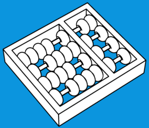
# Análise e correção de algoritmos

MO417 - Complexidade de Algoritmos I

03/24

4



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP

*"Uma demonstração de correção convincente será impossível enquanto o mecanismo seja visto como uma caixa preta, nossa única esperança é não considerar um mecanismo como uma caixa preta."* [1]

Edsger W. Dijkstra.

[1] Edsger W. Dijkstra, *"Notes On Structured Programming"*, 1970.

# Analisando um algoritmo

# Ordenação

Consideremos o problema da ordenação:

**Problem (Ordenação de inteiros)**

- ***Entrada:*** *Um vetor de inteiros.*
- ***Saída:*** *O vetor em ordem crescente.*

# Ordenação por inserção

Uma ideia para ordenar os elementos de um vetor pode ser:

- Percorrer as posições do vetor e a cada nova posição, inserir o elemento associado no lugar correto do sub-vetor (ordenado) à esquerda daquela posição.

Elaborado mais a ideia:

1. Percorrer cada posição **j** do vetor do início ao fim fazendo:
2. Armazenar o valor em **j** numa chave.
3. Percorrer de $\mathbf{j-1}$ até o início enquanto o valor for maior que a chave:
4. Copiar o valor atual na posição à direita.
5. Colocar o valor da chave na posição **i**.

Vejamos um exemplo:

## Algoritmo

**Algoritmo:** INSERTION-SORT$(A, n)$

**para** $j \leftarrow 2$ **até** $n$
    chave $\leftarrow A[j]$
    $i \leftarrow j - 1$
    **enquanto** $i \geq 1$ **e** $A[i] >$ chave
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow$ chave

Perguntas:

- O algoritmo termina?
- Qual a sua complexidade?
- Produz uma resposta correta?

# Contando o número de instruções

| INSERTION-SORT$(A, n)$ | Custo | Quantas vezes? |
|---|---|---|
| 1 **para** $j \leftarrow 2$ **até** $n$ | $c_1$ | $n$ |
| 2 chave $\leftarrow A[j]$ | $c_2$ | $n-1$ |
| 3 $i \leftarrow j-1$ | $c_3$ | $n-1$ |
| 4 **enquanto** $i \geq 1$ **e** $A[i] >$ chave | $c_4$ | $\sum_{j=2}^{n} t_j$ |
| 5 $A[i+1] \leftarrow A[i]$ | $c_5$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| 6 $i \leftarrow i-1$ | $c_6$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| 7 $A[i+1] \leftarrow$ chave | $c_7$ | $n-1$ |

- A linha $k$ executa um número constante de instruções $c_k$.
- Cada linha executa uma ou mais vezes.
- Quantas vezes a linha 4 executa depende da entrada.
  - $t_j$ denota quantas vezes o **enquanto** executa para um certo $j$.

## Tempo de execução total

- Considere uma instância de tamanho $n$.
- $T(n)$ denota o número de instruções executadas para ela.
- Basta somar para todas as linhas:

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \cdot \sum_{j=2}^{n} t_j + c_5 \cdot \sum_{j=2}^{n} (t_j - 1) + c_6 \cdot \sum_{j=2}^{n} (t_j - 1) + c_7 \cdot (n-1).$$

Observações:

- Entradas do mesmo tamanho podem ter tempos diferentes.
- Vamos considerar diferentes instâncias.
  - **Melhor caso:** quando $T(n)$ é o menor possível.
  - **Pior caso:** quando $T(n)$ é o maior possível.

## Melhor caso

Um melhor caso ocorre quando $t_j = 1$ para cada $j$:

- Basta que a condição do **enquanto** sempre falhe.
- Ocorre se a entrada $A$ já vem ordenada.

Nesse caso:

$$\begin{aligned} T(n) &= \mathbf{c_1} \cdot n + \mathbf{c_2} \cdot (n-1) + \mathbf{c_3} \cdot (n-1) + \mathbf{c_4} \cdot (n-1) + \mathbf{c_7} \cdot (n-1) \\ &= (\mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_4} + \mathbf{c_7}) \cdot n - (\mathbf{c_2} + \mathbf{c_3} + \mathbf{c_4} + \mathbf{c_7}) \\ &= \mathbf{a} \cdot n + \mathbf{b}. \end{aligned}$$

- Os valores de $a$ e $b$ são constantes.
- O tempo de execução no melhor caso é **linear** em $n$.
- O algoritmo executa em tempo $\Omega(n)$.

