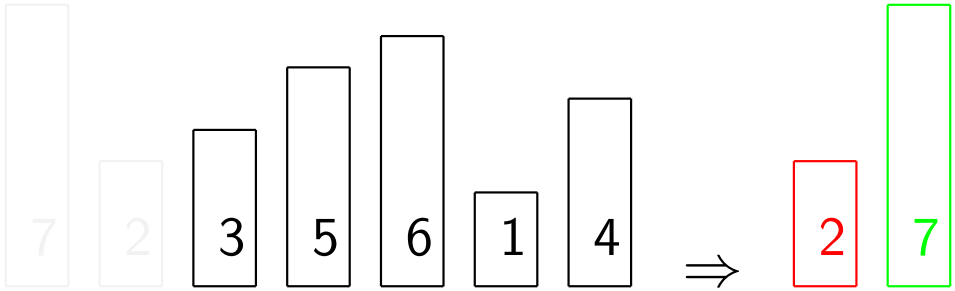
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

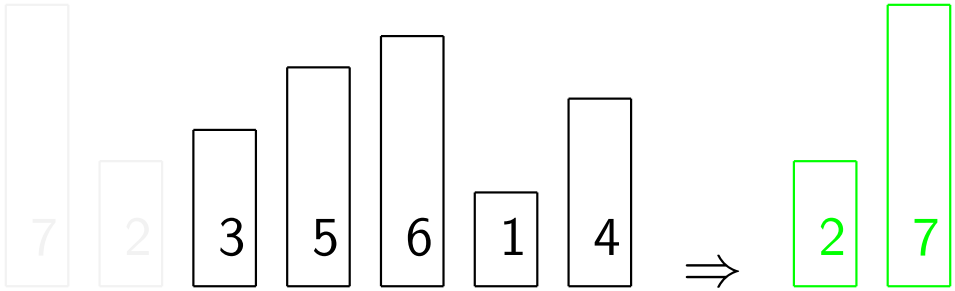
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

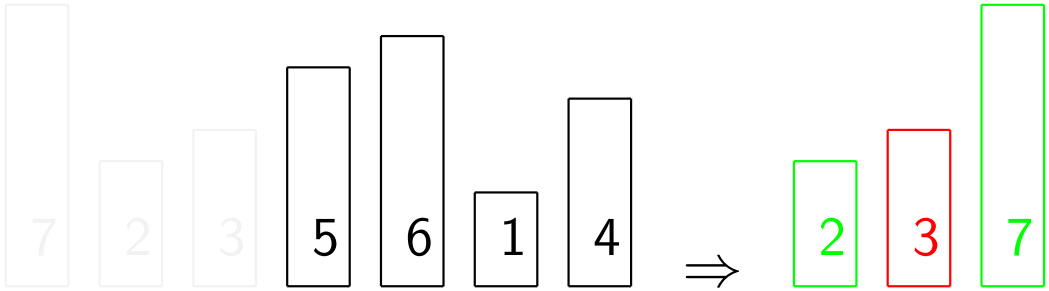
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

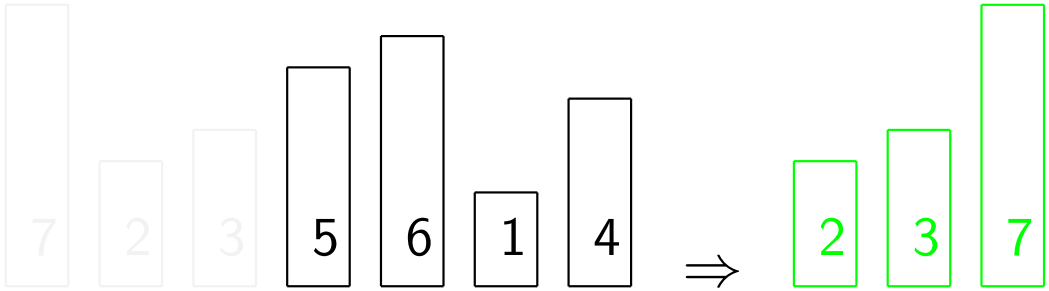
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

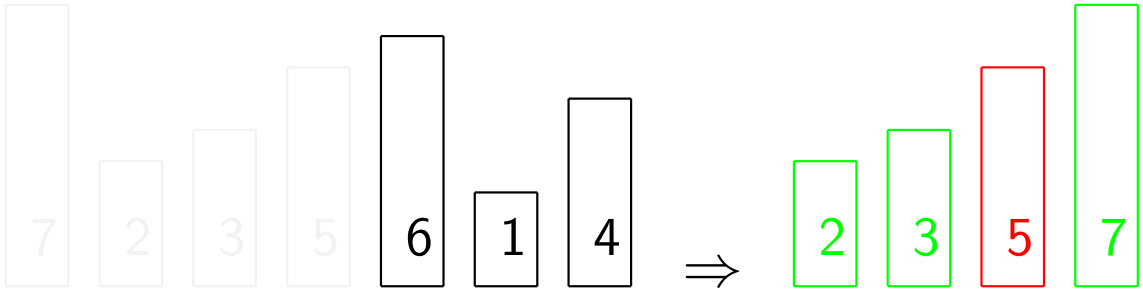
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

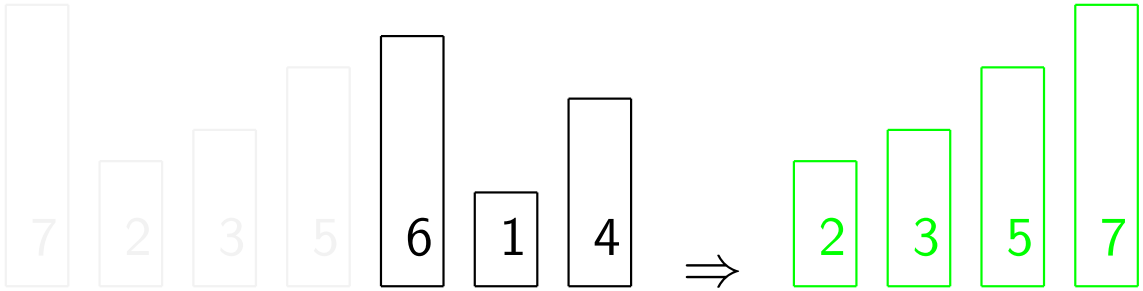
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

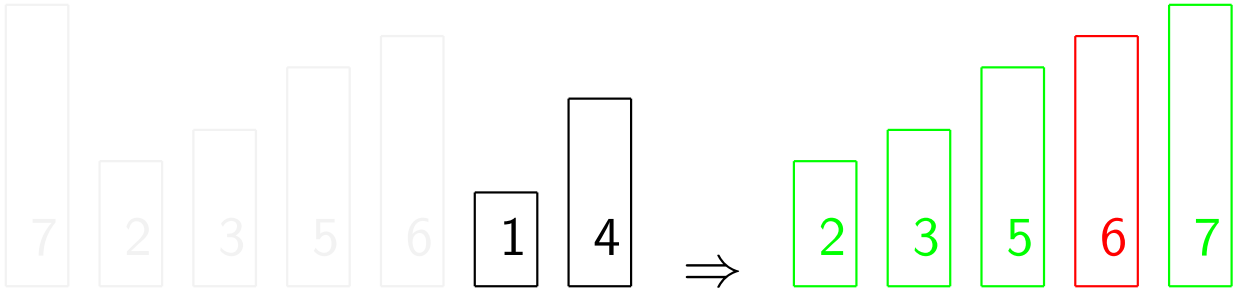
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

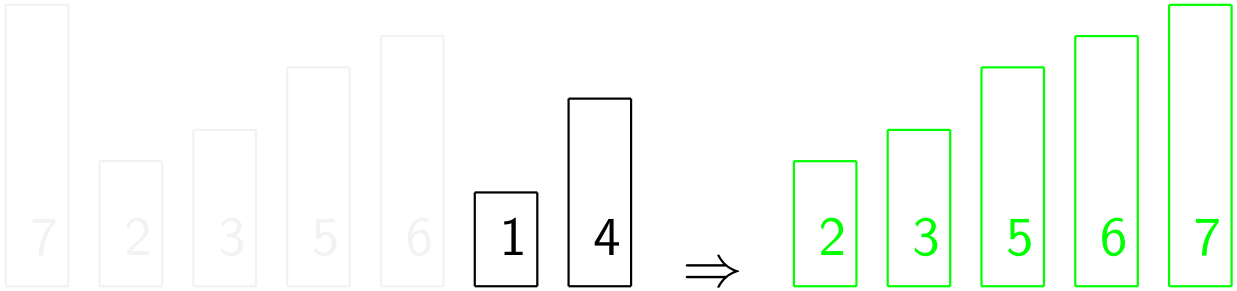
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

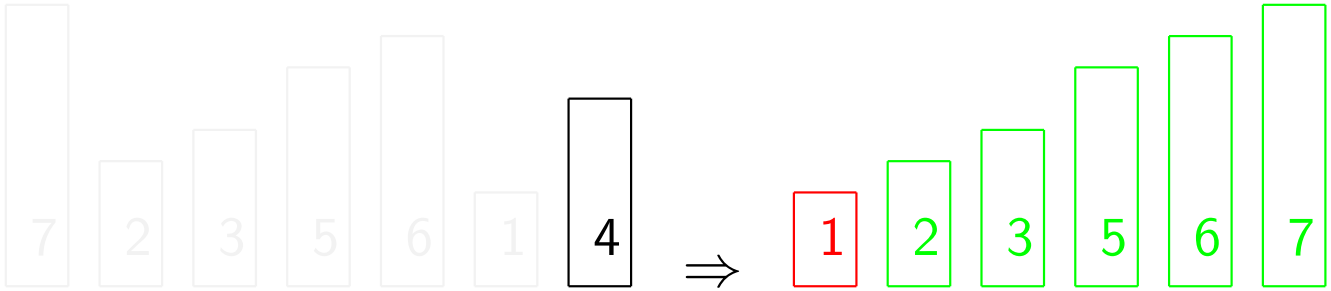
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

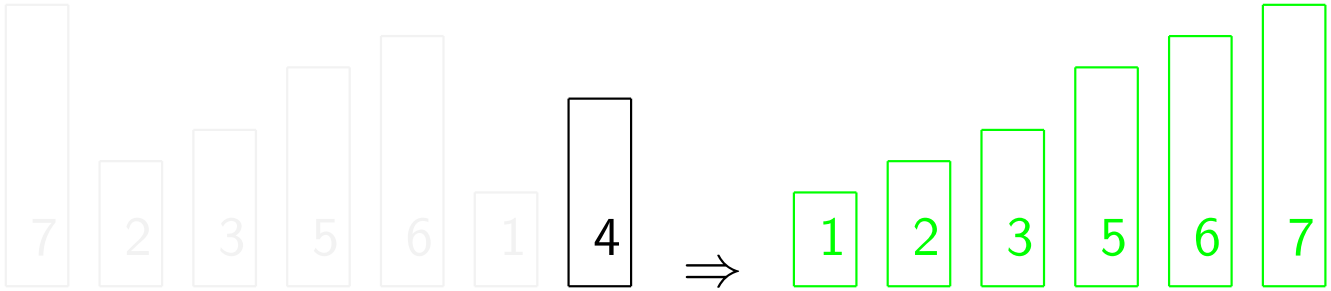
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

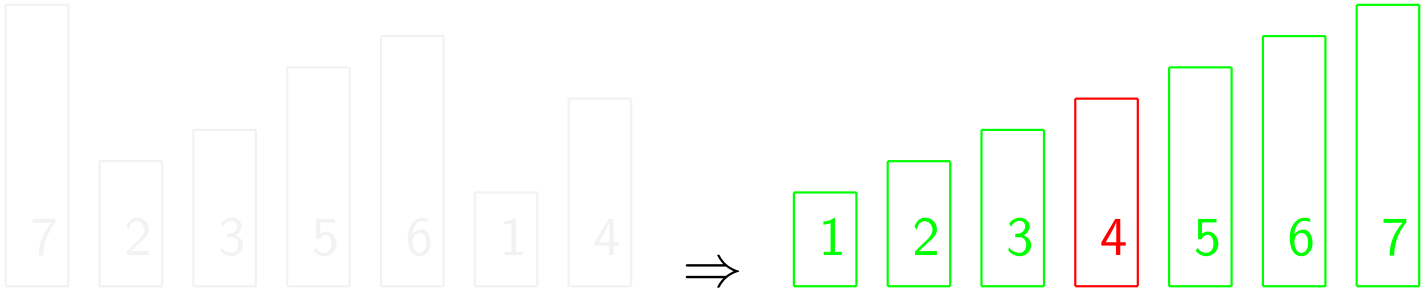
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

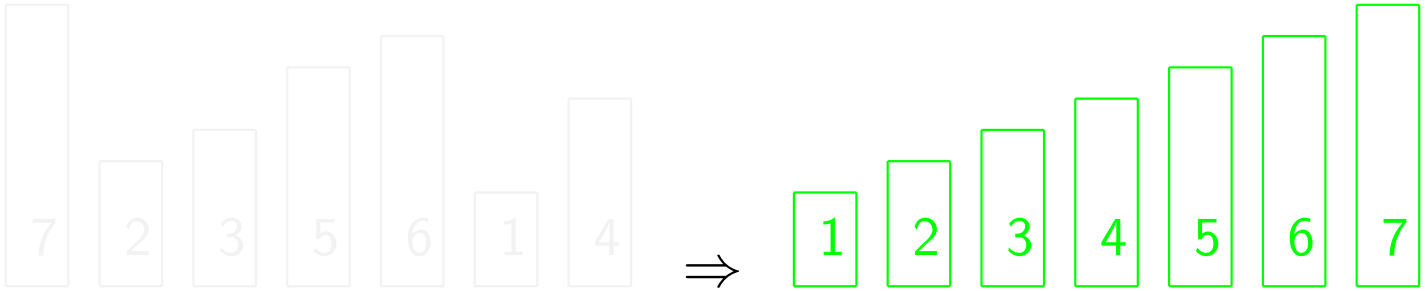
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

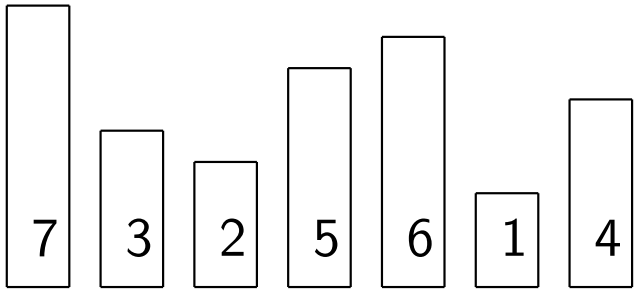
Ordenação por inserção



Ideia

- Inicialmente temos uma lista de itens desordenados
- Retiramos o primeiro elemento
- Inserimos este item em uma nova lista na ordem
- Repetimos tudo com a lista restante (preta)

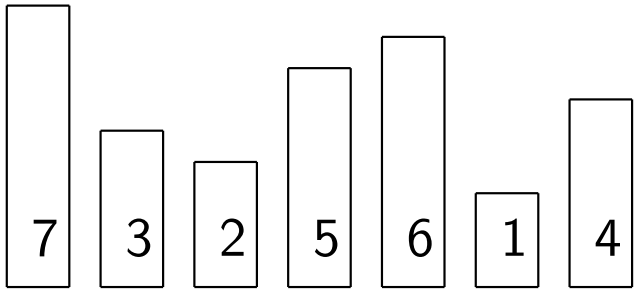
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

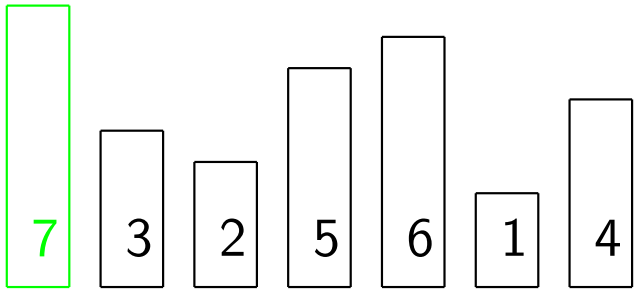
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?

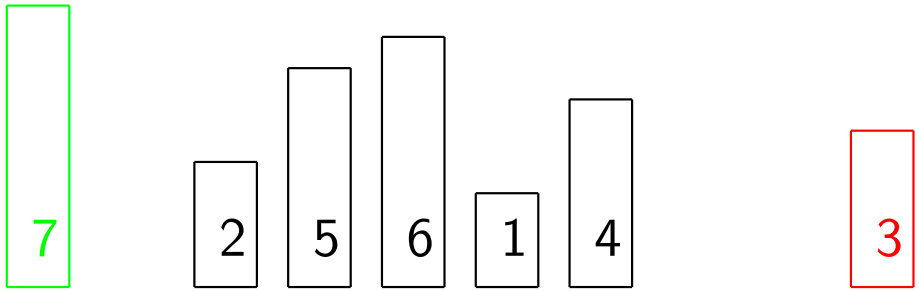
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

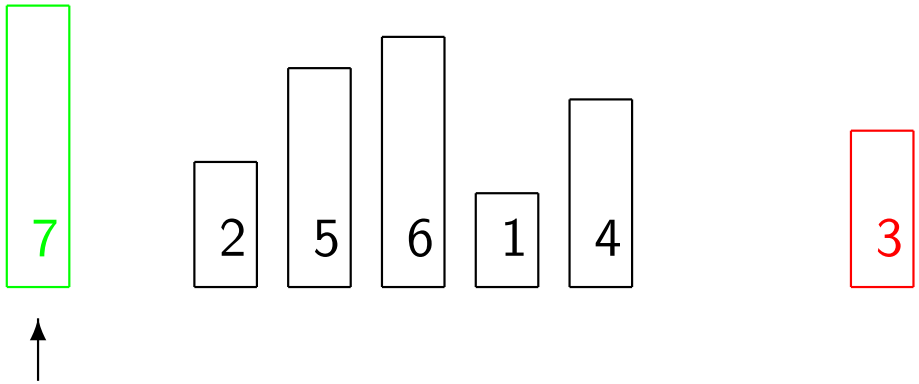
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

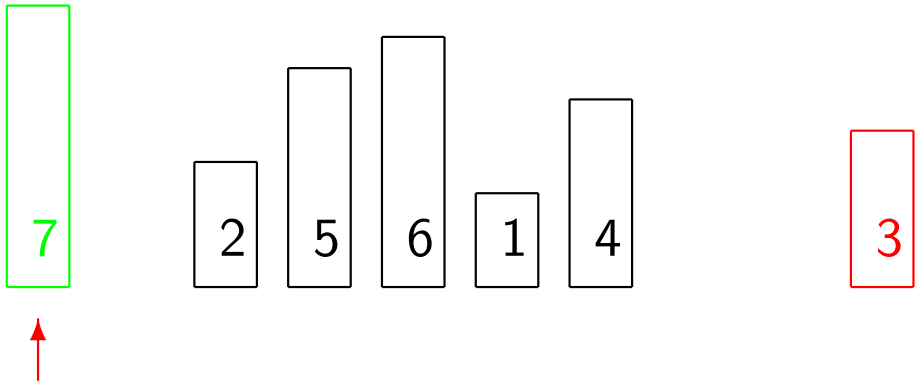
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

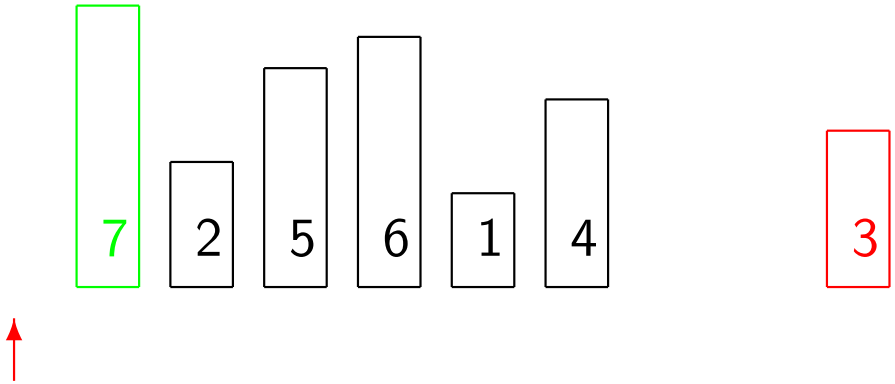
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

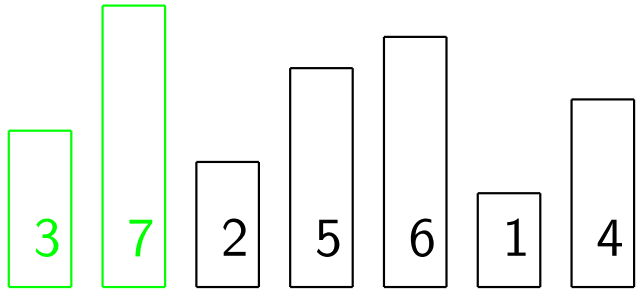
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

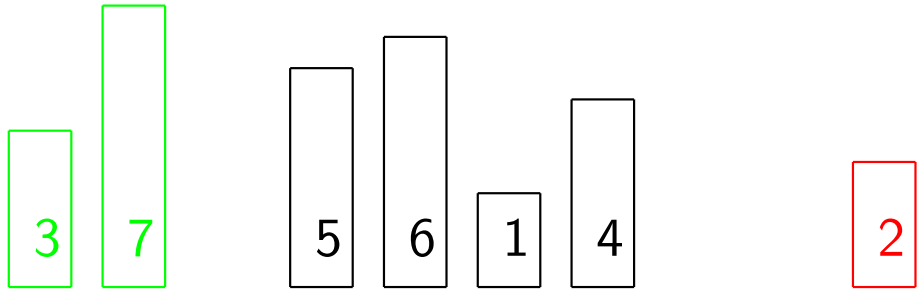
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

