

ANÁLISE DE ALGORITMOS

MC458 - Projeto e Análise de
Algoritmos I

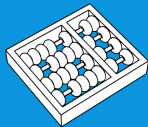
Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

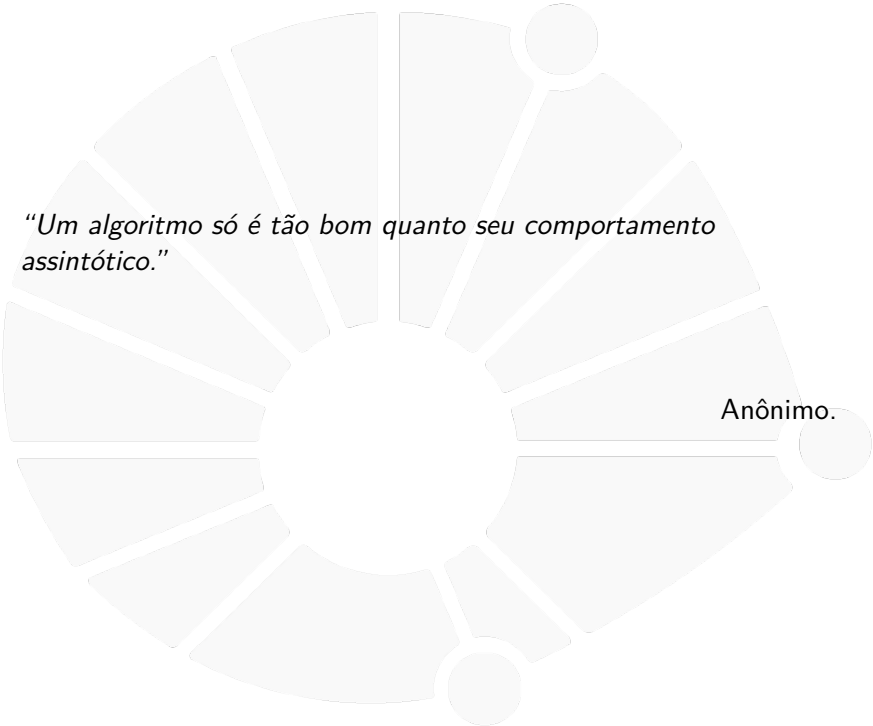
08/25

05



UNICAMP



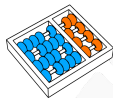


“Um algoritmo só é tão bom quanto seu comportamento assintótico.”

Anônimo.

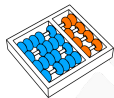


ANALISANDO UM ALGORITMO



Ordenação

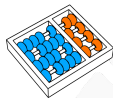
Consideremos o problema da ordenação:



Ordenação

Consideremos o problema da ordenação:

Problem (Ordenação de inteiros)

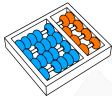


Ordenação

Consideremos o problema da ordenação:

Problem (Ordenação de inteiros)

► **Entrada:** *Um vetor de inteiros.*

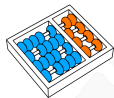


Ordenação

Consideremos o problema da ordenação:

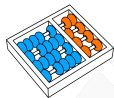
Problem (Ordenação de inteiros)

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



Ordenação por inserção

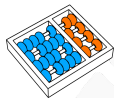
Uma ideia para ordenar os elementos de um vetor pode ser:



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.

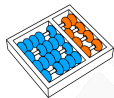


Ordenação por inserção

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:



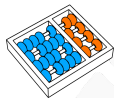
Ordenação por inserção

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 i do vetor do início ao fim fazendo:



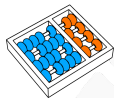
Ordenação por inserção

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 i do vetor do início ao fim fazendo:
2. Percorrer de i até o início enquanto o valor for maior que o da esquerda:



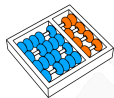
Ordenação por inserção

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

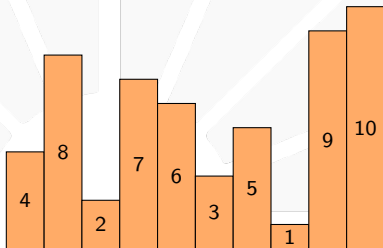
- ▶ Percorrer as posições do vetor e a cada nova posição, inserir o elemento associado no lugar correto do sub-vetor (ordenado) à esquerda daquela posição.