## Pior caso

Um pior caso ocorre quando $t_j$ é máximo para cada $j$:

- Basta que a condição do **enquanto** só falha quando $i = 0$.
- Nessa situação, teremos $t_j = j$.
- Ocorre se a entrada $A$ vem ordenada decrescentemente.

Relembre que:

- $\sum_{j=2}^{n} j = n(n+1)/2 - 1$ e $\sum_{j=2}^{n} (j-1) = n(n-1)/2$.

## Pior caso (cont)

Substituindo, temos:

$$
\begin{aligned}
T(n) &= c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \cdot (n(n+1)/2 - 1) + \\
&\qquad c_5 \cdot n(n-1)/2 + c_6 \cdot n(n-1)/2 + c_7 \cdot (n-1) \\
&= (c_4/2 + c_5/2 + c_6/2) \cdot n^2 + \\
&\qquad (c_1 + c_2 + c_3 + c_4/2 - c_5/2 - c_6/2 + c_7) \cdot n - \\
&\qquad (c_2 + c_3 + c_4 + c_7) \\
&= a \cdot n^2 + b \cdot n + c.
\end{aligned}
$$

- Os valores de $a, b, c$ são constantes.
- O tempo de execução no pior caso é **quadrático** em $n$.
- O algoritmo executa em tempo $\mathrm{O}(n^2)$.

## Pergunta

**Como verificar se o algoritmo produz uma resposta correta?**

# Correção por invariante de laço

# Invariante de laço

## Definição

*Uma **INVARIANTE LAÇO** é uma propriedade que:*

- *Depende dos valores das variáveis.*
- *Está associada a determinada posição de um laço.*
- *É satisfeita em **TODA** execução do laço.*

A posição escolhida é normalmente descrita como:

- Imediatamente **ANTES** ou **DEPOIS** da iteração do laço.
- Imediatamente **ANTES** ou **DEPOIS** de determinada linha.

Objetivos:

- Após o término do laço, deve ser uma propriedade útil para se mostrar a correção do algoritmo.
- Permite nos concentrar apenas em uma iteração do laço.

# Exemplo de invariante

**Algoritmo:** INSERTION-SORT$(A, n)$

```
para j ← 2 até n
    chave ← A[j]
    i ← j − 1
    enquanto i ≥ 1 e A[i] > chave
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← chave
```

## Exemplo (Invariante 1)

*Imediatamente antes de cada iteração do laço* **para**, *o subvetor* $A[1 \dots j-1]$ *está ordenado.*

- **Posição da invariante:** antes da iteração do laço **para**.
- **Propriedade invariante:** $A[1 \dots j-1]$ está ordenado.

## Demonstrando uma invariante

Tipicamente, demonstramos uma invariante com as seguintes etapas:

1. Mostre que a propriedade vale antes de qualquer iteração.
2. Mostre que, se a propriedade vale no início da iteração, então ela também vale no final da iteração
3. Conclua que a invariante vale quando o laço termina.

Estamos usando o **PRINCÍPIO DA INDUÇÃO**!

- A **base** corresponde à etapa 1.
- O **passo indutivo** corresponde à etapa 2.

## Demonstrando uma invariante: caso base

Considere a **primeira iteração** do laço **para**:

- No início da iteração, $j = 2$.
- Assim, o subvetor $A[1 \ldots j-1]$ contém apenas um elemento.
- Então, a invariante vale antes de qualquer iteração.

## Demonstrando uma invariante: passo indutivo

Suponha que a invariante vale no início de **alguma iteração**:

- Nessa iteração, temos chave $= A[j]$.
- Após o laço **enquanto**, inserimos chave na posição $i + 1$.
- Quando inserirmos a chave, queremos:
  1. Que os anteriores sejam menores e estejam ordenados.
  2. Que os posteriores sejam maiores e estejam ordenados.
- Vamos criar uma **sub-invariante** para o laço **enquanto**!

## Sub-invariante

### Exemplo (Sub-invariante)

*Imediatamente antes de cada iteração do laço* **enquanto***:*

1. $A[i+1\ldots j]$ *está ordenado.*
2. chave $\leq A[i+1]$.

Demonstração:

- Antes de qualquer iteração, as afirmações valem.
- Suponha que valem no início de uma iteração:
  - Pela condição do laço, chave $< A[i]$ e $A[i]$ não se altera.
  - Como diminuímos o valor de $A[i+1]$ para $A[i]$, o vetor $A[i+1\ldots j]$ continua ordenado.
- Assim, as afirmações mantêm-se no final.