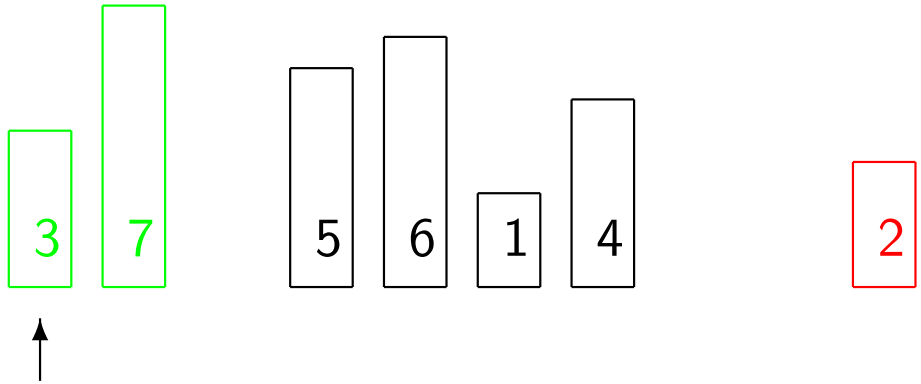
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

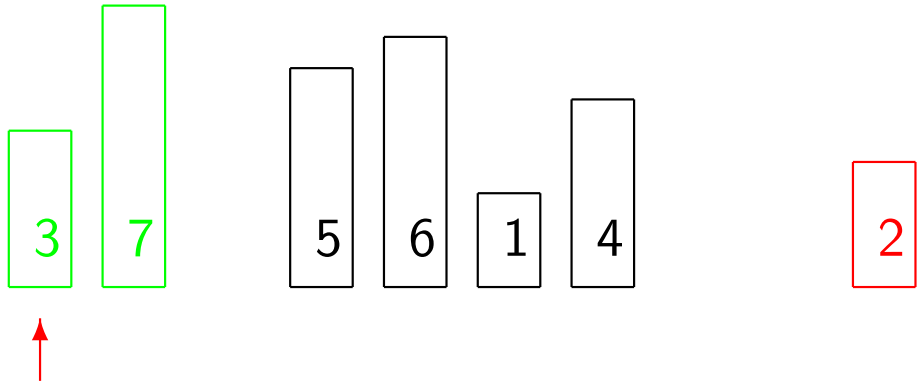
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

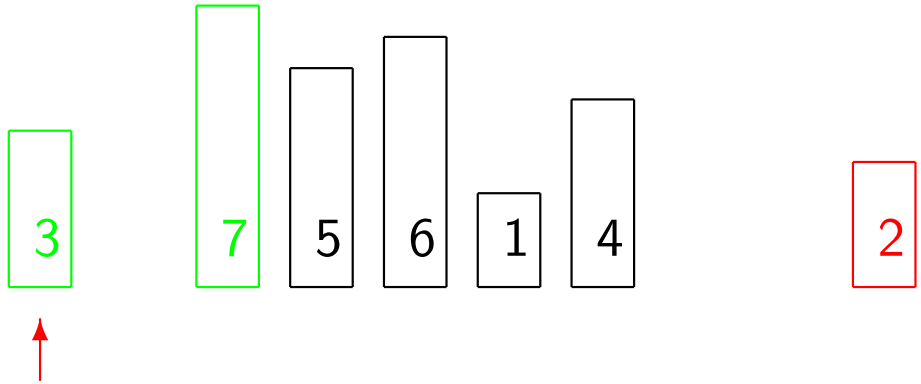
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

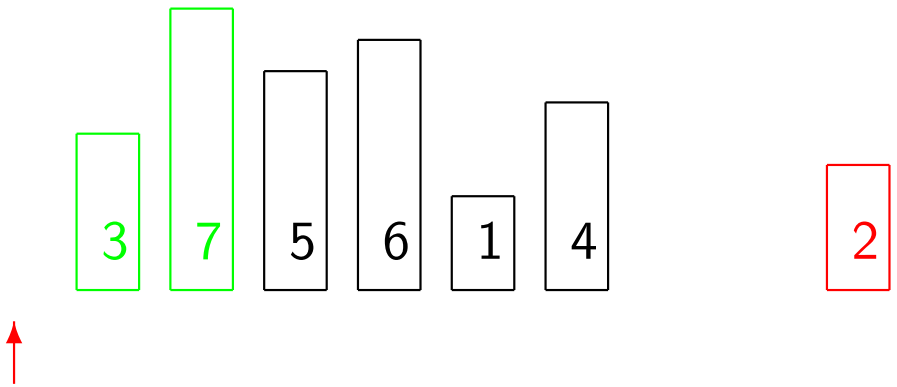
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

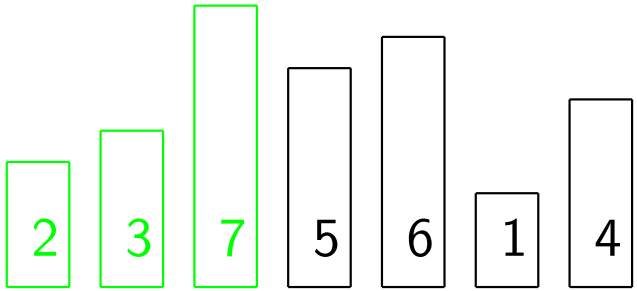
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

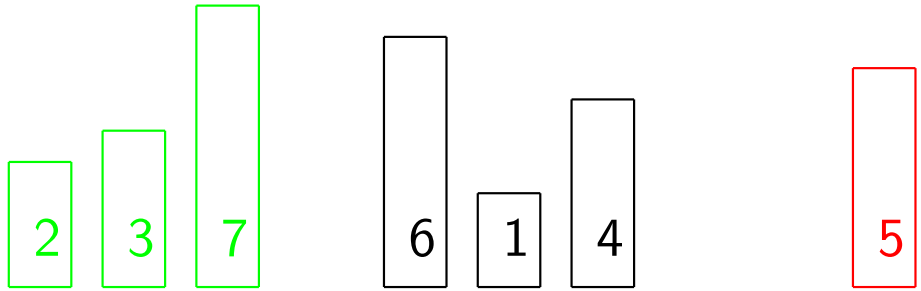
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

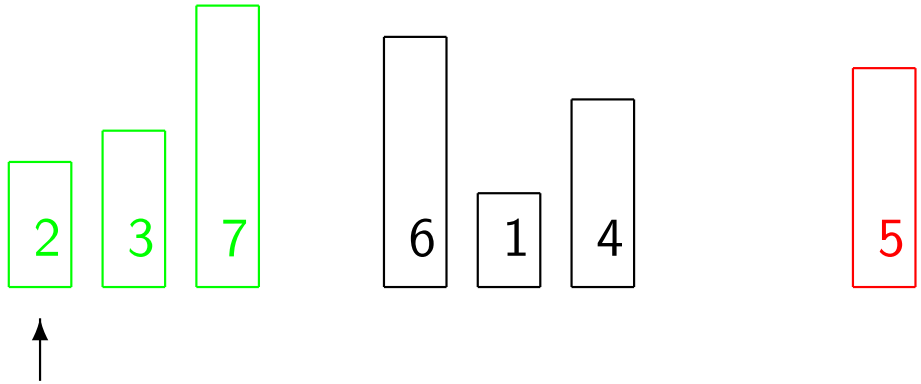
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

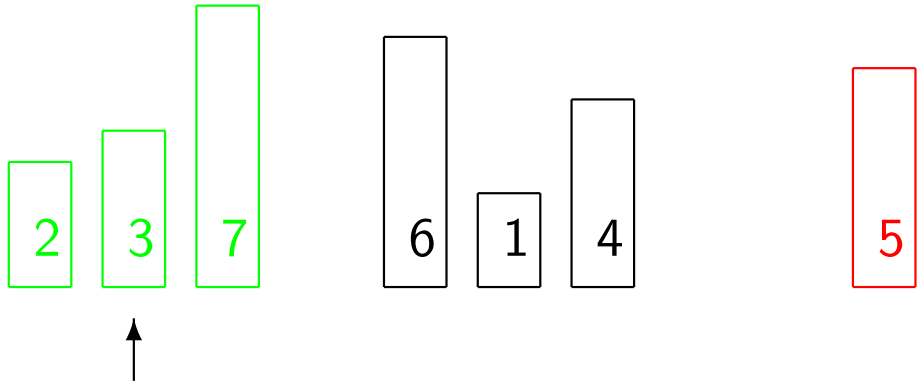
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

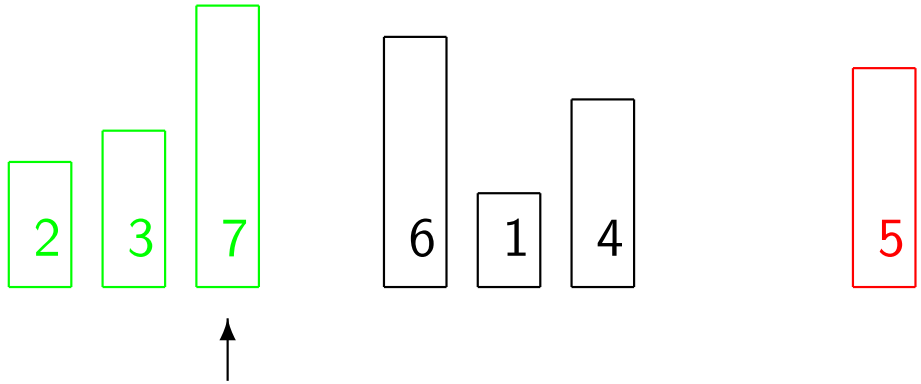
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

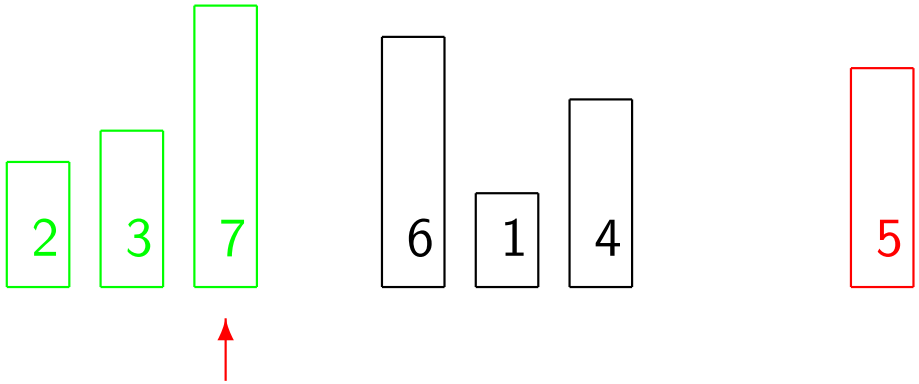
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

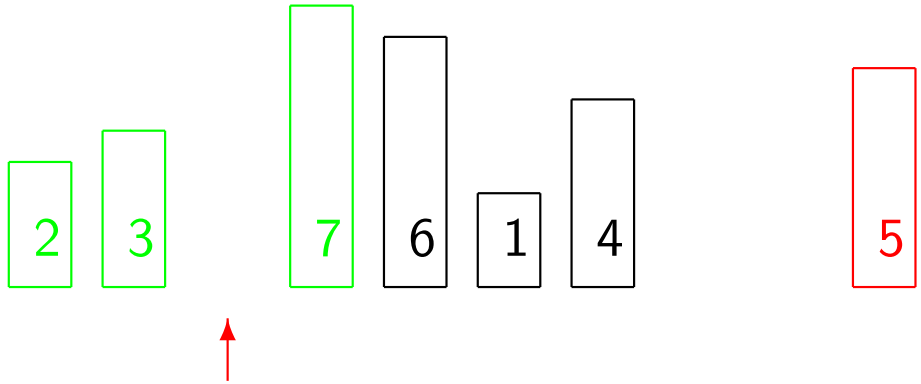
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

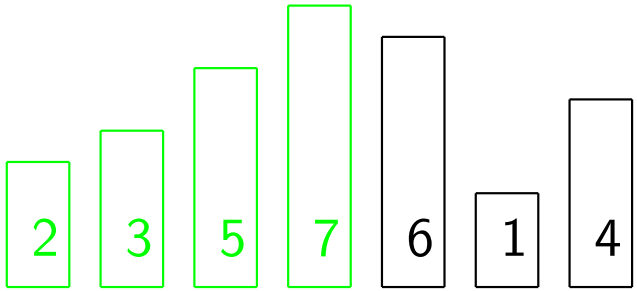
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

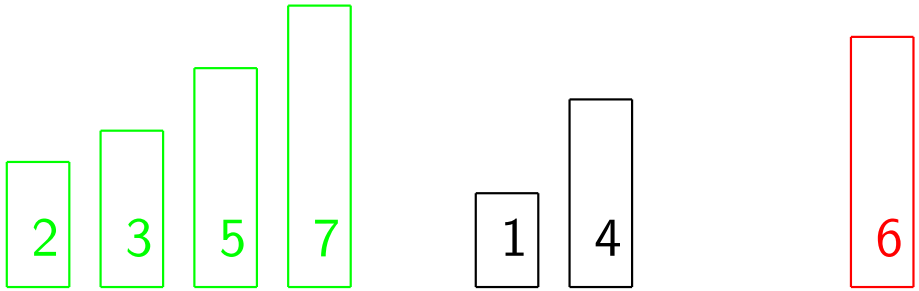
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

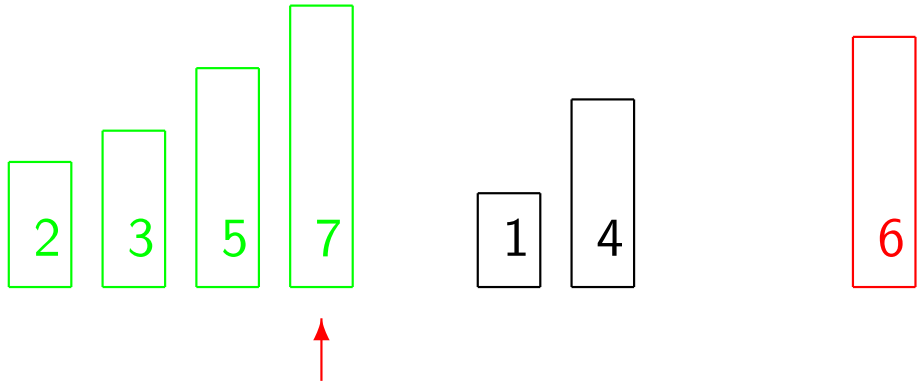
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

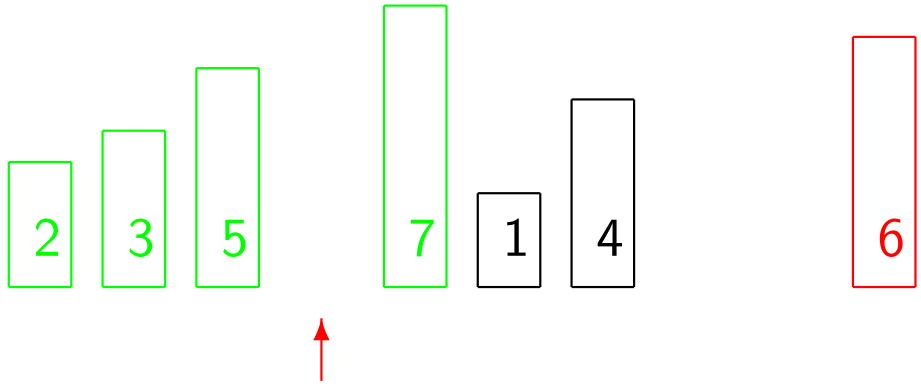
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

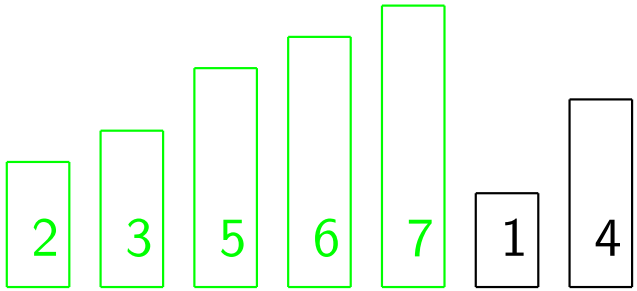
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

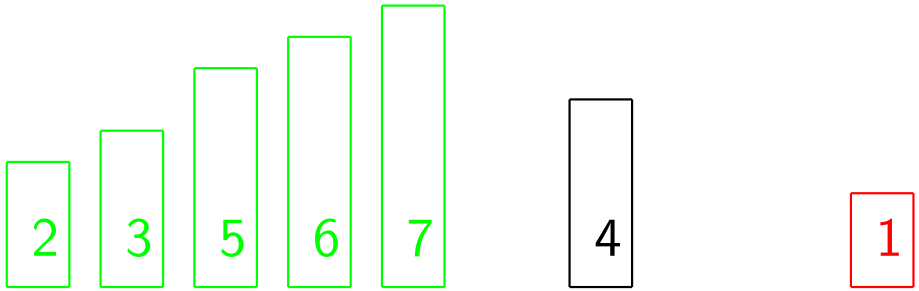
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

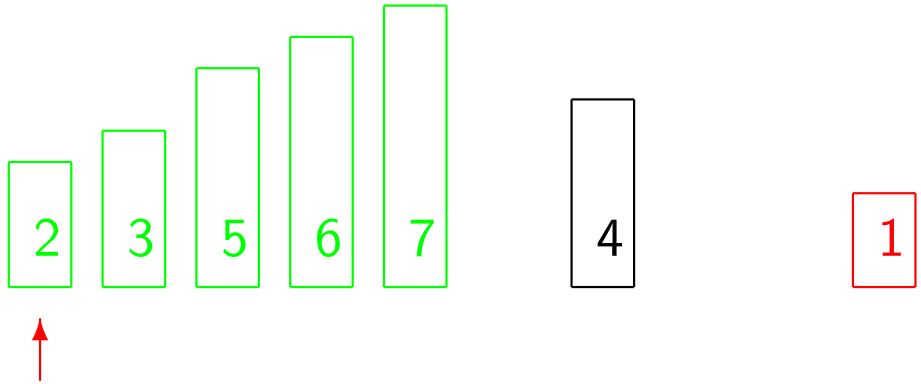
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

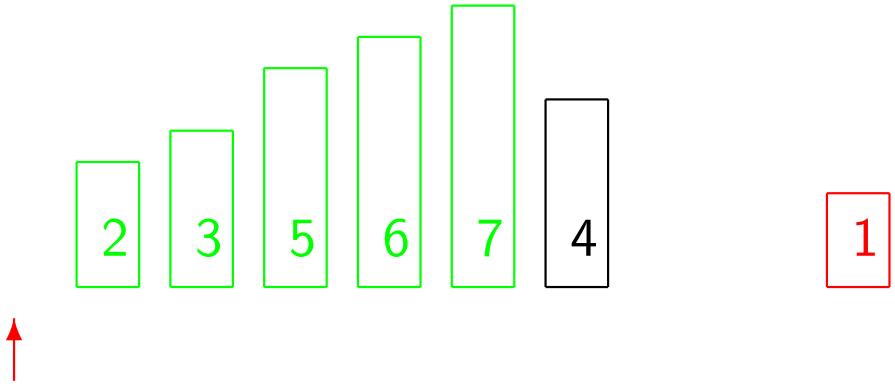
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

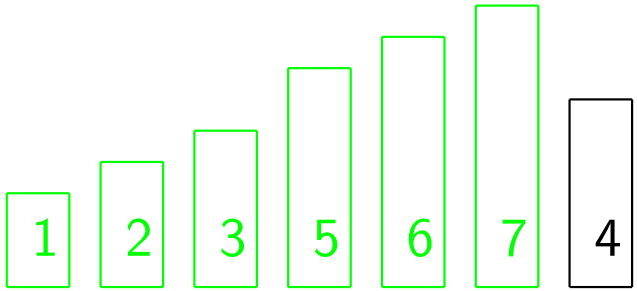
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

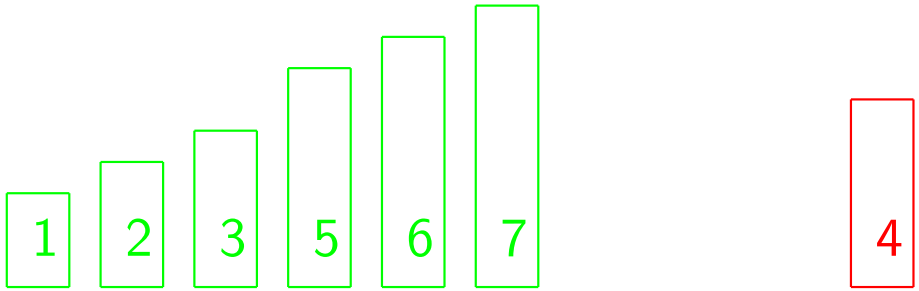
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

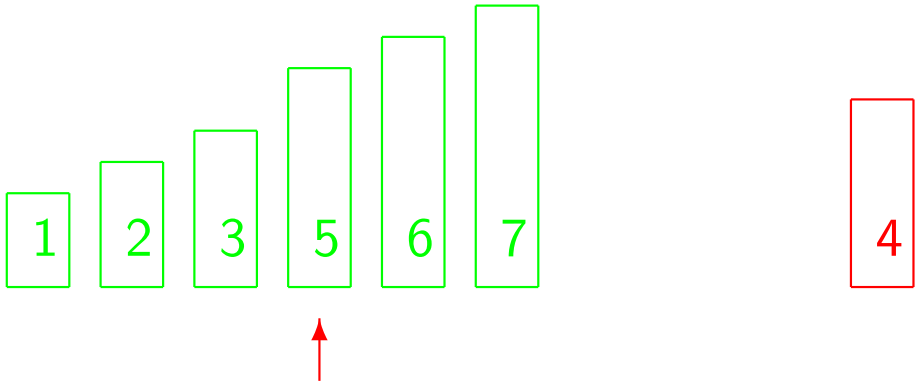
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

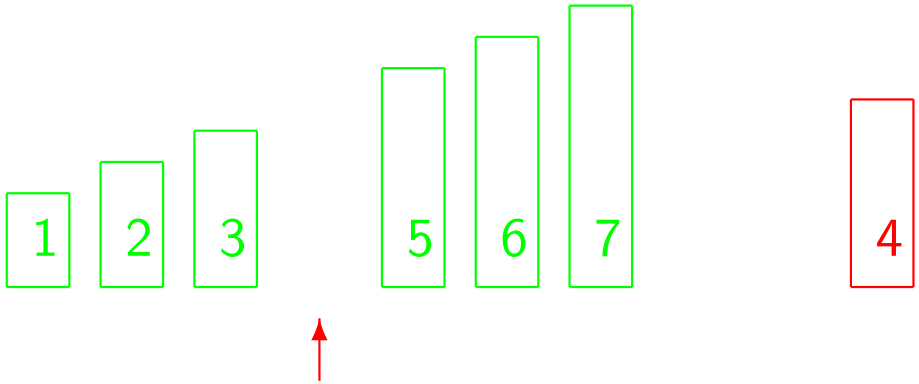
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

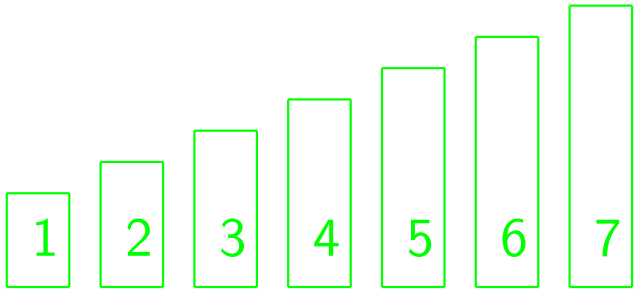
Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

Ordenação por inserção - No mesmo vetor



Ideia

- Como usar apenas um vetor?
  1. O primeiro elemento já está ordenado
  2. Retirar o primeiro elemento desordenado
  3. Procurar a posição em que deve ser inserido
  4. Deslocar os elementos ordenados seguintes
  5. Inserir o elemento retirado na ordem correta
  6. Continuar com a lista restante (preta)

Algoritmo de ordenação por inserção (*Insertion-Sort*)

Posição de inserção (na lista verde)

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
    int i;
    for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
    return i;
}
```

Deslocar parte do vetor

```
void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
    int i;
    for (i = ultimo; i >= primeiro; i--) {
        vetor[i+1] = vetor[i];
    }
}
```

Algoritmo de ordenação por inserção (*Insertion-Sort*)

Posição de inserção (na lista verde)

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
    int i;
    for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
    return i;
}
```

Deslocar parte do vetor

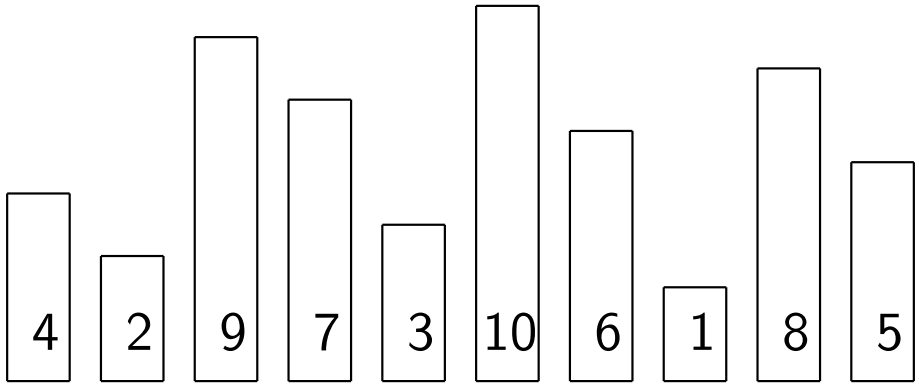
```
void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
    int i;
    for (i = ultimo; i >= primeiro; i--) {
        vetor[i+1] = vetor[i];
    }
}
```

Algoritmo de ordenação por inserção (*Insertion-Sort*)

Ordenação por inserção

```
int ordenar_insercao(int vetor[], int n) {
    int i, posicao;
    int elemento;

    for (i = 1; i < n; i++) {
        elemento = vetor[i];
        posicao = posicao_elemento(vetor, i-1, elemento);
        deslocar_subvetor(vetor, posicao, i-1);
        vetor[posicao] = elemento;
    }
}
```

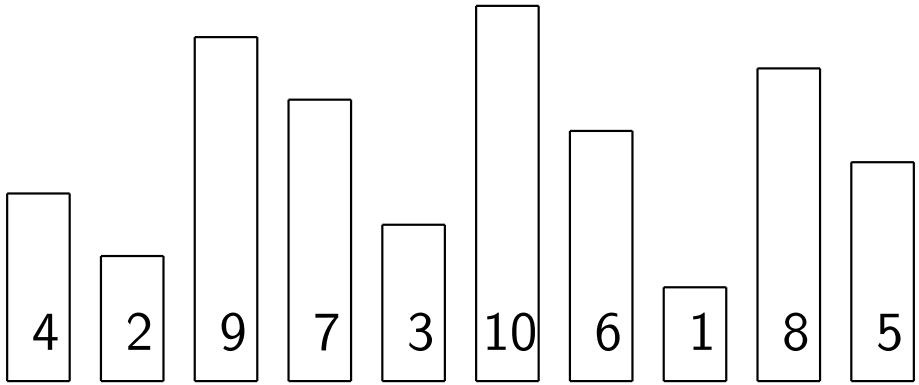
Introdução - Ordenação por Intercalação



Problema 1

Suponha que temos um vetor desordenado com 10 números. Como ordenar a primeira metade da lista números?

Ordenando a primeira parte

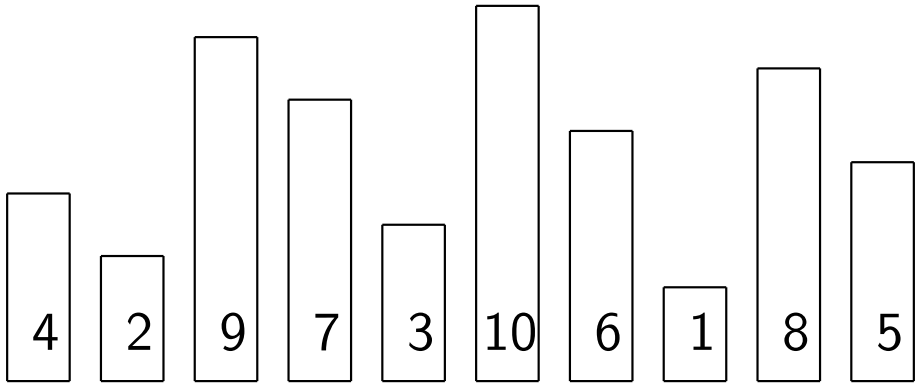


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

Ordenando a primeira parte

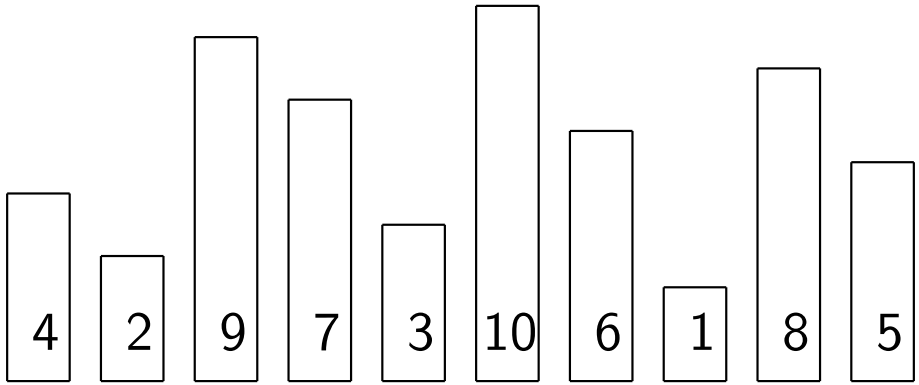


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

Ordenando a primeira parte

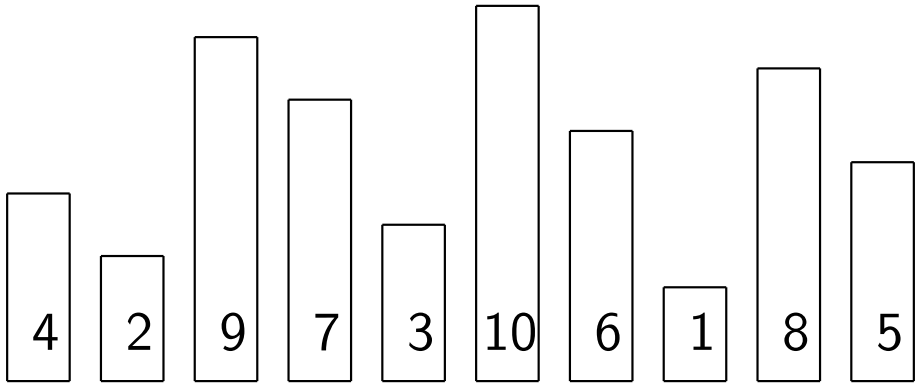


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

Ordenando a primeira parte

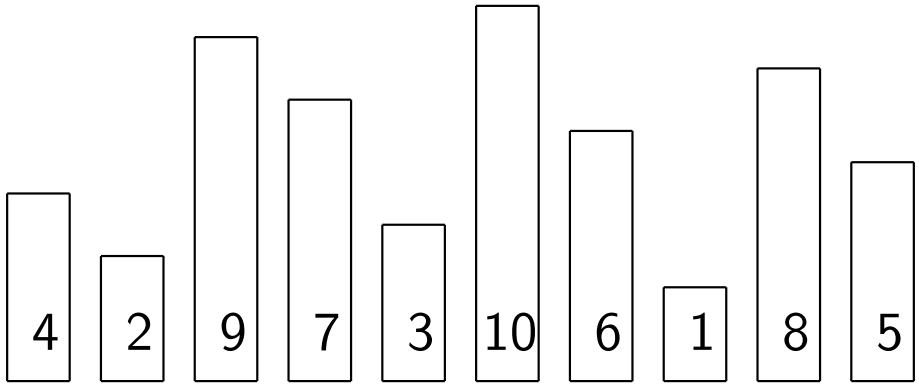


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

Ordenando a primeira parte

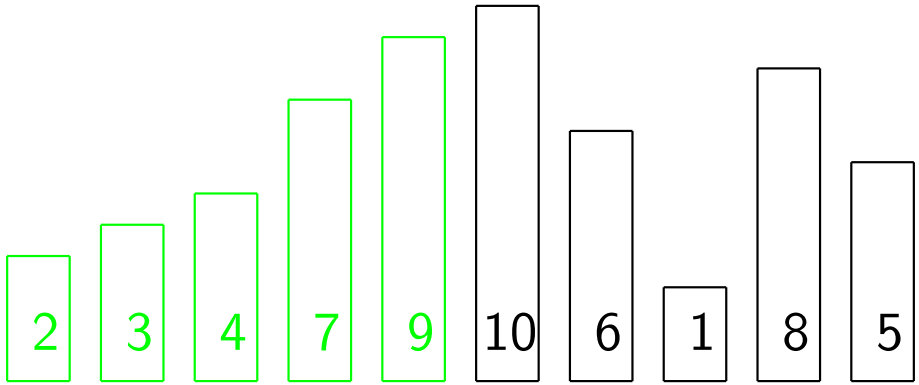


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

Ordenando a primeira parte

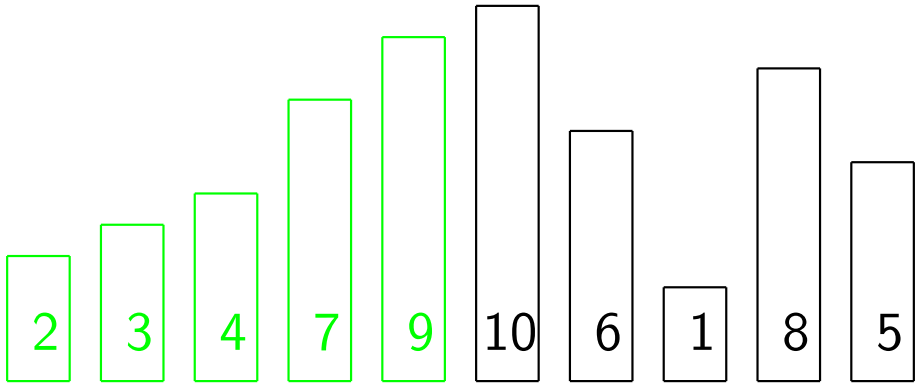


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

Ordenando a primeira parte

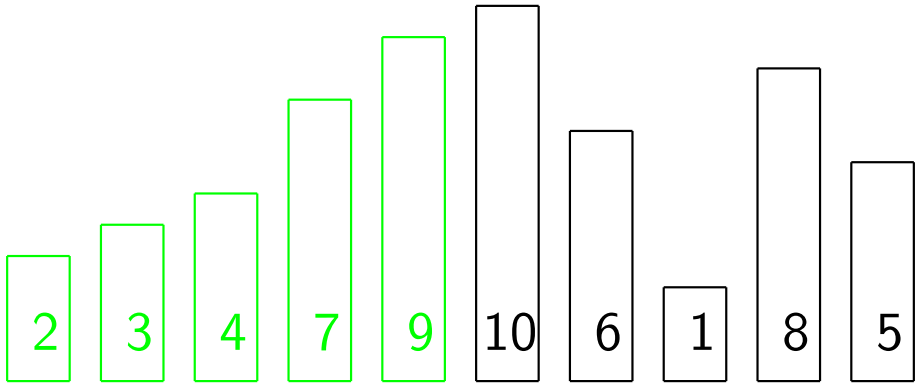


Suponha que

- temos uma função `ordenar(int vetor[], int ini, int fim)`
- ela ordena o vetor da posição *ini* até *fim*
- o vetor é indexado da posição 1 até 10
- executamos `ordenar(vetor, 1, 5);`

E se quiséssemos ordenar a segunda parte?

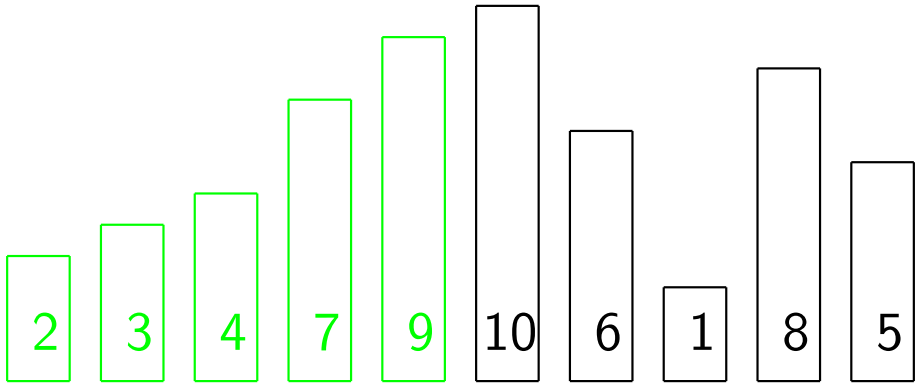
...e a segunda parte



Suponha que

- agora queremos ordenar a segunda parte
- executamos `ordenar(vetor, 6, 10);`

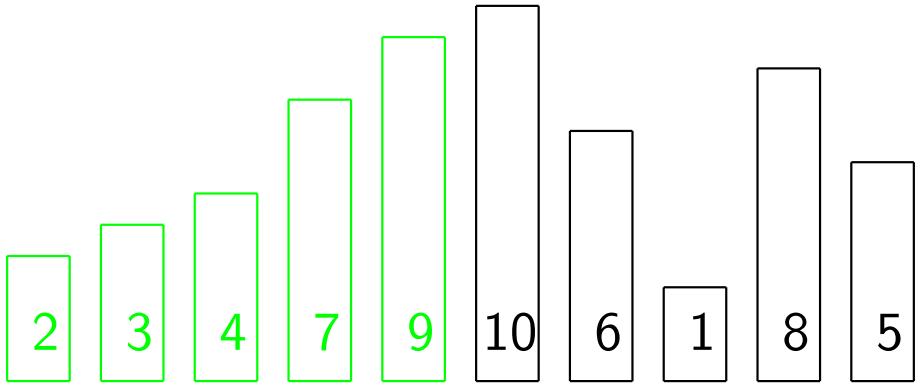
...e a segunda parte



Suponha que

- agora queremos ordenar a segunda parte
- executamos `ordenar(vetor, 6, 10);`

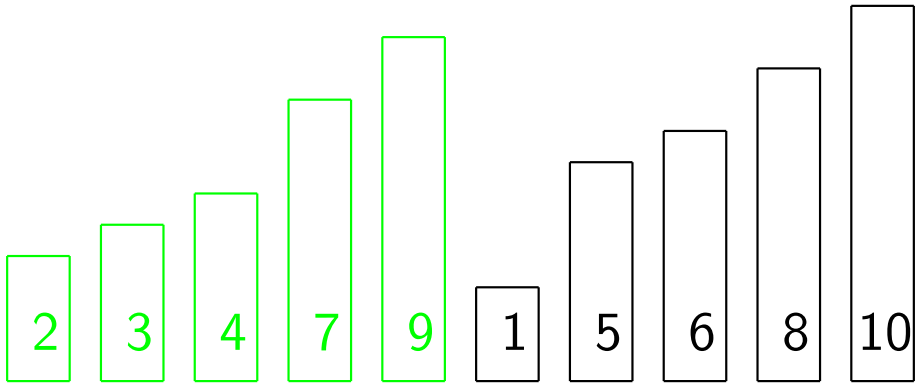
...e a segunda parte



Suponha que

- agora queremos ordenar a segunda parte
- executamos `ordenar(vetor, 6, 10);`

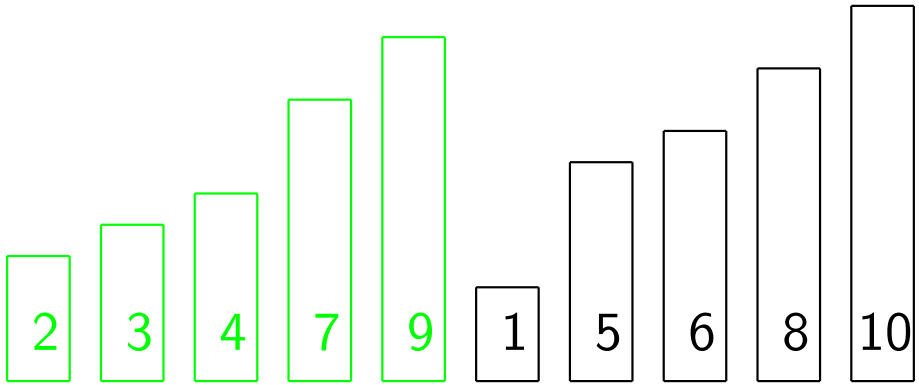
...e a segunda parte



Suponha que

- agora queremos ordenar a segunda parte
- executamos `ordenar(vetor, 6, 10);`

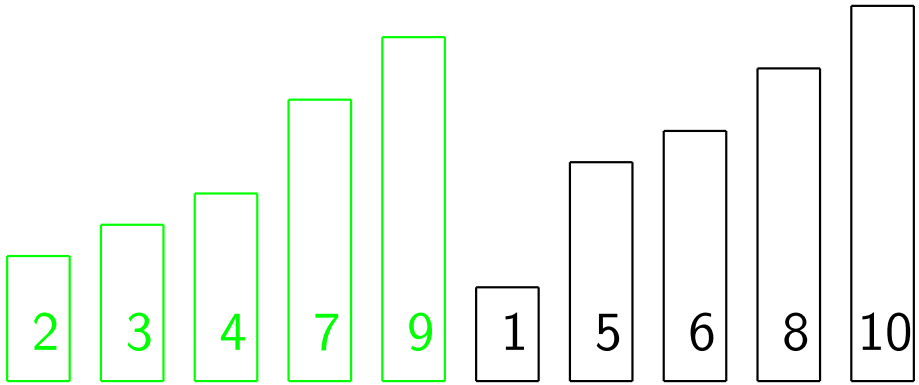
Ordenando tudo



Problema

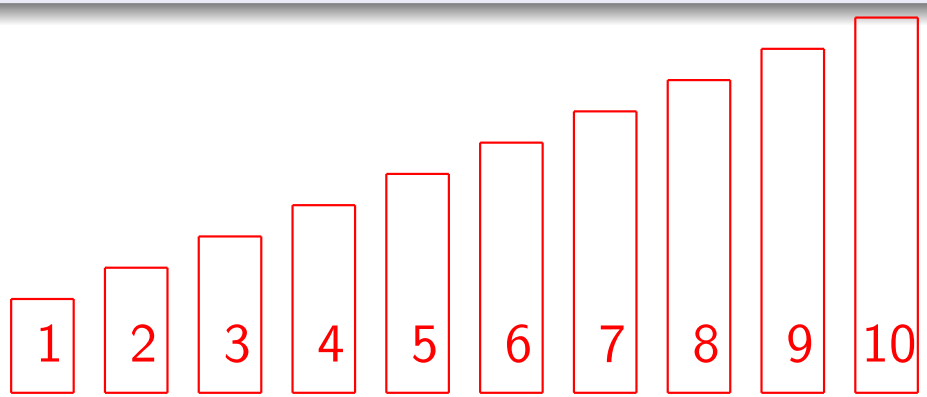
Suponha que temos um vetor de 10 números com as duas metades já ordenadas. Como criar um novo vetor com todos os elementos ordenados?

Ordenando tudo

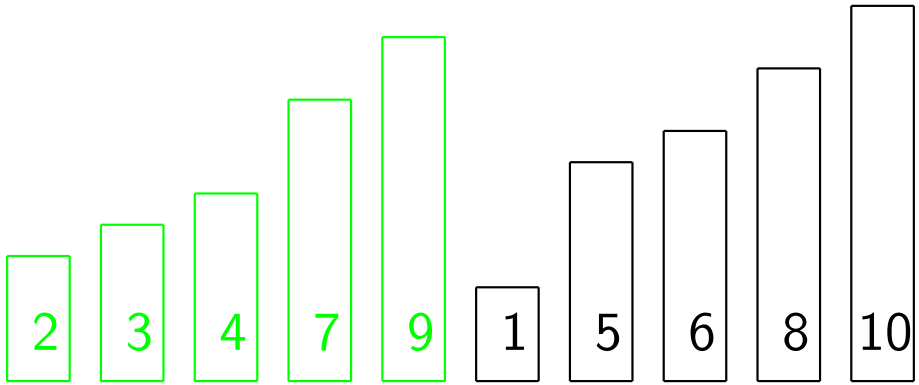


Problema

Suponha que temos um vetor de 10 números com as duas metades já ordenadas. Como criar um novo vetor com todos os elementos ordenados?

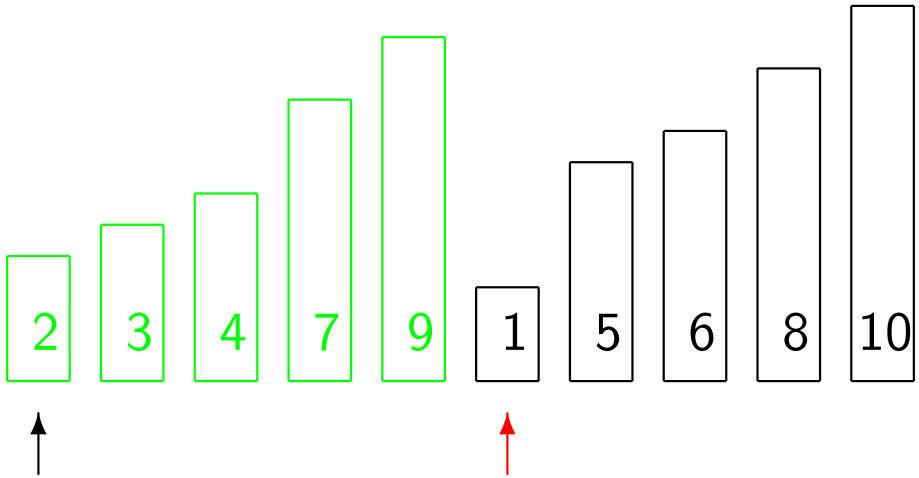


Intercalando



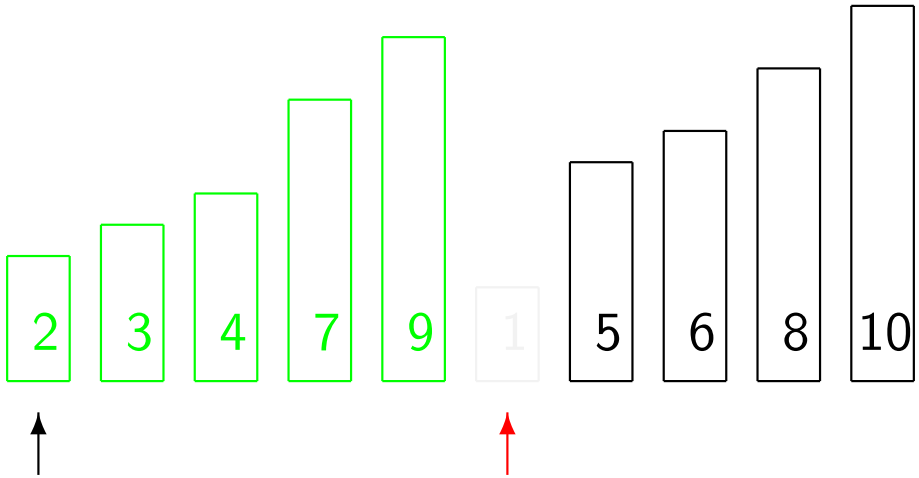
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor
- Depois movemos o resto.

Intercalando

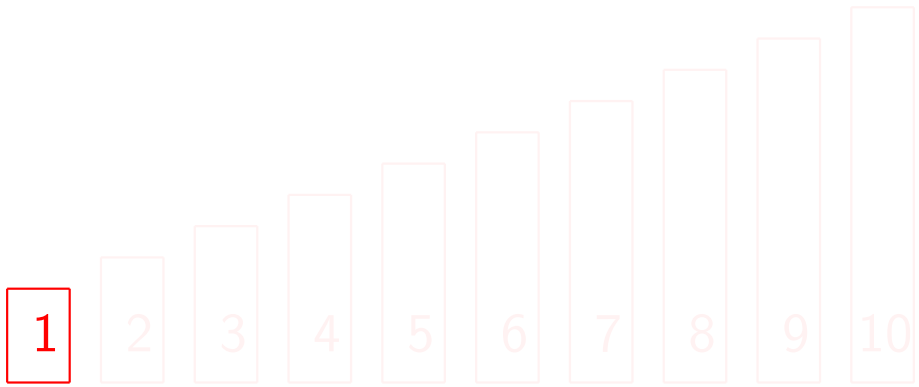


- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor
- Depois movemos o resto.

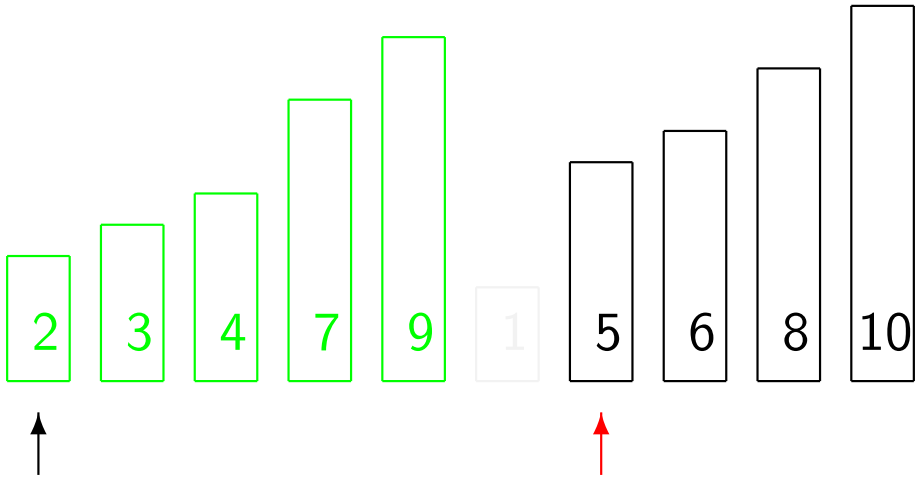
Intercalando



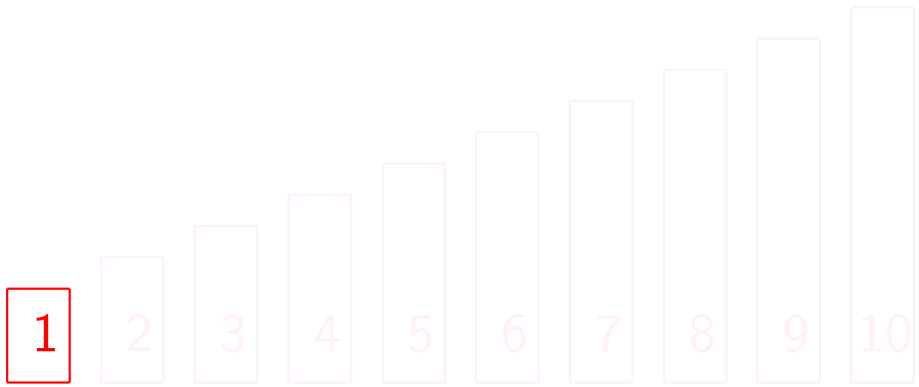
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor
- Depois movemos o resto.



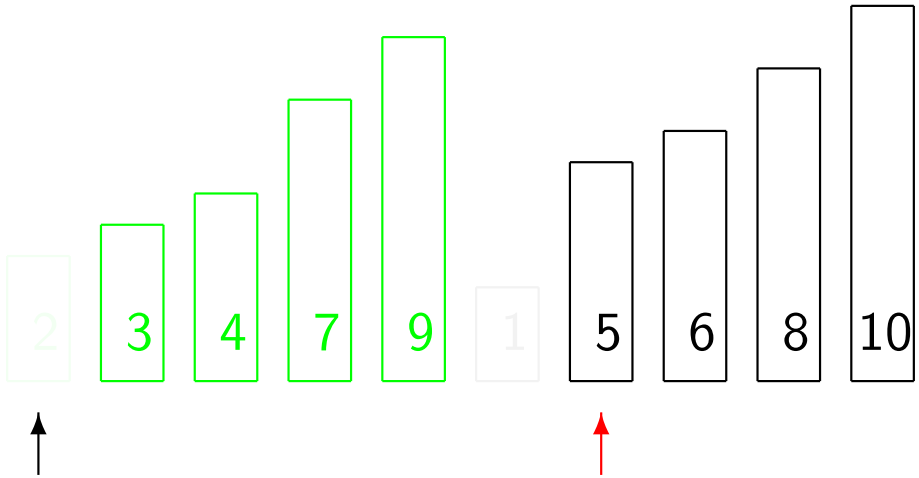
Intercalando



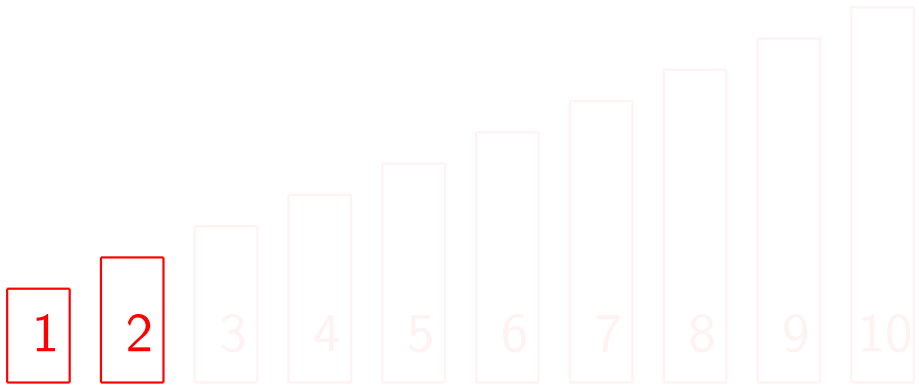
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



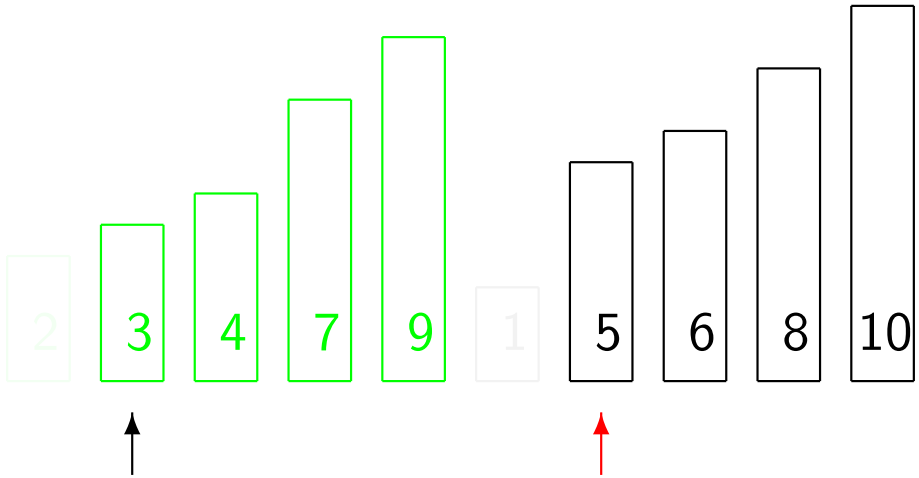
Intercalando



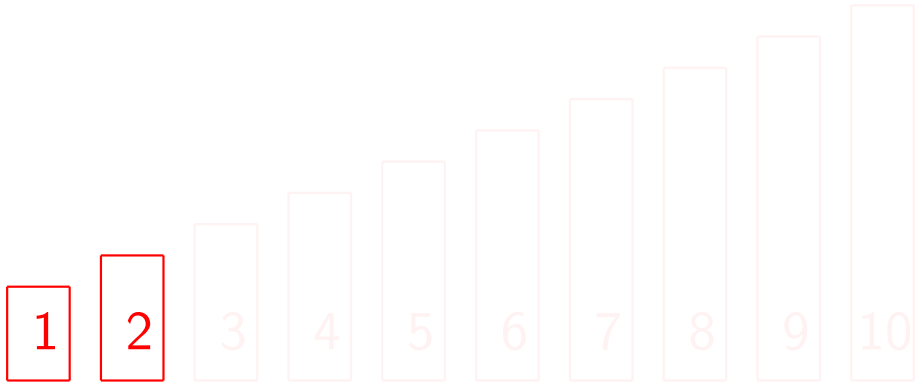
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



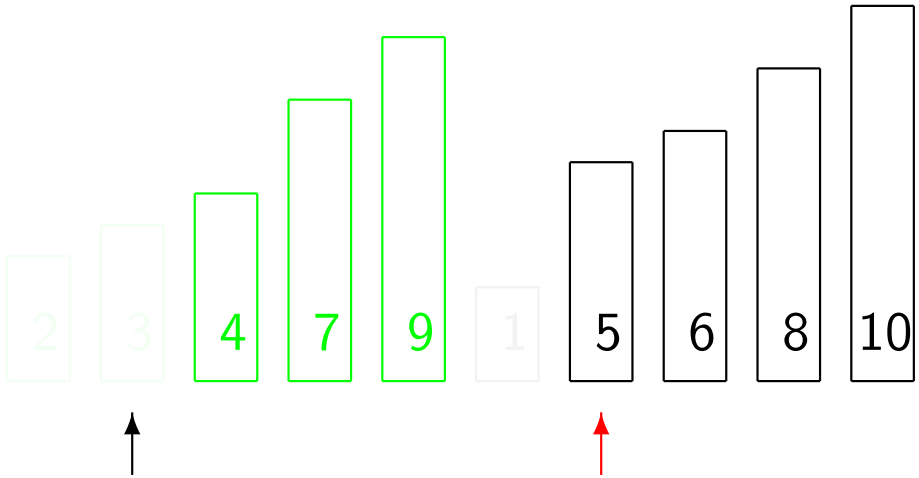
Intercalando



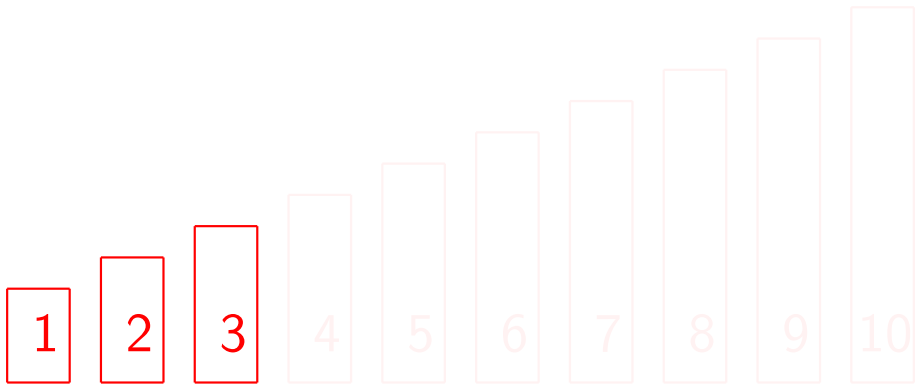
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



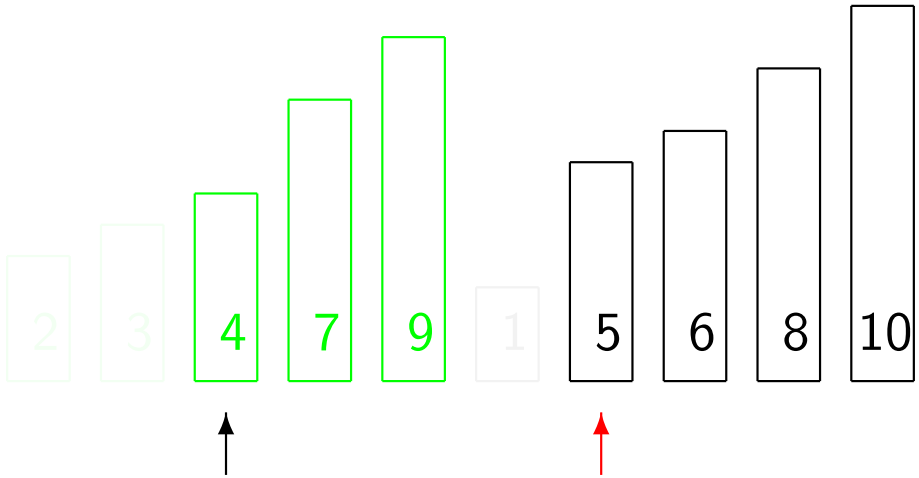
Intercalando



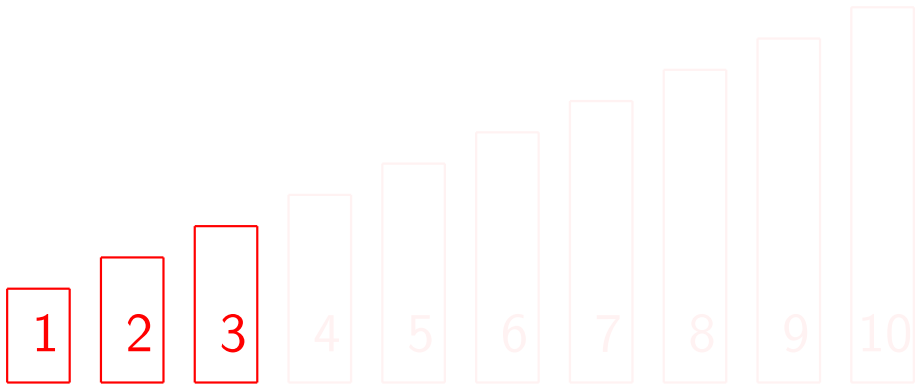
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



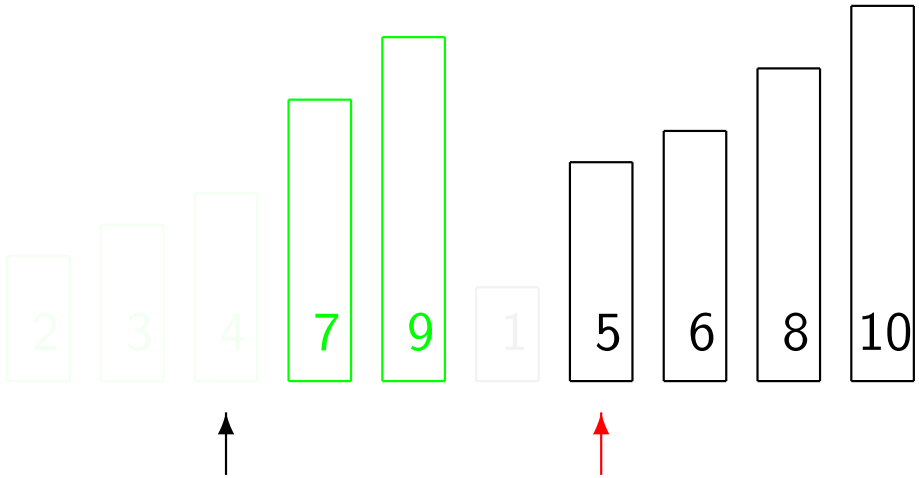
Intercalando



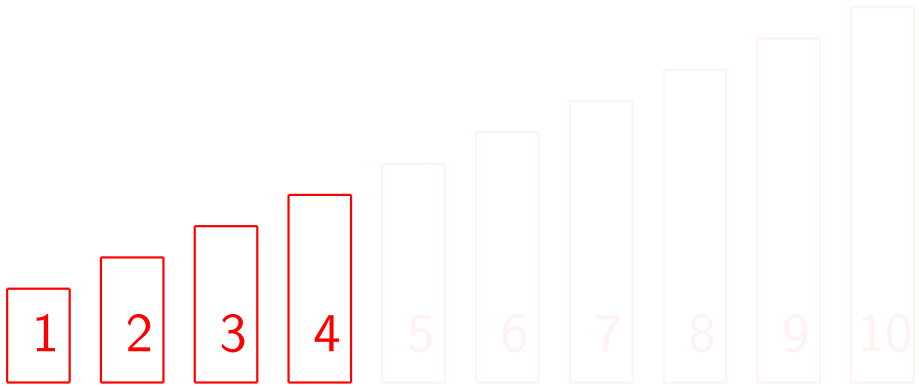
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



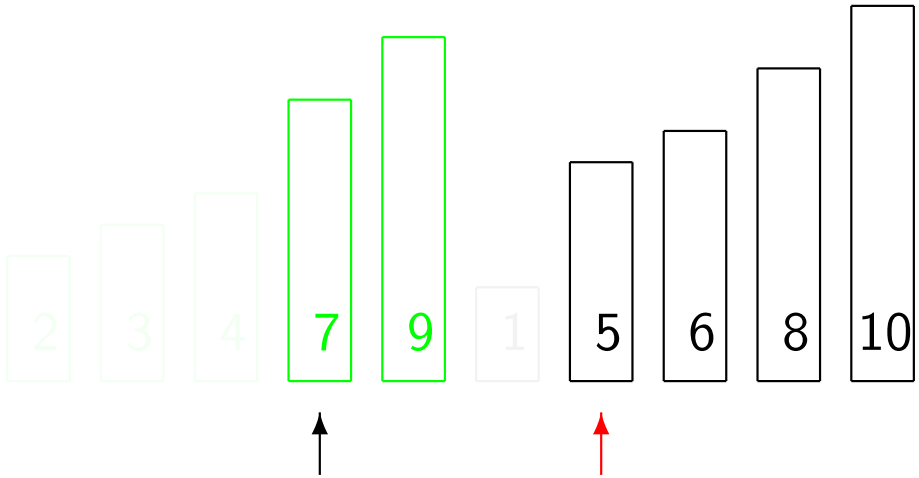
Intercalando



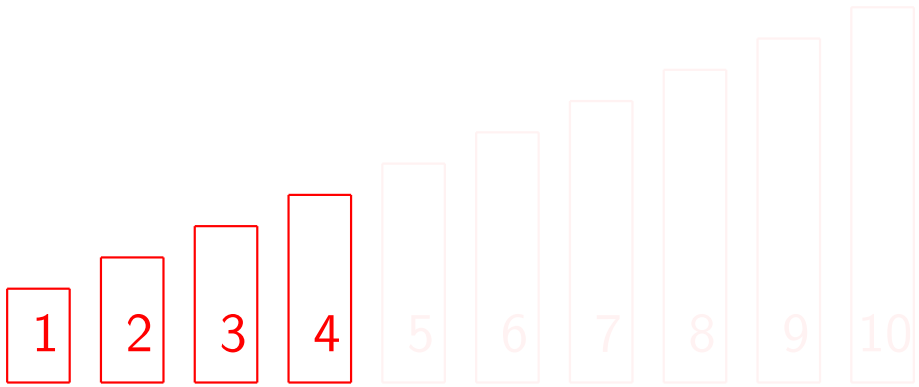
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



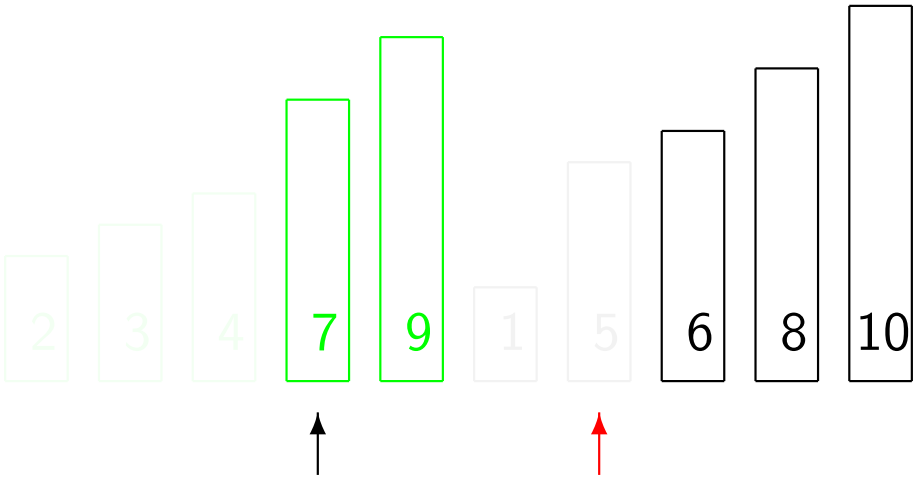
Intercalando



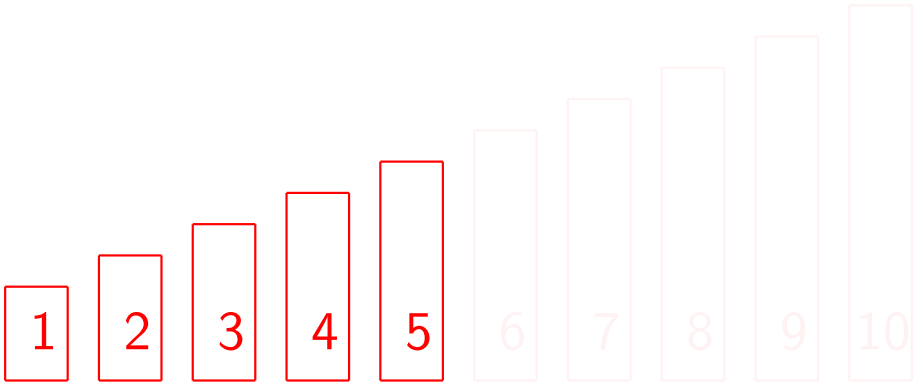
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



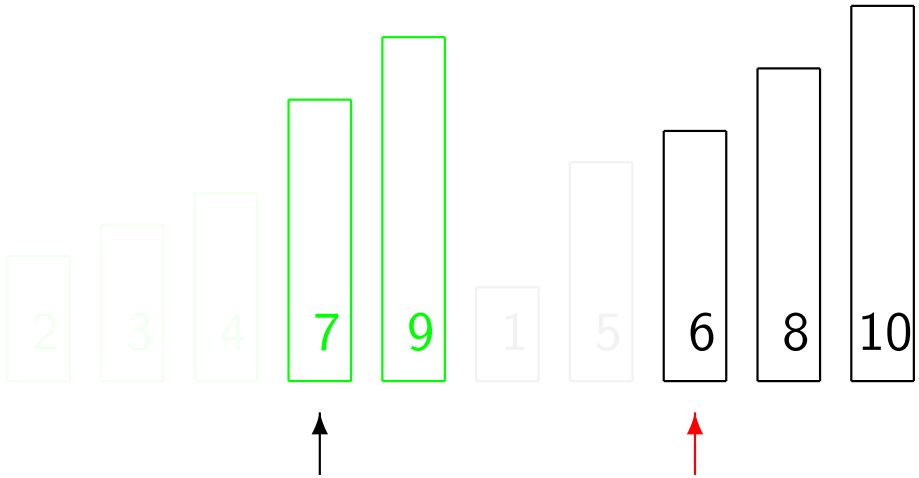
Intercalando



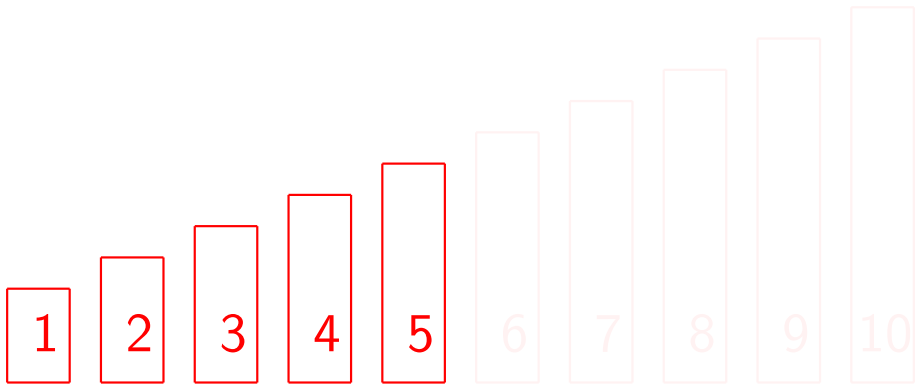
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



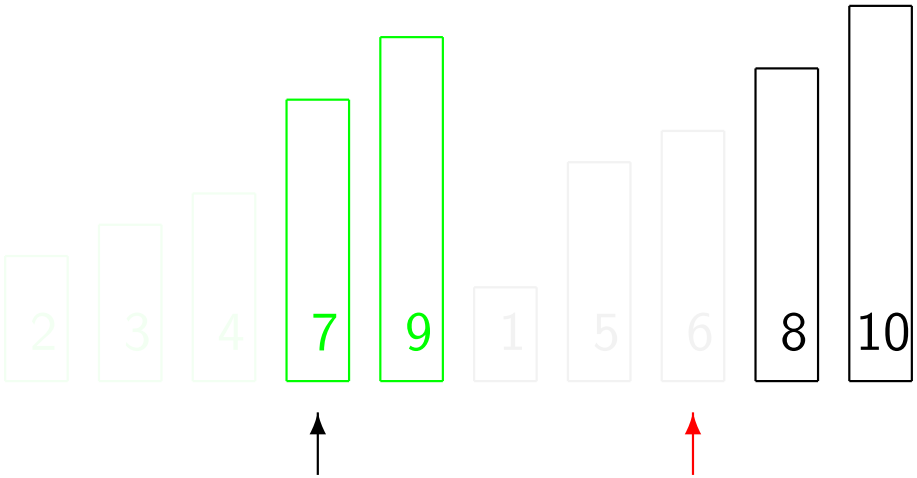
Intercalando



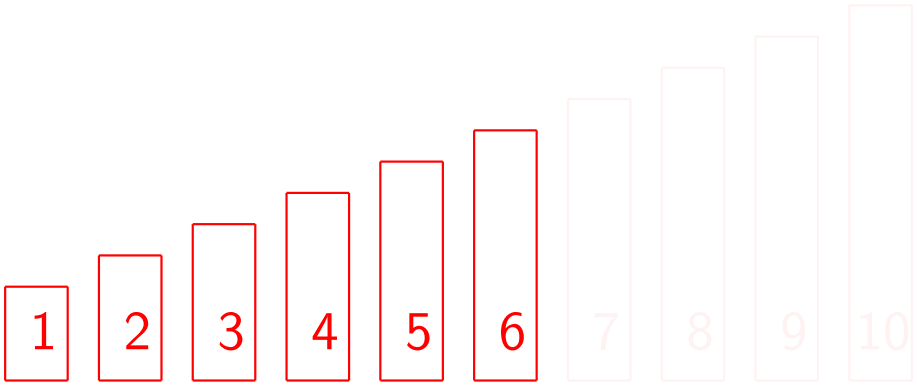
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



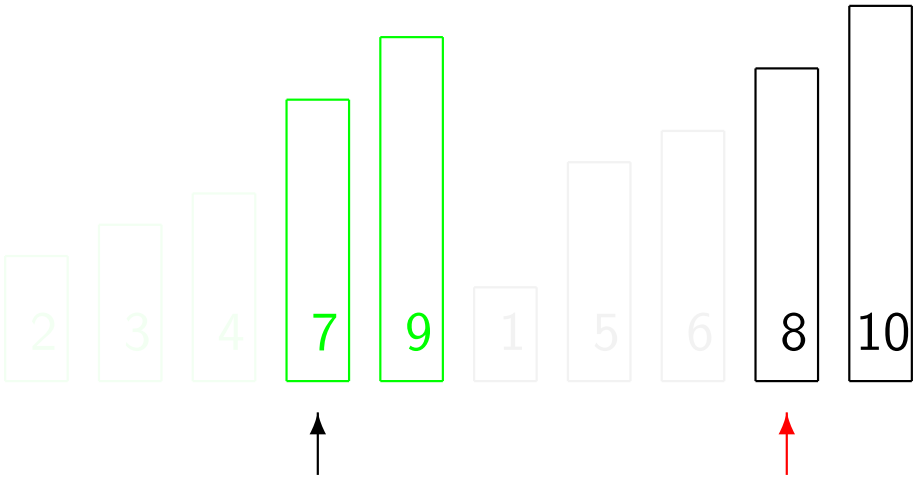
Intercalando



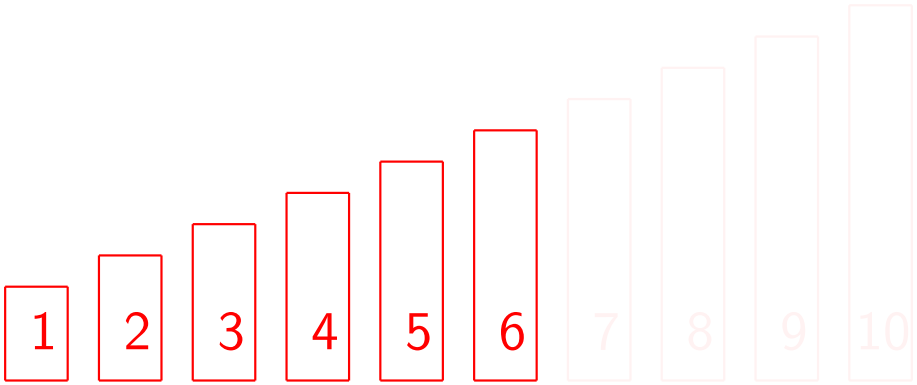
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



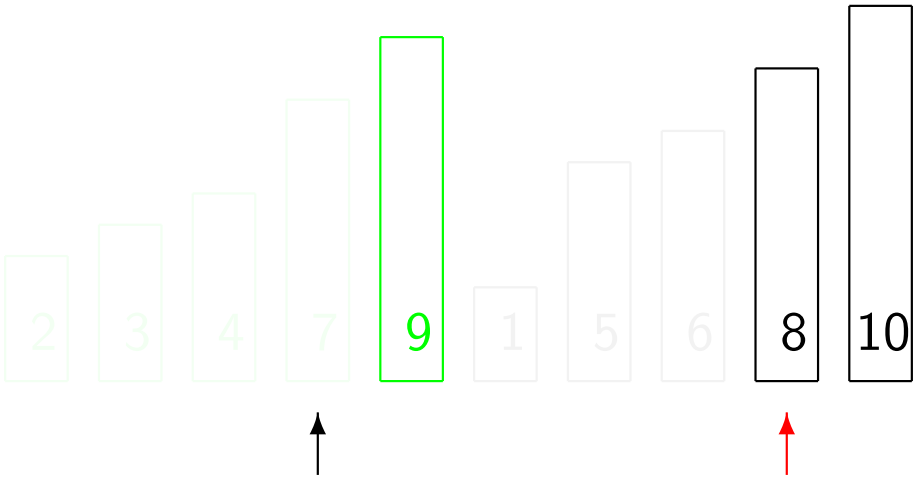
Intercalando



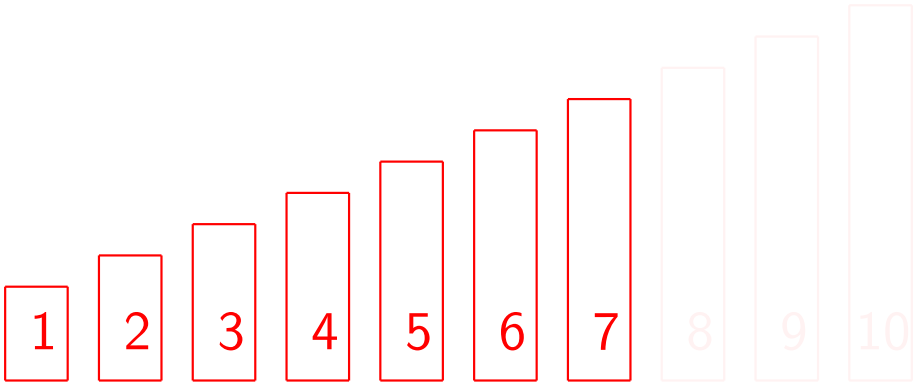
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



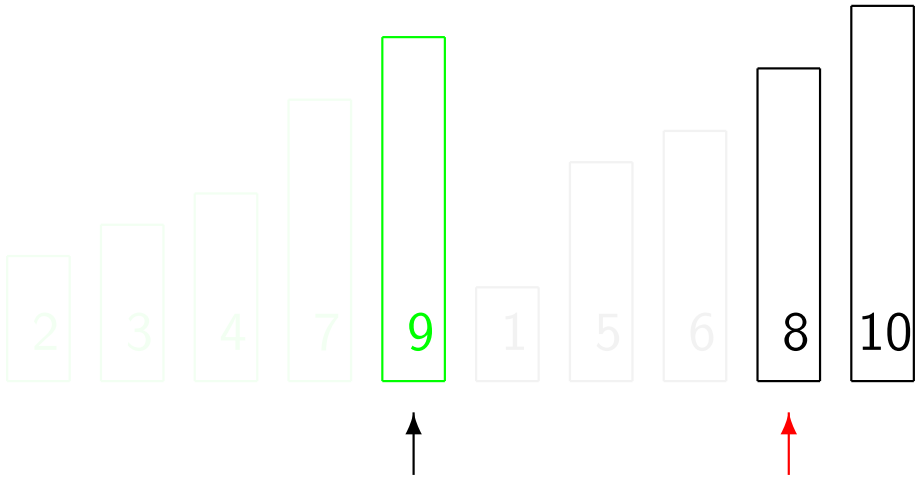
Intercalando



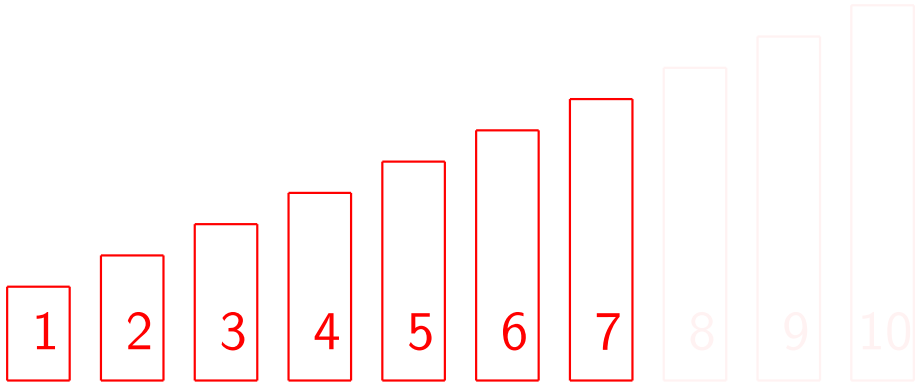
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



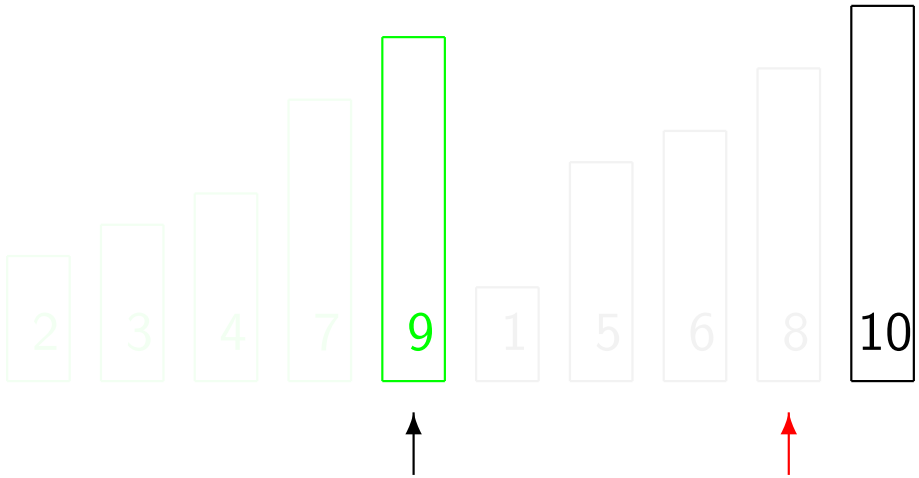
Intercalando



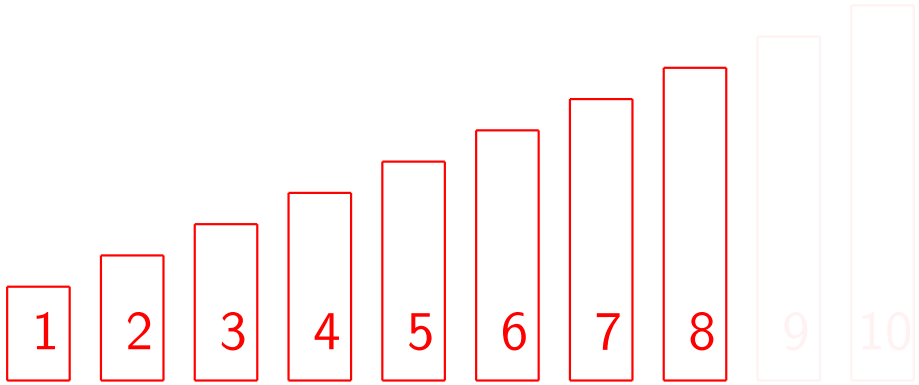
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



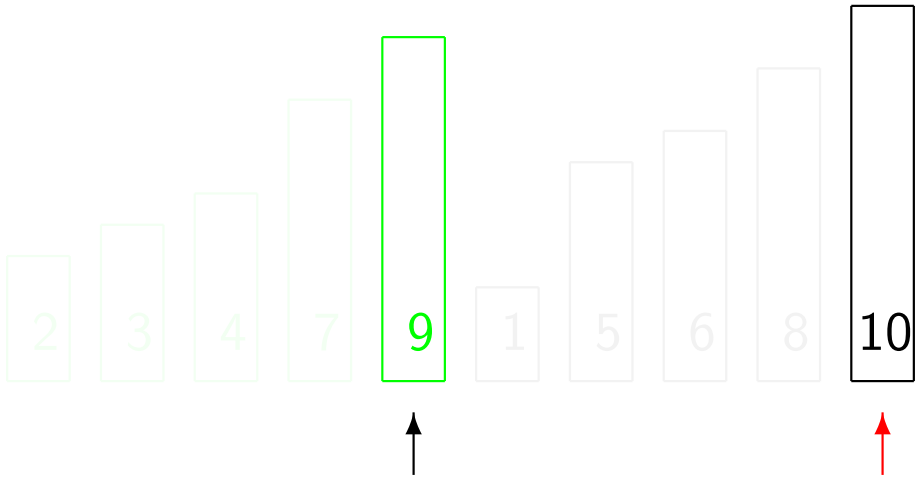
Intercalando



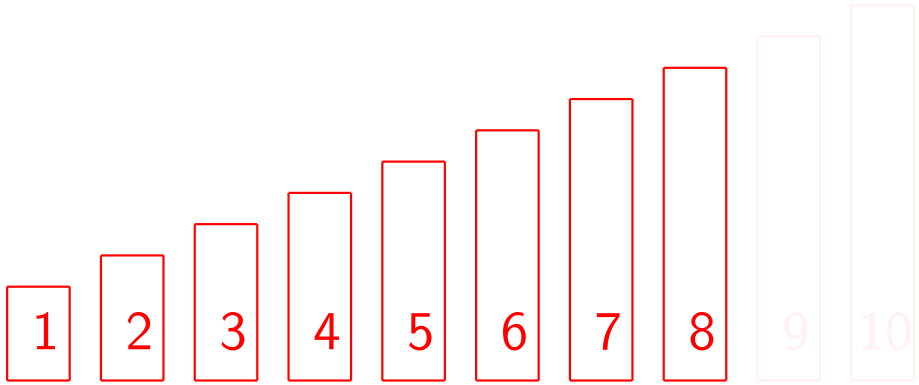
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



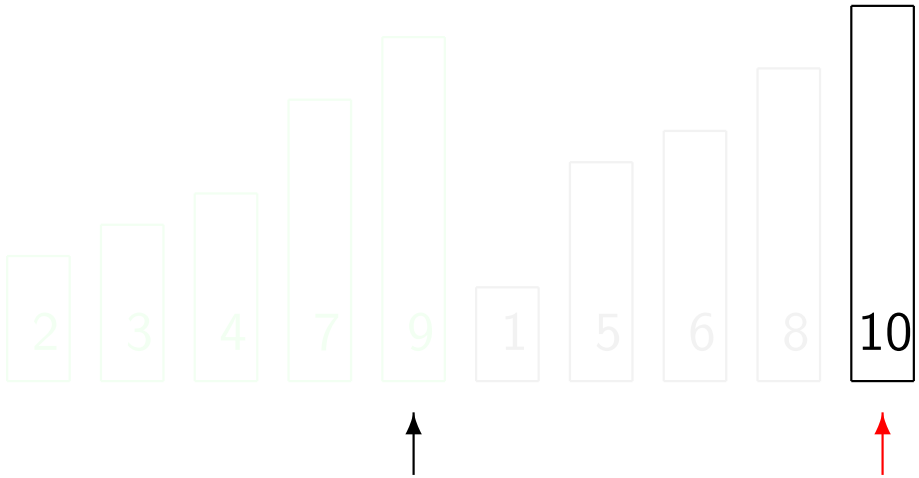
Intercalando



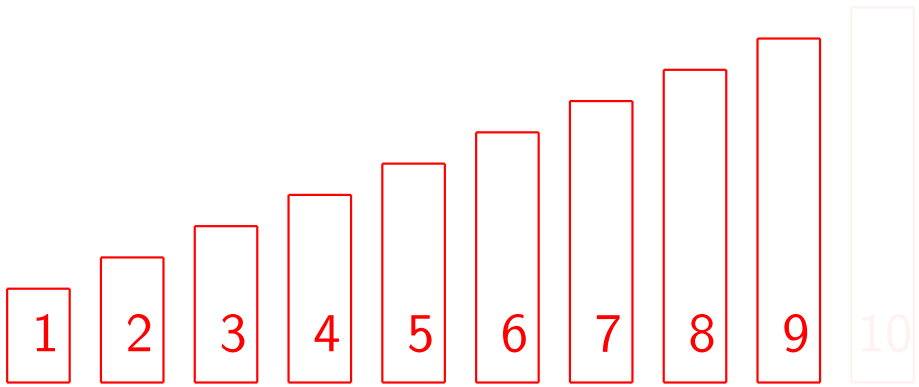
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



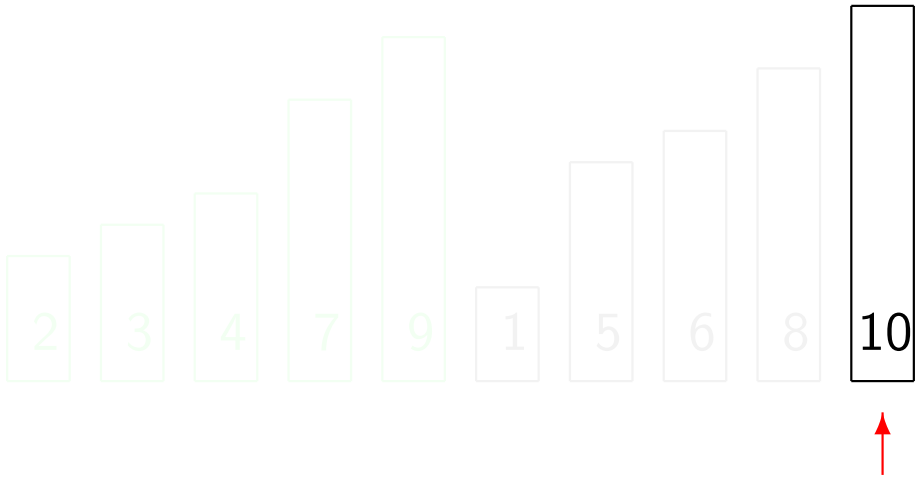
Intercalando



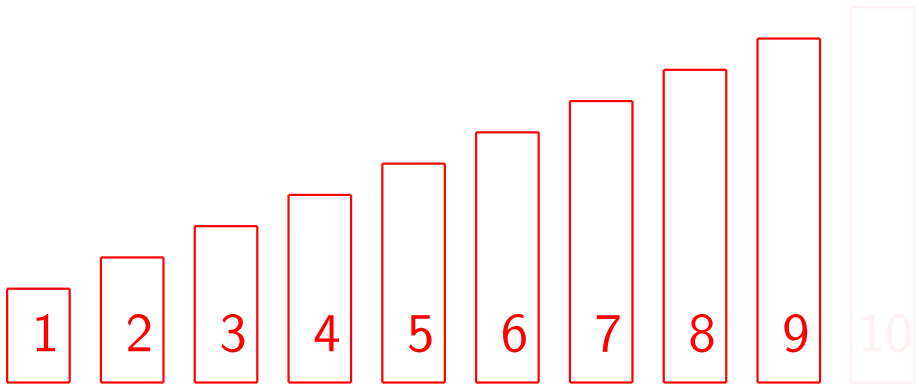
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



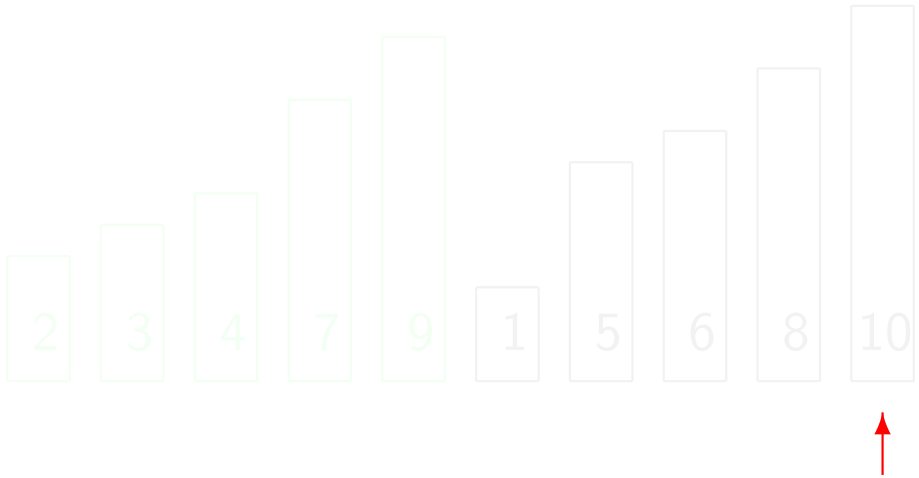
Intercalando



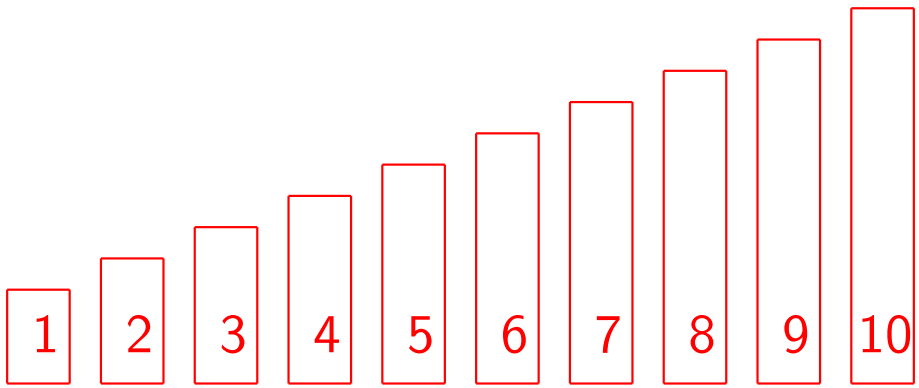
- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



Intercalando

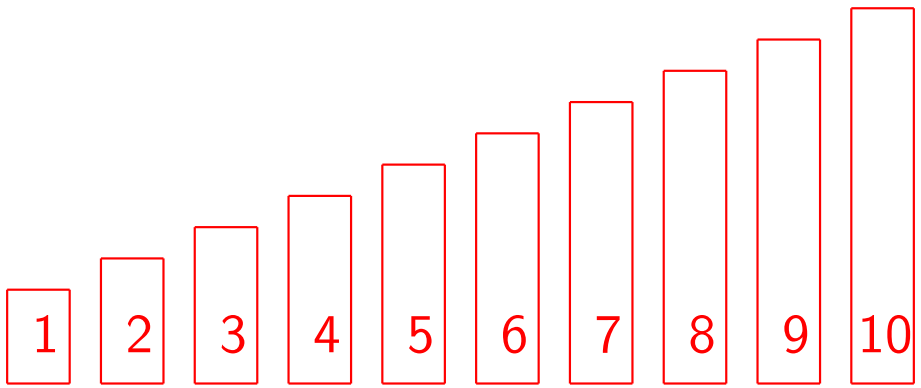


- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



Intercalando

- Percorremos os dois subvetores,
- pegamos o menor e inserimos no novo vetor e continuamos
- Depois movemos o resto.



Divisão e conquista

Observação

- A recursão parte do princípio que é mais fácil resolver problemas menores
- Para certos problemas, podemos dividi-lo em duas ou mais partes

Divisão e conquista

Com isso podemos resolver o problema em duas partes:

- **Divisão:** Quebramos um problema em vários subproblemas menores
- **Conquista:** Combinamos a solução dos problemas menores

Divisão e conquista

Observação

- A recursão parte do princípio que é mais fácil resolver problemas menores
- Para certos problemas, podemos dividi-lo em duas ou mais partes

Divisão e conquista

Com isso podemos resolver o problema em duas partes:

- **Divisão:** Quebramos um problema em vários subproblemas menores
- **Conquista:** Combinamos a solução dos problemas menores

Divisão e conquista

Observação

- A recursão parte do princípio que é mais fácil resolver problemas menores
- Para certos problemas, podemos dividi-lo em duas ou mais partes

Divisão e conquista

Com isso podemos resolver o problema em duas partes:

- **Divisão:** Quebramos um problema em vários subproblemas menores
- **Conquista:** Combinamos a solução dos problemas menores

Divisão e conquista

Observação

- A recursão parte do princípio que é mais fácil resolver problemas menores
- Para certos problemas, podemos dividi-lo em duas ou mais partes

Divisão e conquista

Com isso podemos resolver o problema em duas partes:

- **Divisão:** Quebramos um problema em vários subproblemas menores
- **Conquista:** Combinamos a solução dos problemas menores

Divisão e conquista

Observação

- A recursão parte do princípio que é mais fácil resolver problemas menores
- Para certos problemas, podemos dividi-lo em duas ou mais partes

Divisão e conquista

Com isso podemos resolver o problema em duas partes:

- **Divisão:** Quebramos um problema em vários subproblemas menores
- **Conquista:** Combinamos a solução dos problemas menores

Divisão e conquista

Observação

- A recursão parte do princípio que é mais fácil resolver problemas menores
- Para certos problemas, podemos dividi-lo em duas ou mais partes

Divisão e conquista

Com isso podemos resolver o problema em duas partes:

- **Divisão:** Quebramos um problema em vários subproblemas menores
- **Conquista:** Combinamos a solução dos problemas menores

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para a intercalação

- Os dois subvetores estão armazenados em `vetor`:
  - O primeiro nas posições de `ini` até `meio`
  - O segundo nas posições de `meio + 1` até `fim`
- Precisamos de um vetor auxiliar do tamanho do vetor
- Vamos considerar que o maior vetor tem tamanho `MAX`
  - Exemplo `#define MAX 100`

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para a intercalação

- Os dois subvetores estão armazenados em `vetor`:
  - O primeiro nas posições de `ini` até `meio`
  - O segundo nas posições de `meio + 1` até `fim`
- Precisamos de um vetor auxiliar do tamanho do vetor
- Vamos considerar que o maior vetor tem tamanho `MAX`
  - Exemplo `#define MAX 100`

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para a intercalação

- Os dois subvetores estão armazenados em `vetor`:
  - O primeiro nas posições de `ini` até `meio`
  - O segundo nas posições de `meio + 1` até `fim`
- Precisamos de um vetor auxiliar do tamanho do vetor
- Vamos considerar que o maior vetor tem tamanho `MAX`
  - Exemplo `#define MAX 100`

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para a intercalação

- Os dois subvetores estão armazenados em `vetor`:
  - O primeiro nas posições de `ini` até `meio`
  - O segundo nas posições de `meio + 1` até `fim`
- Precisamos de um vetor auxiliar do tamanho do vetor
- Vamos considerar que o maior vetor tem tamanho `MAX`
  - Exemplo `#define MAX 100`

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para a intercalação

- Os dois subvetores estão armazenados em `vetor`:
  - O primeiro nas posições de `ini` até `meio`
  - O segundo nas posições de `meio + 1` até `fim`
- Precisamos de um vetor auxiliar do tamanho do vetor
- Vamos considerar que o maior vetor tem tamanho `MAX`
  - Exemplo `#define MAX 100`

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para a intercalação

- Os dois subvetores estão armazenados em `vetor`:
  - O primeiro nas posições de `ini` até `meio`
  - O segundo nas posições de `meio + 1` até `fim`
- Precisamos de um vetor auxiliar do tamanho do vetor
- Vamos considerar que o maior vetor tem tamanho `MAX`
  - Exemplo `#define MAX 100`

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Intercalar subvetores

```
int intercalar(int vetor[], int ini, int meio, int fim) {
    int auxiliar[MAX];                  // vetor auxiliar
    int i = ini, j = meio + 1, k = 0; // índices dos vetores

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }
    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++, k++)
        vetor[i] = auxiliar[k];
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Algoritmo de ordenação por intercalação (*Merge-Sort*)

Convenções para ordenação

- Recebemos um `vetor` de tamanho n com limites:
  - O vetor começa na posição `vetor[ini]`
  - O vetor termina na posição `vetor[fim]`
- Dividimos o vetor em dois subvetores de tamanho $\frac{n}{2}$.
- O caso base é um vetor de tamanho 0 ou 1.

Ordenação por intercalação

```
void ordenar_intercalacao(int vetor[], int ini, int fim) {
    int meio;

    if (ini < fim) {
        meio = (ini + fim) / 2;
        ordenar_intercalacao(vetor, ini, meio);
        ordenar_intercalacao(vetor, meio + 1, fim);
        intercalar(vetor, ini, meio, fim);
    }
}
```

Ordenação por intercalação - Exemplo

```
#include "stdio.h"

#define MAX 100

void intercalar(int vetor[], int ini, int meio, int fim);
void ordenar_intercalacao(int vetor[], int ini, int fim);

int main() {
    int i;
    int vetor[] = { 4, 5, 1, 0, 7, 6, 3, 2 };

    ordenar_intercalacao(vetor, 0, 7);

    for (i = 0; i < 8; i++)
        printf("%d\n", vetor[i]);
}
```

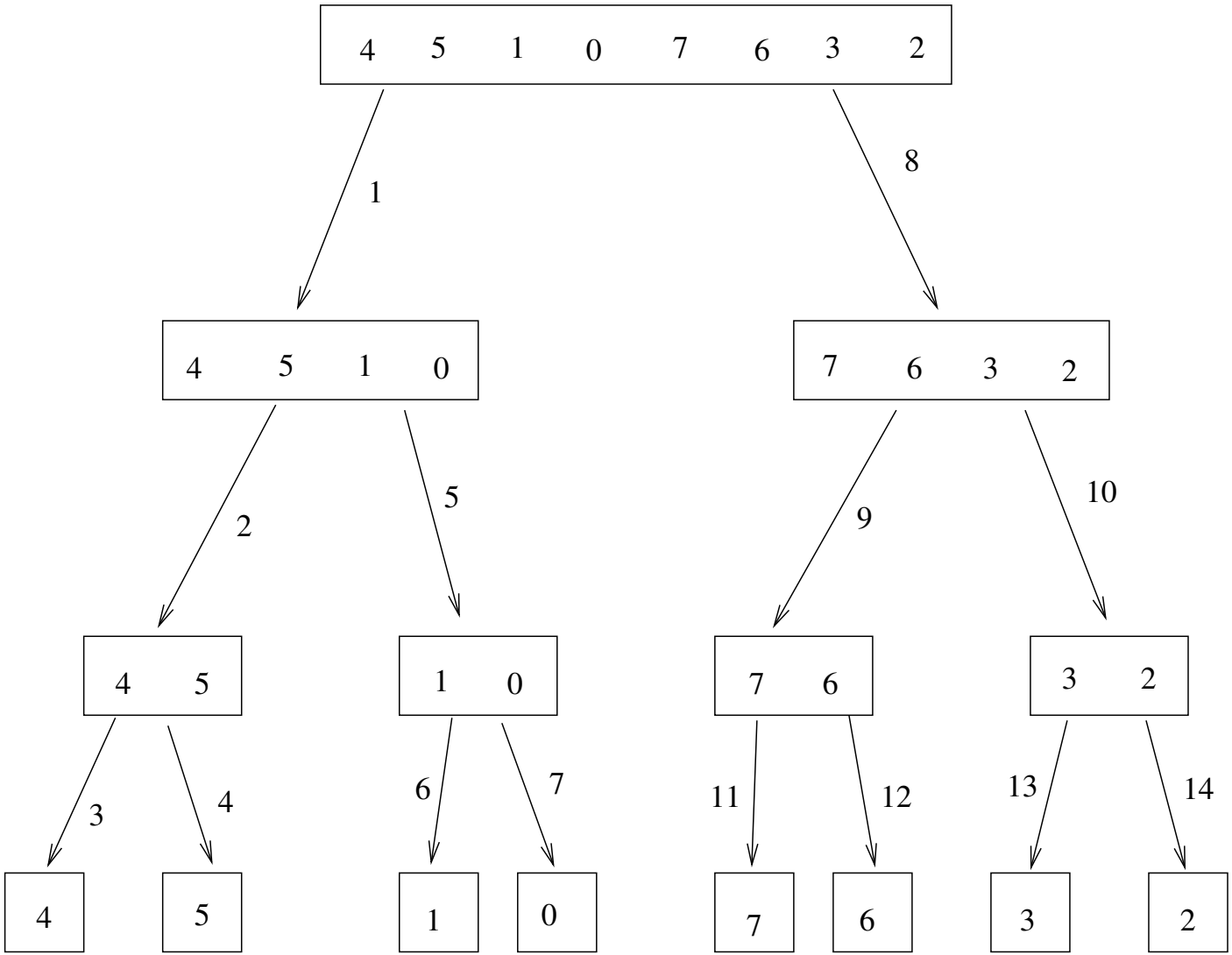
Repare que como podemos inicializar um vetor em C com constantes!

Ordenação por intercalação - Exemplo

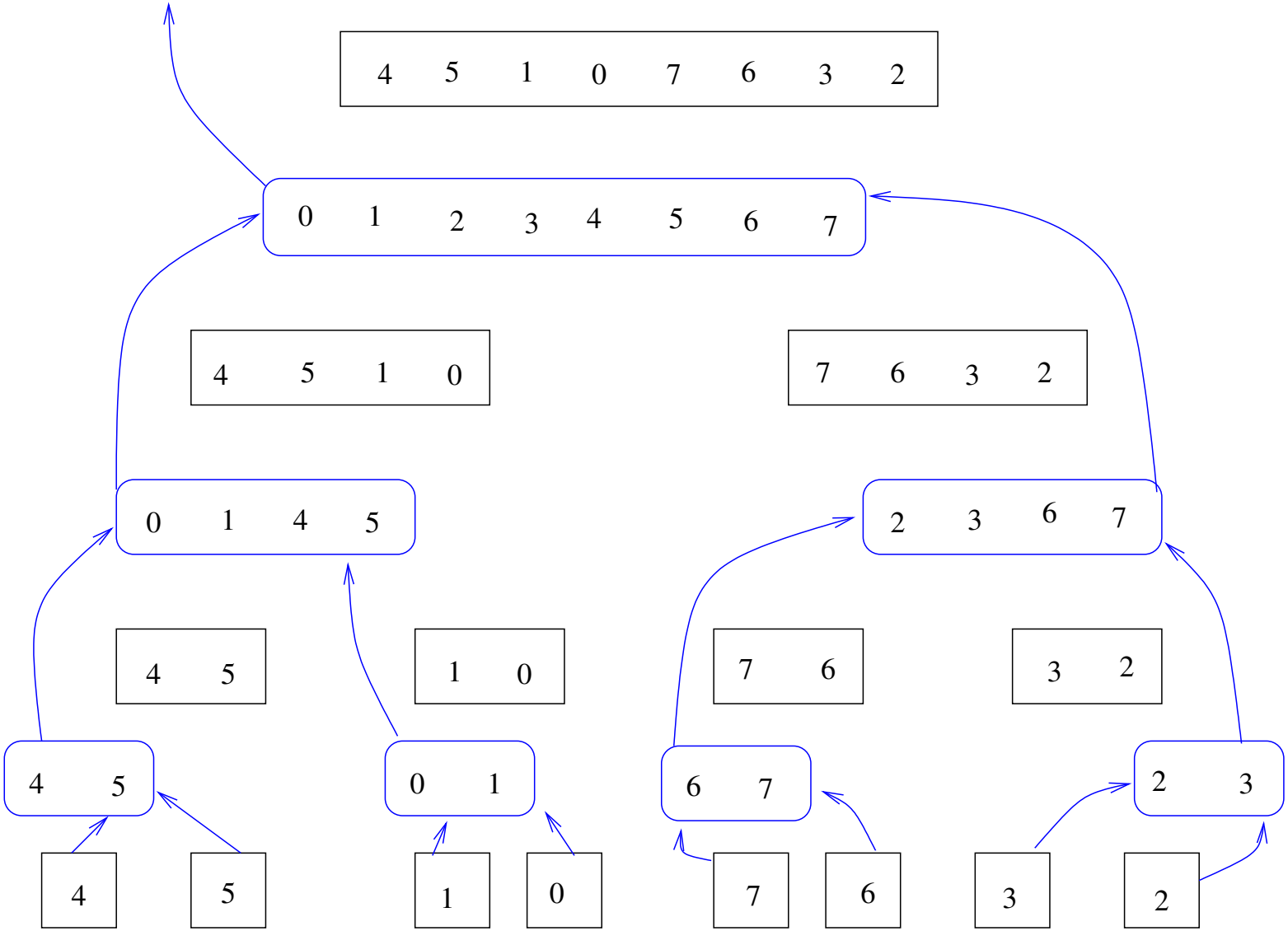
```
#include "stdio.h"

#define MAX 100

void intercalar(int vetor[], int ini, int meio, int fim);
void ordenar_intercalacao(int vetor[], int ini, int fim);

int main() {
    int i;
    int vetor[] = { 4, 5, 1, 0, 7, 6, 3, 2 };

    ordenar_intercalacao(vetor, 0, 7);

    for (i = 0; i < 8; i++)
        printf("%d\n", vetor[i]);
}
```

Repare que como podemos inicializar um vetor em C com constantes!

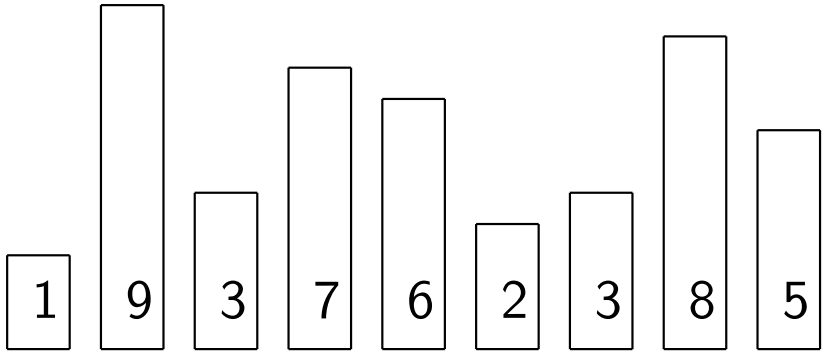
Ordenação por intercalação - Chamadas



Ordenação por intercalação - Retornos



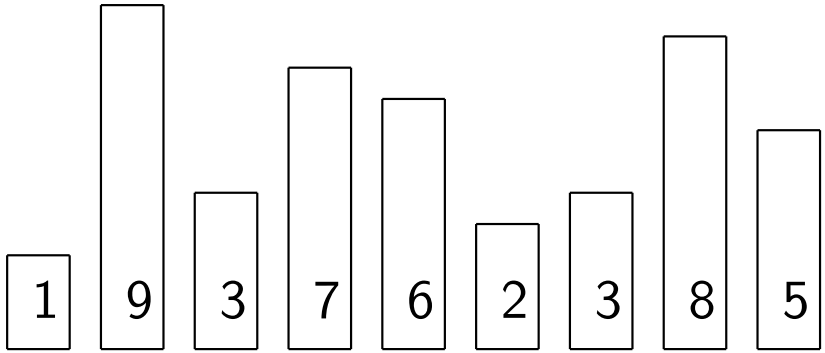
Introdução - Ordenação por Particionamento



Problema 1

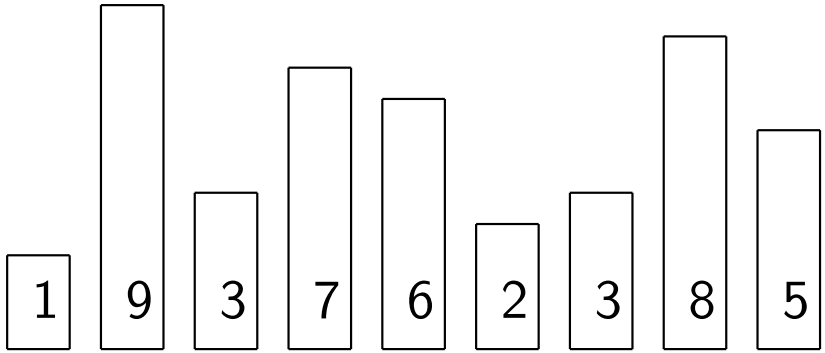
Suponha que temos um vetor desordenado com 10 números. Como fazer com que números *pequenos* (menores que 5) fiquem antes dos números *grandes* (maiores que 5)?

Considere a função

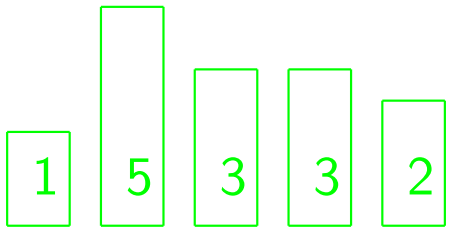


- `int particionar(int vetor[], int ini, int fim)`

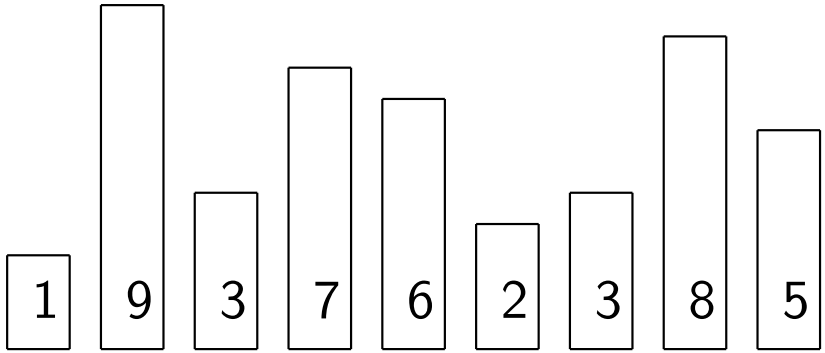
Considere a função



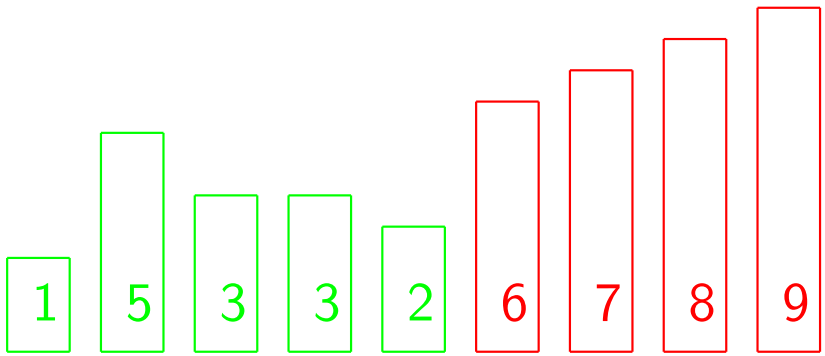
- `int particionar(int vetor[], int ini, int fim)`
  - a primeira parte do vetor contém elementos **"pequenos"**
  - a segunda parte do vetor contém elementos **"grandes"**



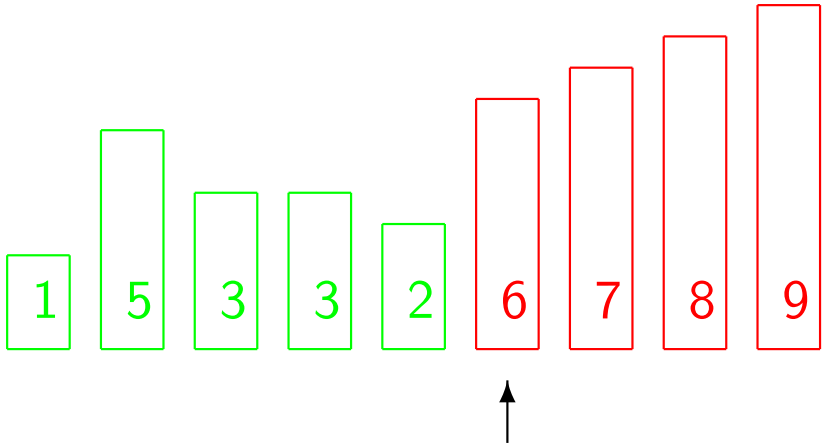
Considere a função



- `int particionar(int vetor[], int ini, int fim)`
  - a primeira parte do vetor contém elementos **"pequenos"**
  - a segunda parte do vetor contém elementos **"grandes"**



Combinando



Problema 2

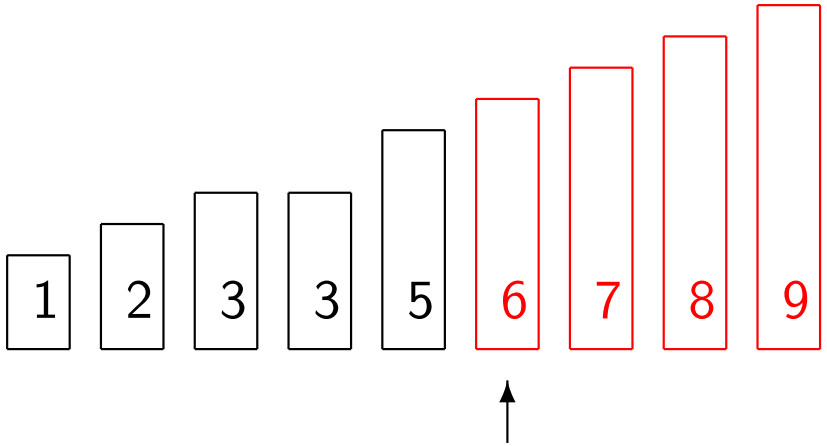
Suponha que o subvetor

- da posição **pos** a **fim**: contenha apenas elementos grandes
- da posição **ini** a **pos - 1**: contenha apenas elementos pequenos

Como ordenar?

- Ordenamos recursivamente o primeiro subvetor
- Depois o segundo subvetor

Combinando



Problema 2

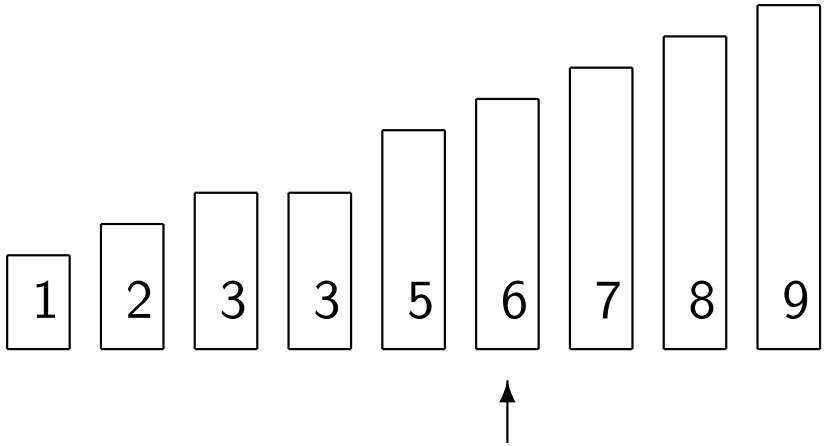
Suponha que o subvetor

- da posição **pos** a **fim**: contenha apenas elementos grandes
- da posição **ini** a **pos - 1**: contenha apenas elementos pequenos

Como ordenar?

- Ordenamos recursivamente o primeiro subvetor
- Depois o segundo subvetor

Combinando



Problema 2

Suponha que o subvetor

- da posição **pos** a **fim**: contenha apenas elementos grandes
- da posição **ini** a **pos - 1**: contenha apenas elementos pequenos

Como ordenar?

- Ordenamos recursivamente o primeiro subvetor
- Depois o segundo subvetor

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

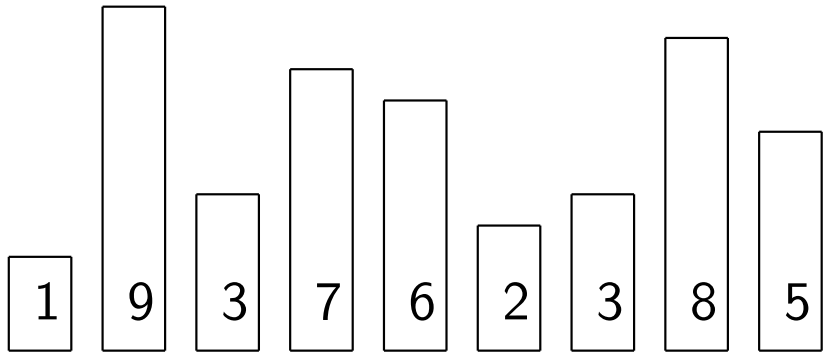
1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Algoritmo de particionamento

Particionar vetor

```
int particionar(int vetor[], int ini, int fim) {
    int pivo;

    pivo = vetor[fim];

    while (ini < fim) {
        while (ini < fim && vetor[ini] <= pivo)
            ini++;

        while (ini < fim && vetor[fim] > pivo)
            fim--;

        troca(&vetor[ini], &vetor[fim]);
    }

    return ini; // ini é a posição do primeiro elemento grande
}
```

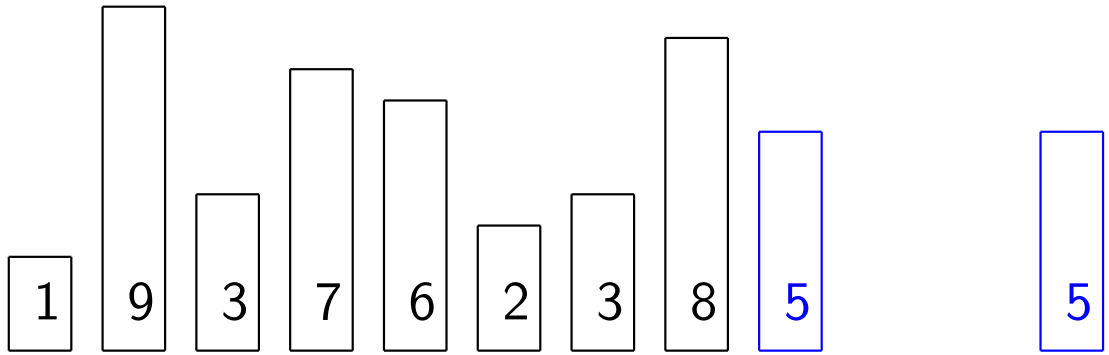
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - da direita para a esquerda em busca de elementos menores que o pivô;
   - da esquerda para a direita em busca de elementos maiores que o pivô;
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

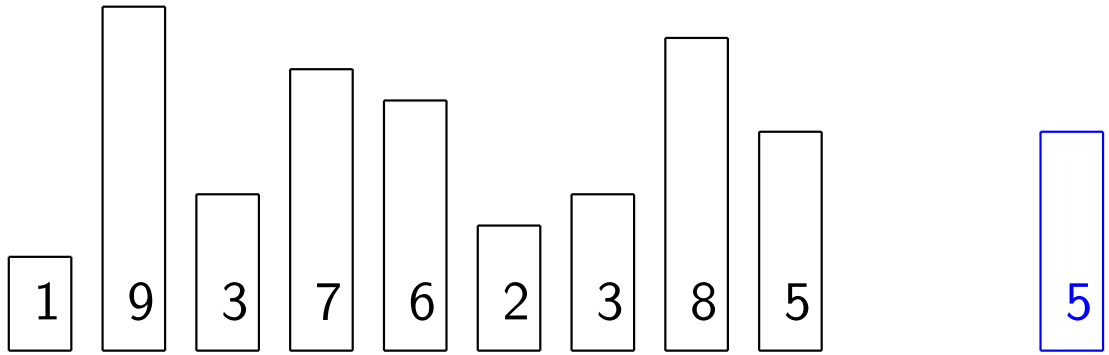
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - da direita para a esquerda em busca de elementos menores que o pivô
   - da esquerda para a direita em busca de elementos maiores que o pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

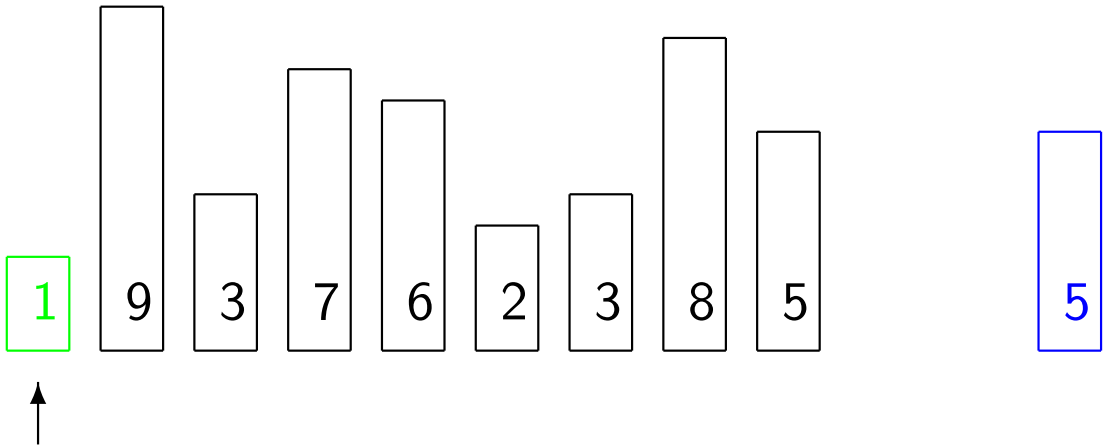
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

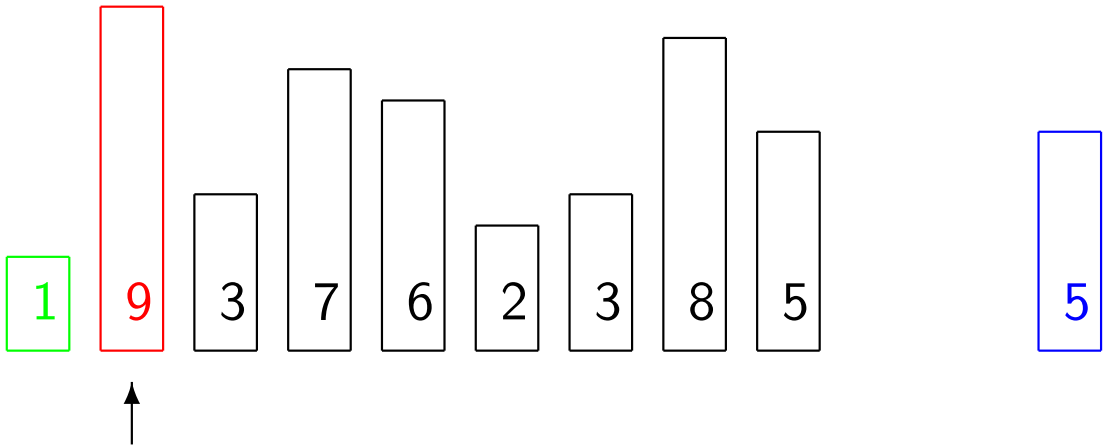
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

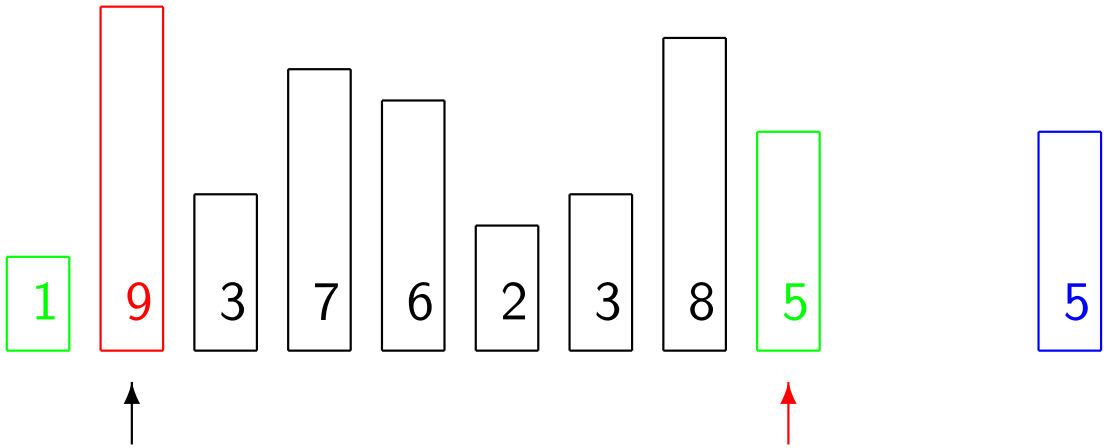
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

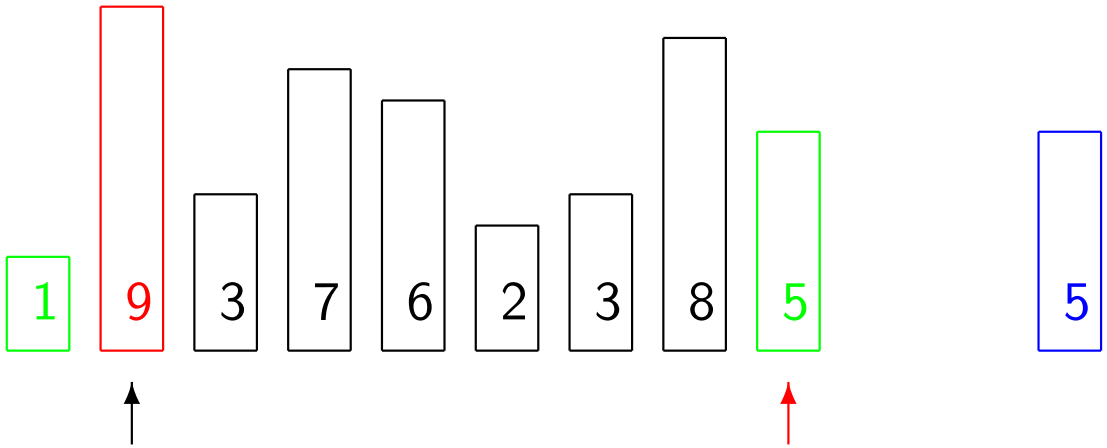
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

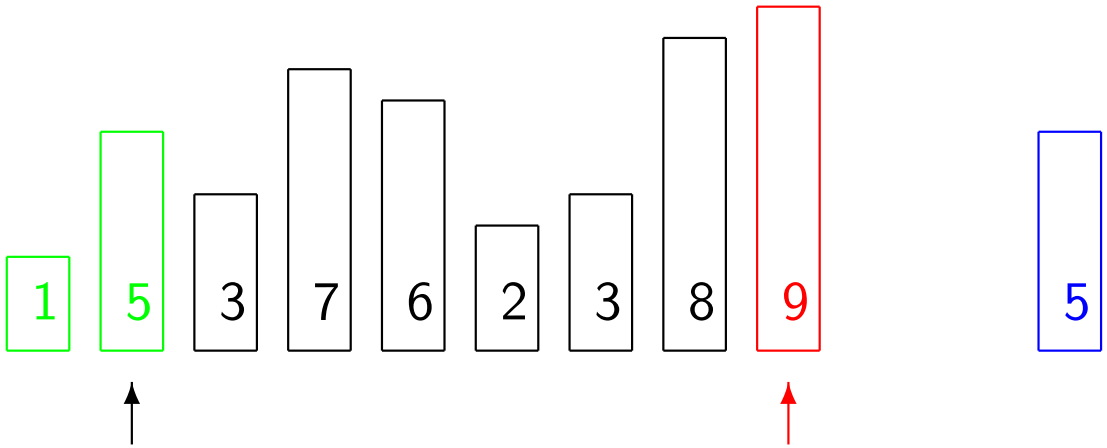
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

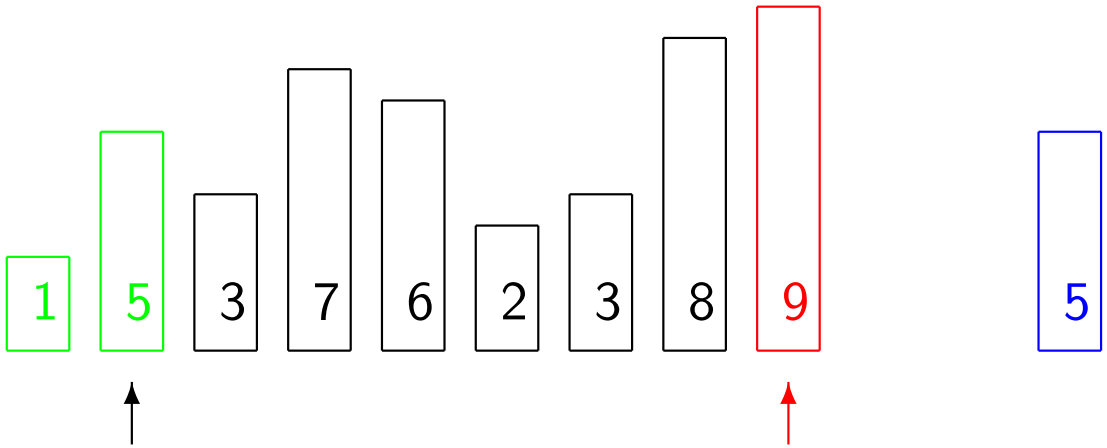
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

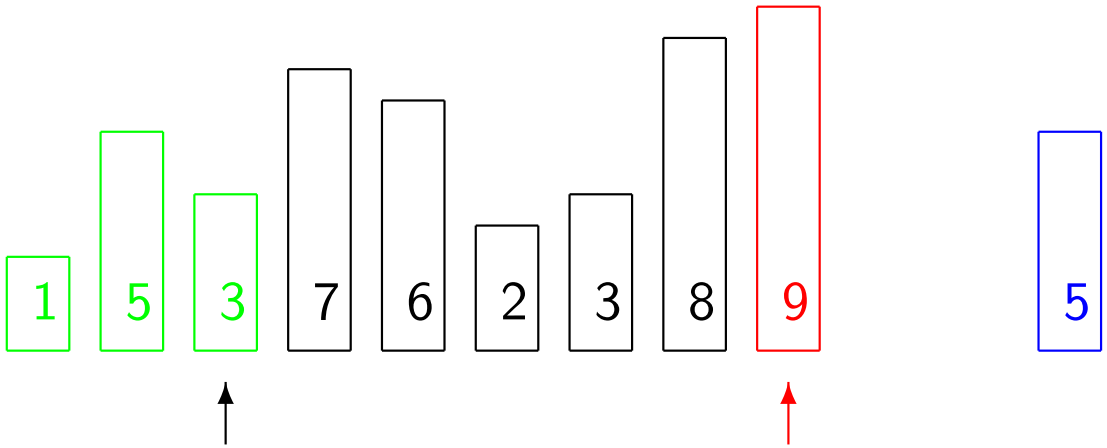
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

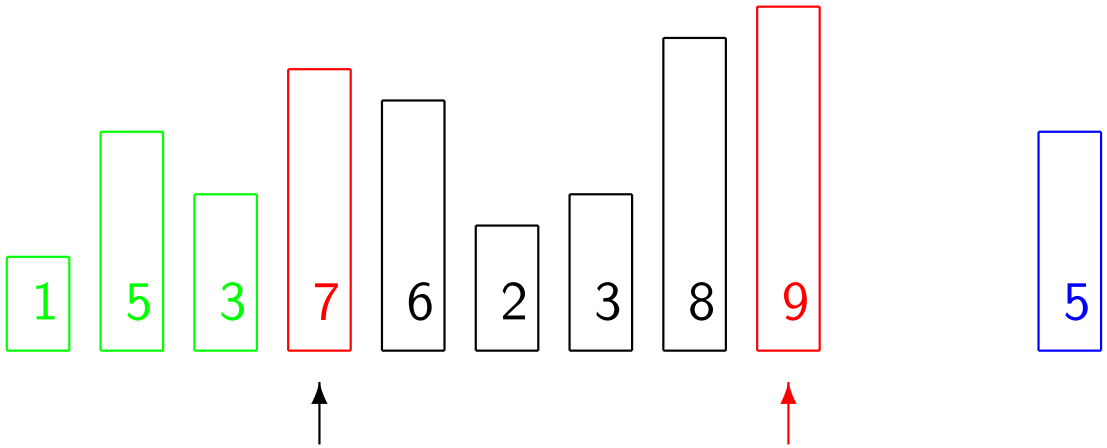
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

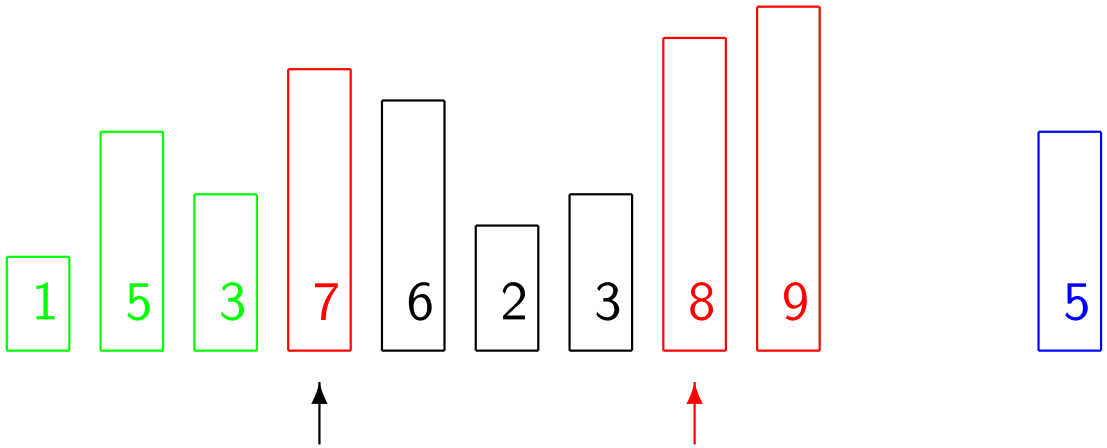
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

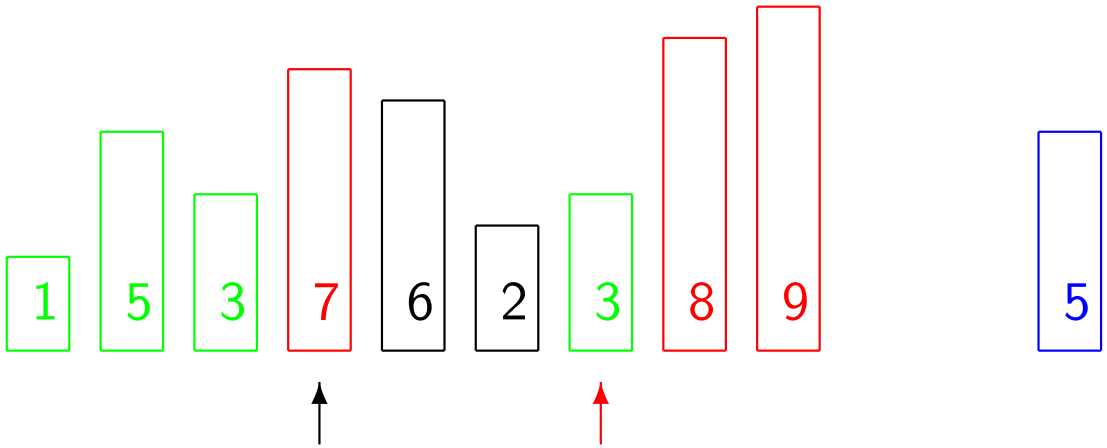
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

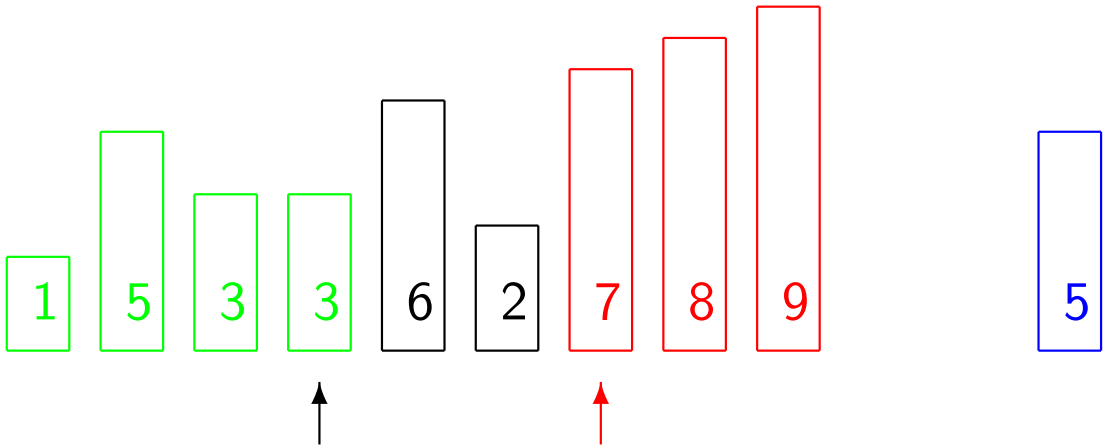
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

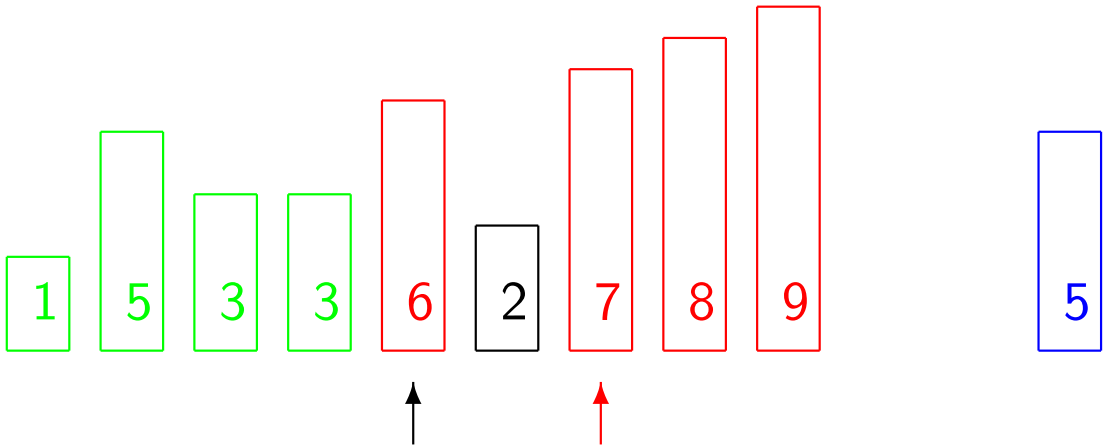
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

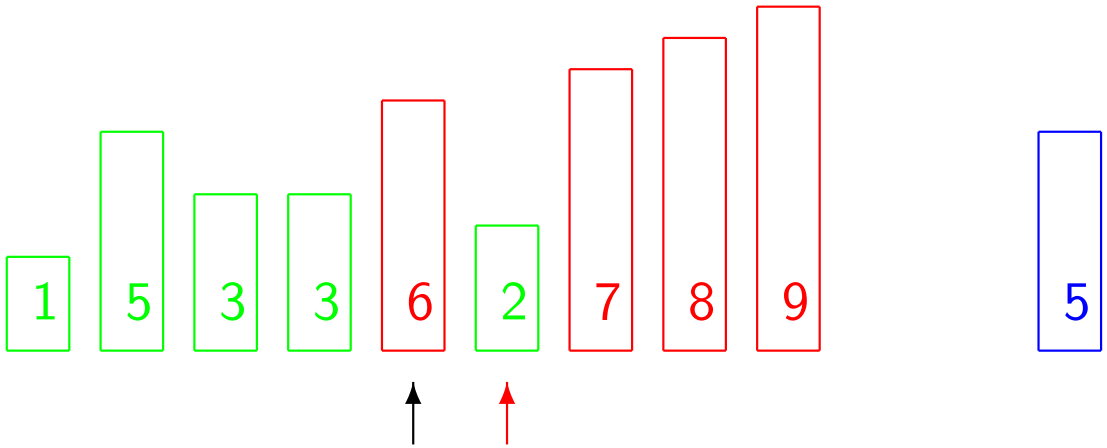
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

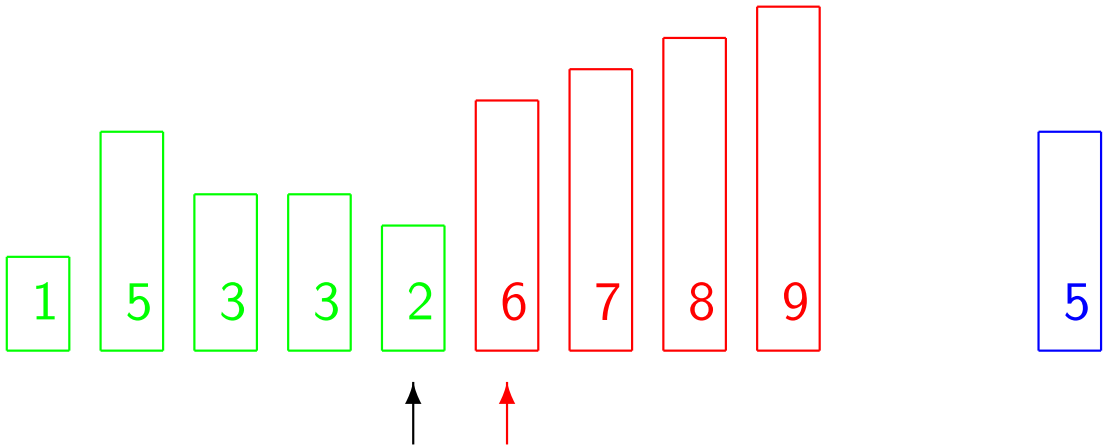
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

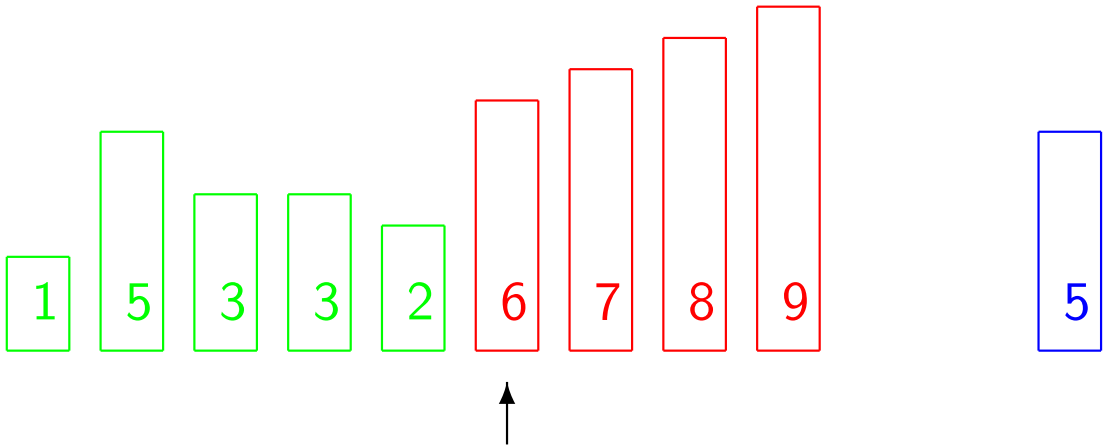
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

Exercício 1

1. Reescreva a função `ordenar_selecao` para que ela não utilize as funções auxiliares (`menor_elemento` e `trocar`).
2. Reescreva a função `ordenar_insercao` para que ela não utilize funções auxiliares. Na implementação acima, nós selecionamos a posição de inserção do elemento primeiro e depois deslocamos um subvetor. Mas podemos fazer as duas coisas de uma só vez. Qual a vantagem?
3. Na função `ordenar_selecao`, é realmente necessária a última iteração do laço de repetição? Por quê? E para a função `ordenar_insercao`?

Exercício 2

1. Escreva uma função para ordenar um vetor de Pessoa em ordem decrescente de idade e, havendo empate, em ordem crescente de nome.
2. Suponha que existe um vetor de pessoas. Queremos criar um vetor de mulheres ordenado por idade em ordem decrescente e, havendo empate, em ordem crescente de nome. Não queremos modificar o vetor original mas também não queremos desperdiçar espaço nem duplicar informação. Para isso: (i) crie um vetor de ponteiros para pessoas; (ii) modifique a questão 1 para que ela receba um vetor de ponteiros. Implemente essa estratégia e explique suas vantagens e como ela funciona.
3. Ao invés de usar ponteiros, você poderia usar índices para o vetor original. Explique as vantagens e desvantagens.

Ordenação da bolha

https://www.youtube.com/watch?v=lyZQPjUT5B4

Exercício 3 - Ordenação da bolha

Bubble-Sort

```
void ordenar_bolha(int vetor[], int n) {
    int i, mudou;
    do {
        mudou = 0;
        for (i = 1; i < n; i++) {
            if (vetor[i-1] > vetor[i]) {
                trocar(&vetor[i-1], &vetor[i]);
                mudou = 1;
            }
        }
    } while (mudou);
}
```

1. Explique o que faz e qual é a ideia do algoritmo.
2. Faça um teste de mesa para um vetor com elementos $(5, 4, 3, 2, 1)$ e para um vetor com elementos $(1, 4, 3, 2, 5)$. Conte as trocas.
3. Você consegue dizer por que o algoritmo tem esse nome? Por quê?

Exercício 4

Problema

Escreva uma função que recebe um vetor de inteiros ordenado decrescentemente e um número, realize uma busca binária e devolva a posição do número no vetor. Depois reescreva essa função de maneira **recursiva**.

Exercício 5

1. Aplique o algoritmo de particionamento sobre o vetor $(13, 19, 9, 5, 12, 21, 7, 4, 11, 2, 6, 6)$ com pivô igual a 6.
2. Modifique o algoritmo QuickSort para ordenar vetores em ordem decrescente.