Elaborado mais a ideia:

1. Percorrer cada posição i do vetor do início ao fim fazendo:
2. Percorrer de i até o início enquanto o valor for maior que o da esquerda:
3. Trocar o valor atual com o da posição à esquerda.

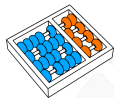


Ordenação por Inserção

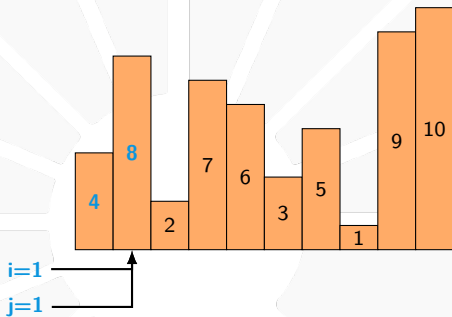


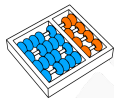
i

j

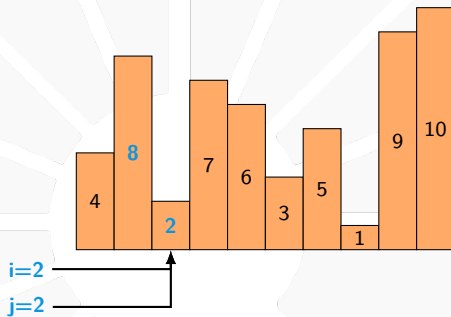


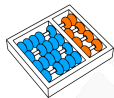
Ordenação por Inserção



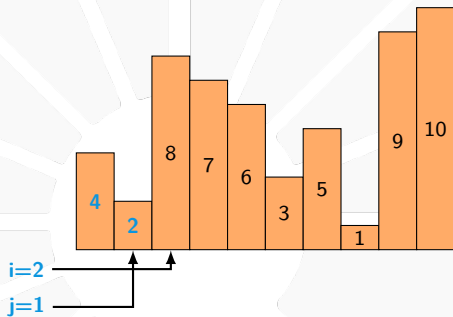


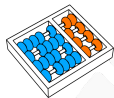
Ordenação por Inserção



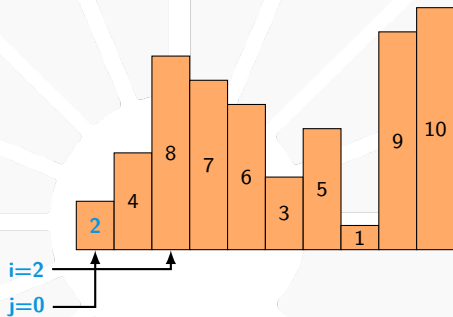


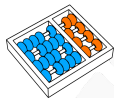
Ordenação por Inserção



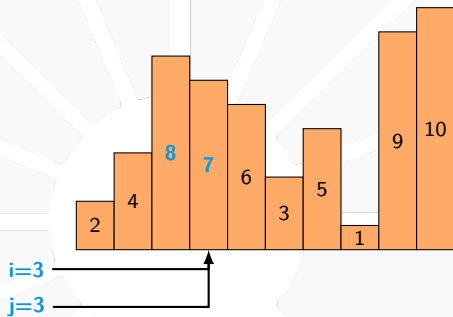


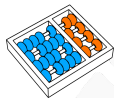
Ordenação por Inserção



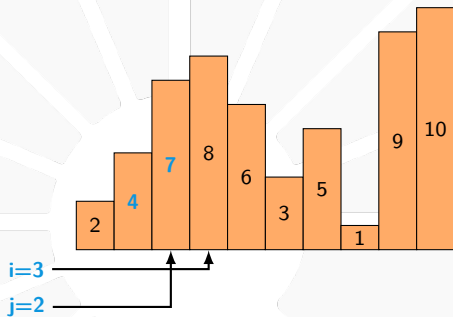


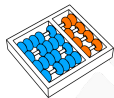
Ordenação por Inserção



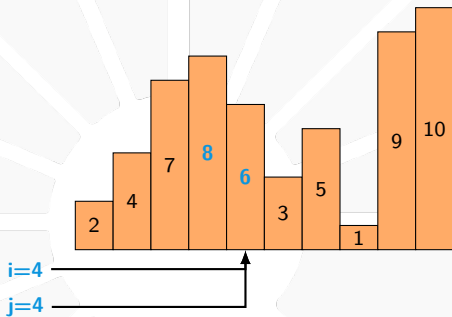


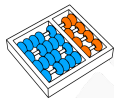
Ordenação por Inserção



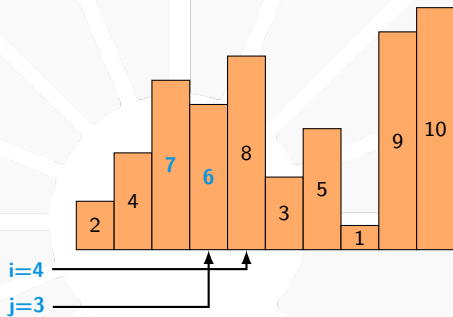


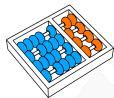
Ordenação por Inserção



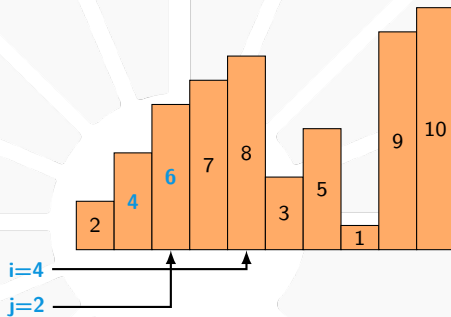


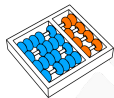
Ordenação por Inserção



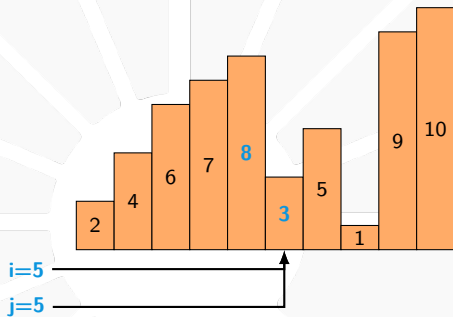


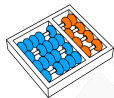
Ordenação por Inserção



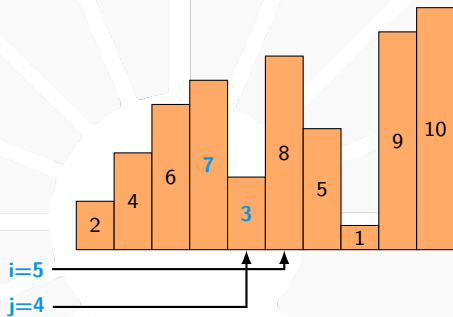


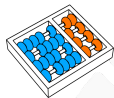
Ordenação por Inserção



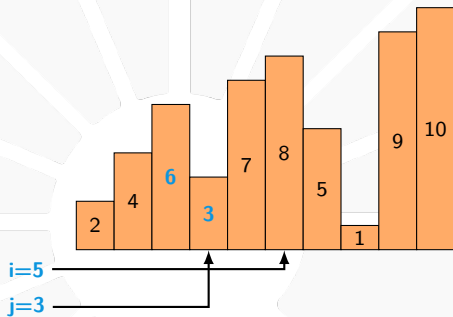


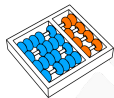
Ordenação por Inserção



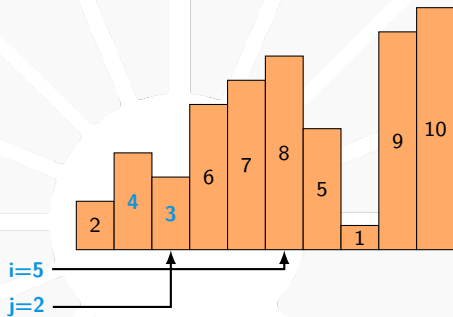


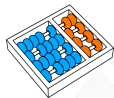
Ordenação por Inserção



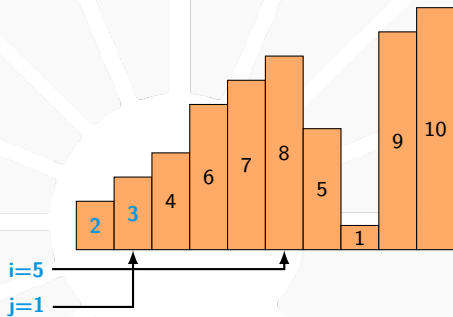


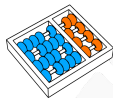
Ordenação por Inserção



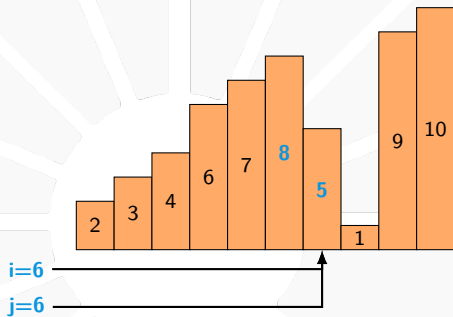


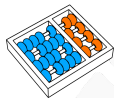
Ordenação por Inserção



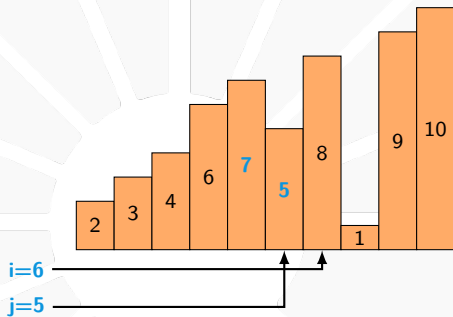


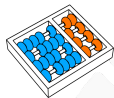
Ordenação por Inserção



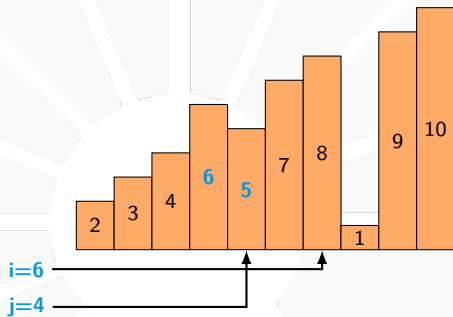


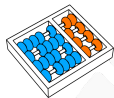
Ordenação por Inserção



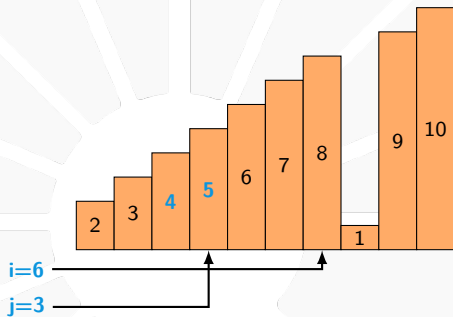


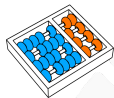
Ordenação por Inserção



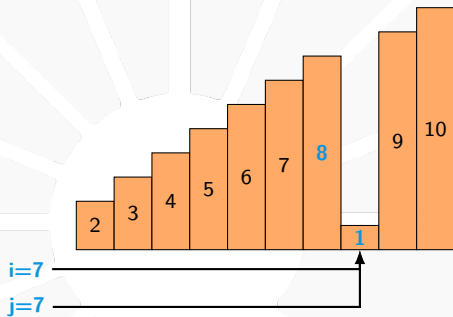


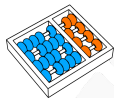
Ordenação por Inserção



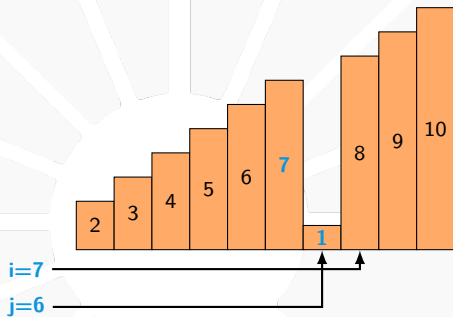


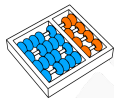
Ordenação por Inserção



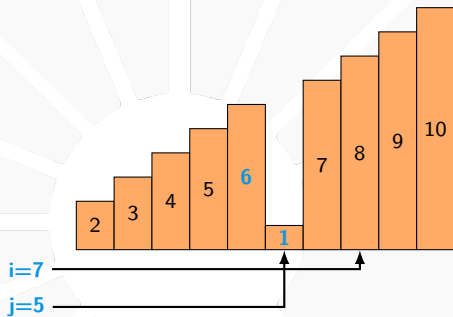


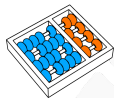
Ordenação por Inserção



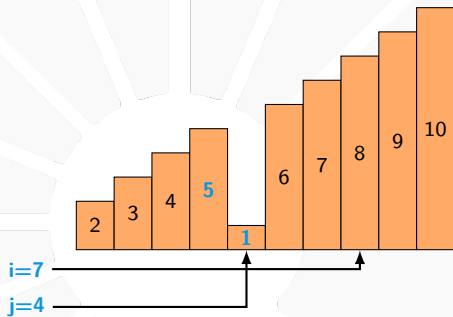


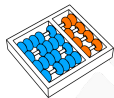
Ordenação por Inserção



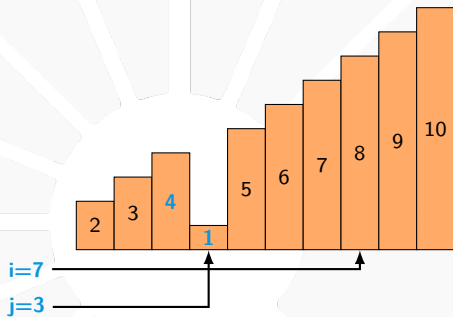


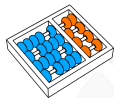
Ordenação por Inserção



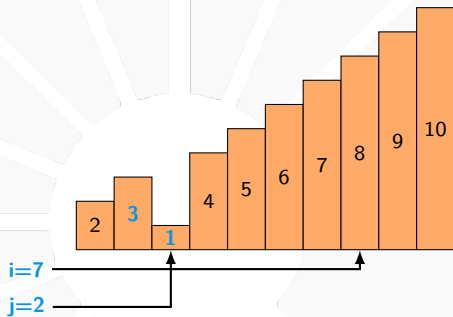


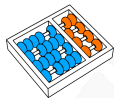
Ordenação por Inserção



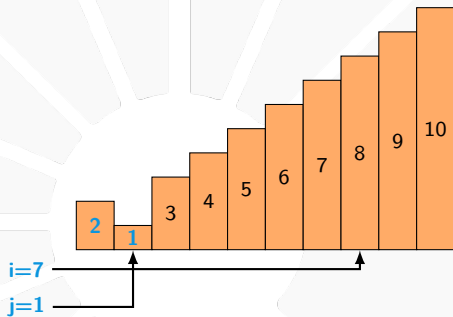


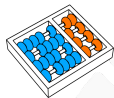
Ordenação por Inserção



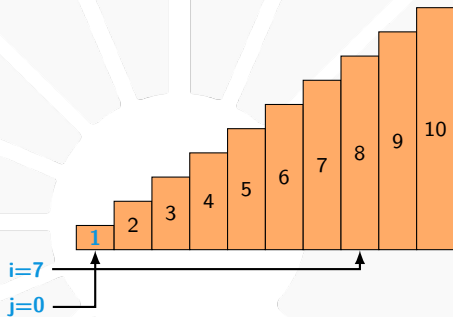


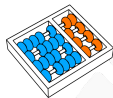
Ordenação por Inserção



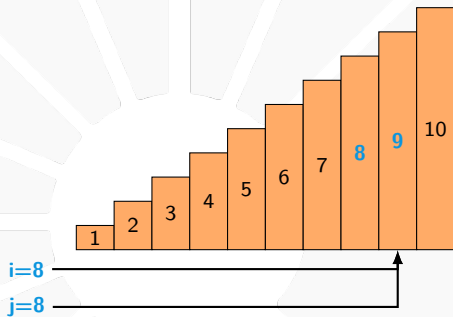


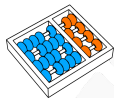
Ordenação por Inserção



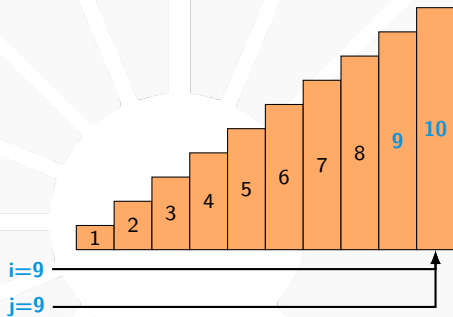


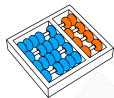
Ordenação por Inserção



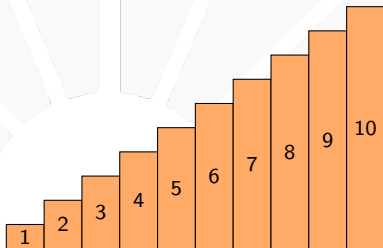


Ordenação por Inserção



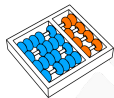


Ordenação por Inserção



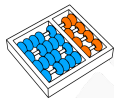
i

j



Ordenação por inserção

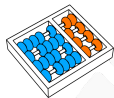
Para evitar muitas trocas:



Ordenação por inserção

Para evitar muitas trocas:

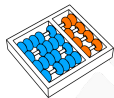
1. Percorrer cada posição i do vetor do início ao fim fazendo:



Ordenação por inserção

Para evitar muitas trocas:

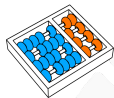
1. Percorrer cada posição **i** do vetor do início ao fim fazendo:
2. Armazenar o valor em **i** numa **chave**.



Ordenação por inserção

Para evitar muitas trocas:

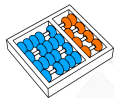
1. Percorrer cada posição i do vetor do início ao fim fazendo:
2. Armazenar o valor em i numa *chave*.
3. Percorrer de $i - 1$ até o início enquanto o valor for maior que *chave*:



Ordenação por inserção

Para evitar muitas trocas:

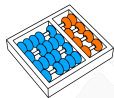
1. Percorrer cada posição i do vetor do início ao fim fazendo:
2. Armazenar o valor em i numa *chave*.
3. Percorrer de $i - 1$ até o início enquanto o valor for maior que *chave*:
4. Copiar o valor atual na posição à direita.



Ordenação por inserção

Para evitar muitas trocas:

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

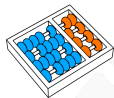


Algoritmo

Algoritmo: INSERTION-SORT(A, n)

```

1  para  $i \leftarrow 2$  até  $n$ 
2      chave  $\leftarrow A[i]$ 
3       $j \leftarrow i - 1$ 
4      enquanto  $j \geq 1$  e  $A[j] > \text{chave}$ 
5           $A[j + 1] \leftarrow A[j]$ 
6           $j \leftarrow j - 1$ 
7       $A[j + 1] \leftarrow \text{chave}$ 
    
```



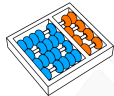
Algoritmo

Algoritmo: INSERTION-SORT(A, n)

```

1  para  $i \leftarrow 2$  até  $n$ 
2      chave  $\leftarrow A[i]$ 
3       $j \leftarrow i - 1$ 
4      enquanto  $j \geq 1$  e  $A[j] > \text{chave}$ 
5           $A[j + 1] \leftarrow A[j]$ 
6           $j \leftarrow j - 1$ 
7       $A[j + 1] \leftarrow \text{chave}$ 
    
```

Perguntas:



Algoritmo

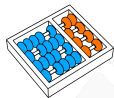
Algoritmo: INSERTION-SORT(A, n)

```

1  para  $i \leftarrow 2$  até  $n$ 
2      chave  $\leftarrow A[i]$ 
3       $j \leftarrow i - 1$ 
4      enquanto  $j \geq 1$  e  $A[j] > \text{chave}$ 
5           $A[j + 1] \leftarrow A[j]$ 
6           $j \leftarrow j - 1$ 
7       $A[j + 1] \leftarrow \text{chave}$ 
    
```

Perguntas:

- ▶ O algoritmo termina?



Algoritmo

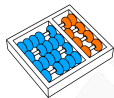
Algoritmo: INSERTION-SORT(A, n)

```

1  para  $i \leftarrow 2$  até  $n$ 
2      chave  $\leftarrow A[i]$ 
3       $j \leftarrow i - 1$ 
4      enquanto  $j \geq 1$  e  $A[j] > \text{chave}$ 
5           $A[j + 1] \leftarrow A[j]$ 
6           $j \leftarrow j - 1$ 
7       $A[j + 1] \leftarrow \text{chave}$ 
    
```

Perguntas:

- ▶ O algoritmo termina?
- ▶ Qual a sua complexidade?



Algoritmo

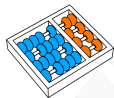
Algoritmo: INSERTION-SORT(A, n)

```

1  para  $i \leftarrow 2$  até  $n$ 
2      chave  $\leftarrow A[i]$ 
3       $j \leftarrow i - 1$ 
4      enquanto  $j \geq 1$  e  $A[j] > \text{chave}$ 
5           $A[j + 1] \leftarrow A[j]$ 
6           $j \leftarrow j - 1$ 
7       $A[j + 1] \leftarrow \text{chave}$ 
    
```

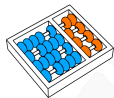
Perguntas:

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



Contando o número de instruções

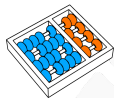
INSERTION-SORT(A, n)	Custo	Quantas vezes?
1 para $i \leftarrow 2$ até n		
2 chave $\leftarrow A[i]$		
3 $j \leftarrow i - 1$		
4 enquanto $j \geq 1$ e $A[j] > \text{chave}$		
5 $A[j + 1] \leftarrow A[j]$		
6 $j \leftarrow j - 1$		
7 $A[j + 1] \leftarrow \text{chave}$		



Contando o número de instruções

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

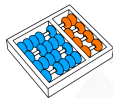
- ▶ A linha k executa um número constante de instruções c_k .



Contando o número de instruções

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

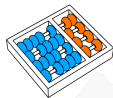
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.



Contando o número de instruções

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

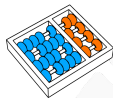
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.



Contando o número de instruções

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

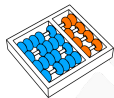
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.



Contando o número de instruções

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

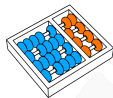
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.



Contando o número de instruções

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

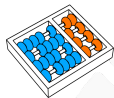
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.
- ▶ Quantas vezes a linha 4 executa depende da entrada.



Contando o número de instruções

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

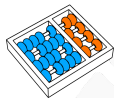
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.
- ▶ Quantas vezes a linha 4 executa depende da entrada.
 - ▶ t_i denota quantas vezes o **enquanto** executa para um certo i .



Contando o número de instruções

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

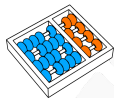
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.
- ▶ Quantas vezes a linha 4 executa depende da entrada.
 - ▶ t_i denota quantas vezes o **enquanto** executa para um certo i .



Contando o número de instruções

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

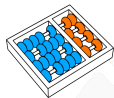
- ▶ A linha k executa um número constante de instruções c_k .
- ▶ Cada linha executa uma ou mais vezes.
- ▶ Quantas vezes a linha 4 executa depende da entrada.
 - ▶ t_i denota quantas vezes o **enquanto** executa para um certo i .



Contando o número de instruções

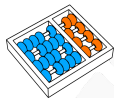
INSERTION-SORT(A, n)	Custo	Quantas vezes?
1 para $i \leftarrow 2$ até n	c_1	n
2 chave $\leftarrow A[i]$	c_2	$n - 1$
3 $j \leftarrow i - 1$	c_3	$n - 1$
4 enquanto $j \geq 1$ e $A[j] > \text{chave}$	c_4	$\sum_{i=2}^n t_i$
5 $A[j + 1] \leftarrow A[j]$	c_5	$\sum_{i=2}^n (t_i - 1)$
6 $j \leftarrow j - 1$	c_6	$\sum_{i=2}^n (t_i - 1)$
7 $A[j + 1] \leftarrow \text{chave}$	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_i denota quantas vezes o **enquanto** executa para um certo i .



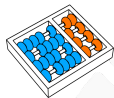
Tempo de execução total

- ▶ Considere uma instância de tamanho n .



Tempo de execução total

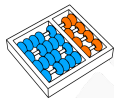
- ▶ Considere uma instância de tamanho n .
- ▶ $T(n)$ denota o número de instruções executadas para ela.



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:

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

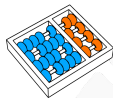


Tempo de execução total

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

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

Observações:



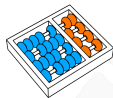
Tempo de execução total

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

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

Observações:

- ▶ Entradas do mesmo tamanho podem ter tempos diferentes.



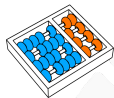
Tempo de execução total

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

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

Observações:

- ▶ Entradas do mesmo tamanho podem ter tempos diferentes.
- ▶ Vamos considerar diferentes instâncias.



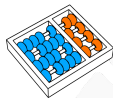
Tempo de execução total

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

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

Observações:

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



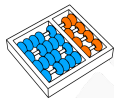
Tempo de execução total

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

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

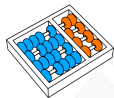
Observações:

- ▶ 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

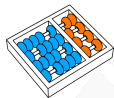
Um melhor caso ocorre quando $t_i = 1$ para cada i :



Melhor caso

Um melhor caso ocorre quando $t_i = 1$ para cada i :

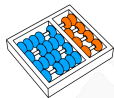
- ▶ Basta que a condição do **enquanto** sempre falhe.



Melhor caso

Um melhor caso ocorre quando $t_i = 1$ para cada i :

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

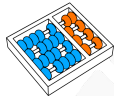


Melhor caso

Um melhor caso ocorre quando $t_i = 1$ para cada i :

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

Nesse caso:



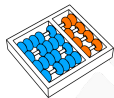
Melhor caso

Um melhor caso ocorre quando $t_i = 1$ para cada i :

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

Nesse caso:

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



Melhor caso

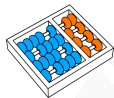
Um melhor caso ocorre quando $t_i = 1$ para cada i :

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

Nesse caso:

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

- ▶ Os valores de a e b são constantes.



Melhor caso

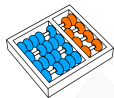
Um melhor caso ocorre quando $t_i = 1$ para cada i :

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

Nesse caso:

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

- ▶ Os valores de a e b são constantes.
- ▶ O tempo de execução no melhor caso é **linear** em n .



Melhor caso

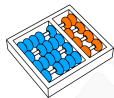
Um melhor caso ocorre quando $t_i = 1$ para cada i :

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

Nesse caso:

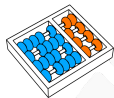
$$\begin{aligned} T(n) &= c_1 \cdot n + c_2 \cdot (n - 1) + c_3 \cdot (n - 1) + c_4 \cdot (n - 1) + c_7 \cdot (n - 1) \\ &= (c_1 + c_2 + c_3 + c_4 + c_7) \cdot n - (c_2 + c_3 + c_4 + c_7) \\ &= a \cdot n + b. \end{aligned}$$

- ▶ 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

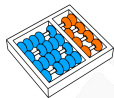
Um pior caso ocorre quando t_i é máximo para cada i :



Pior caso

Um pior caso ocorre quando t_i é máximo para cada i :

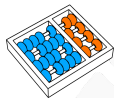
- ▶ Basta que a condição do **enquanto** só falhe quando $j = 0$.



Pior caso

Um pior caso ocorre quando t_i é máximo para cada i :

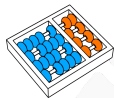
- ▶ Basta que a condição do **enquanto** só falha quando $j = 0$.
- ▶ Nessa situação, teremos $t_i = i$.



Pior caso

Um pior caso ocorre quando t_i é máximo para cada i :

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

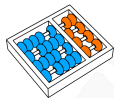


Pior caso

Um pior caso ocorre quando t_i é máximo para cada i :

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

Relembre que:



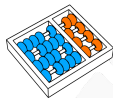
Pior caso

Um pior caso ocorre quando t_i é máximo para cada i :

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

Relembre que:

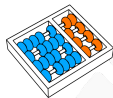
- ▶ $\sum_{i=2}^n i = n(n+1)/2 - 1$ e $\sum_{i=2}^n (i-1) = n(n-1)/2$.



Pior caso (cont)

Substituindo, temos:

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

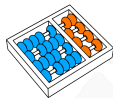


Pior caso (cont)

Substituindo, temos:

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

- Os valores de a , b , c são constantes.

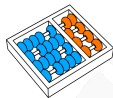


Pior caso (cont)

Substituindo, temos:

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

- ▶ Os valores de a , b , c são constantes.
- ▶ O tempo de execução no pior caso é **quadrático** em n .



Pior caso (cont)

Substituindo, temos:

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

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

ANÁLISE DE ALGORITMOS

MC458 - Projeto e Análise de
Algoritmos I

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

08/25

05



UNICAMP