## Demonstrando uma invariante: passo indutivo (cont)

Quando o laço **enquanto** termina:

- Pela sub-invariante, o vetor $A[i+1 \ldots j]$ está ordenado.
- Também pela sub-invariante, chave $\leq A[i+1]$.
- Assim, após fazer $A[i+1] \leftarrow$ chave, $A[i+1 \ldots j]$ continua ordenado.
- Se paramos porque $i = 0$, $A[1 \ldots j]$ está ordenado no final do laço **para**.
- Do contrário, paramos porque $A[i] <$ chave e $A[i \ldots j]$ está ordenado no final do laço **para**:
  - Como $i < j$, pela invariante $A[1 \ldots i]$ está ordenado no início do laço e como esses valores não são alterados durante a iteração, $A[1 \ldots i]$ se mantém ordenado.
  - Como $A[1 \ldots i]$ e $A[i \ldots j]$ estão ordenados no final do **para**, temos que $A[1 \ldots j]$ está ordenado no final do laço.
- Em qualquer caso, a invariante vale quando o laço **para** termina.

## Outra invariante

- Já demonstramos a Invariante 1.
- O laço termina apenas quando $j = n + 1$.
- Pela invariante, nesse instante $A[1 \dots n]$ está ordenado.
- Mas isso não é suficiente, pois o vetor poderia ser:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |

**Exemplo (Invariante 2)**

*Imediatamente antes de cada iteração do laço* **para***, o subvetor $A[1 \dots n]$ é uma permutação dos dados da entrada.*

**Exercício:** Demonstre essa invariante.

## Demonstrando a correção do algoritmo

Suponha que já demonstramos as Invariantes 1 e 2.

**Teorema**

*O algoritmo* INSERTION-SORT *está correto.*

Demonstração:

- Quando o laço termina, temos $j = n + 1$, então a Invariante 1 implica que $A[1 \dots n]$ está ordenado.
- Nesse instante, a Invariante 2 implica que o vetor $A[1 \dots n]$ contém todos elementos da entrada.
- Portanto, o vetor devolvido é uma ordenação do vetor de entrada.

# Conversão binária

# Conversão para representação binária

## Problema (Conversão binária)

- ***Entrada:*** *Um número inteiro não negativo $n$.*
- ***Saída:*** *Um vetor $B$ com representação binária (invertida) de $n$.*

**Algoritmo:** CONVERTE-BINARIO($n$)

```
t ← n
k ← 0
enquanto t > 0
    B[k] ← t mod 2
    t ← t div 2
    k ← k + 1
devolva B
```

# Invariante para CONVERTE-BINARIO

## Exemplo (Invariante)

*No início da iteração do laço* **enquanto***:*

$$n = t \cdot 2^k + \sum_{i=0}^{k-1} 2^i \cdot B[i].$$

Demonstração:

- Antes de qualquer iteração, temos $k = 0$ e $t = n$.
- Assim,
$$n = t \cdot 1 + 0 = t \cdot 2^k + \sum_{i=0}^{k-1} 2^i \cdot B[i].$$
- Portanto, a afirmação vale no início do laço.

## Invariante para CONVERTE-BINARIO (cont)

Suponha que a invariante vale no início da iteração:

- Sejam $k' = k + 1$ e $t' = t \operatorname{div} 2$.
- Sabemos que $B[k] = t \bmod 2$.
- Como a afirmação vale no início da iteração,

$$\begin{aligned} n &= \mathbf{t} \cdot \mathbf{2^k} + \textstyle\sum_{i=0}^{k-1} 2^i \cdot B[i] \\ &= (\mathbf{t} \operatorname{div} \mathbf{2}) \cdot \mathbf{2^{k+1}} + \mathbf{2^k} \cdot (\mathbf{t} \bmod \mathbf{2}) + \textstyle\sum_{i=0}^{k-1} 2^i \cdot B[i] \\ &= (t \operatorname{div} 2) \cdot 2^{k+1} + 2^k \cdot B[k] + \textstyle\sum_{i=0}^{k-1} 2^i \cdot B[i] \\ &= (t \operatorname{div} 2) \cdot 2^{k+1} + \textstyle\sum_{i=0}^{k} 2^i \cdot B[i] \\ &= t' \cdot 2^{k'} + \textstyle\sum_{i=0}^{k'-1} 2^i \cdot B[i]. \end{aligned}$$

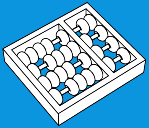
- Portanto, a invariante vale no final da iteração.

# Análise e correção de algoritmos

MO417 - Complexidade de Algoritmos I

03/24

4



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP