

# Complexidade de Tempo de Algoritmos Iterativos

Prof. André Vignatti

## Algoritmos e Notações $O$ e $\Omega$

**Definição** (Tempo de Execução de um Algoritmo).

- Um algoritmo é dito  $\Omega(f(n))$  se **todas** as instâncias executam  $\Omega(f(n))$ .
- Um algoritmo é dito  $O(g(n))$ , se **todas** as instâncias executam  $O(g(n))$ .

**Exemplo.**

- InsertionSort é  $O(n^2)$  pois **todos** os vetores passados como entrada fazem o algoritmo executar em tempo  $O(n^2)$ .
- InsertionSort **NÃO** é  $O(n)$  pois existem algumas instâncias (vetor ordenado ao inverso) que executam em tempo  $O(n^2)$ .
- InsertionSort é  $\Omega(n)$  pois **todos** os vetores passados como entrada fazem o algoritmo executar em tempo  $\Omega(n)$ .
- InsertionSort **NÃO** é  $\Omega(n^2)$  pois existem algumas instâncias (vetor ordenado) que executam em tempo  $\Omega(n)$ .

InsertionSort é  $O(n^3)$ ?

InsertionSort é  $\Omega(1)$ ?

Ambas as respostas são SIM, mas não diz muita coisa sobre o algoritmo...

- É melhor encontrar a **menor** função  $f(n)$  tal que o algoritmo é  $O(f(n))$ .
- É melhor encontrar a **maior** função  $g(n)$  tal que o algoritmo é  $\Omega(g(n))$ .

Como  $O$  limita por cima, pode ter sido feita uma análise **folgada**.

**Exemplo.** Supondo análise  $O(n^3)$  do InsertionSort (o que NÃO está errado). Podemos melhorar a análise?

- Não conseguimos encontrar vetor que gaste tempo  $\Omega(n^3)$ !

Como saber que a análise de  $O(f(n))$  não está folgada?

**Um jeito:** achar uma instância que execute em  $\Omega(f(n))$ .

- Neste caso, não haveria “espaço” para melhorias na análise.

**Exemplo.** Supondo análise  $O(n^2)$  e  $\Omega(n)$  do InsertionSort. Podemos melhorar?

- Vetor ordenado gasta  $O(n) \implies$  não dá pra aumentar  $\Omega(n)$
- Vetor ordenado ao inverso gasta  $\Omega(n^2) \implies$  não dá pra diminuir  $O(n^2)$ .

Ao comparar **melhor caso** X **pior caso** dificilmente temos uma análise justa (i.e., sem folgas).

**Solução:** fazer a análise justa do melhor caso separadamente do pior caso.

## Análise de algoritmos não justa

InsertionSort executa em tempo  $O(n^2)$  e  $\Omega(n)$ .

NÃO dá pra deixar a análise mais justa (apertada)!

Solução: separar por melhor e pior caso.

**Pior caso:** o pior caso do InsertionSort executa em  $O(n^2)$  e  $\Omega(n^2)$ .

- No pior caso, **todas** as instâncias executam em  $O(n^2)$ .
- No pior caso, **todas** as instâncias executam em tempo  $\Omega(n^2)$ .

**Melhor caso:** o melhor caso do InsertionSort executa em  $O(n)$  e  $\Omega(n)$ .

- No melhor caso, **todas** as instâncias executam em  $O(n)$ .
- No melhor caso, **todas** as instâncias executam em tempo  $\Omega(n)$ .

## Regras de Simplificação da Análise

atribuição	$\Theta(1)$
entrada da função	$\Theta(1)$
saída da função	$\Theta(1)$
desvio condicional (if)	tempo do teste mais $\Theta(\text{máximo dos dois branches})$
laço	soma de todas iterações do tempo de cada iteração

Juntar tudo isso usando a Regra da Soma e Regra do Produto.

Exceção - algoritmos recursivos.

## Multiplicação

```
1 Algoritmo MULTIPLICA( $y, z$ )
2    $x \leftarrow 0$ 
3   enquanto  $z > 0$  faça
4     se  $z$  é ímpar então  $x \leftarrow x + y$ 
5      $y \leftarrow 2y$ 
6      $z \leftarrow \lfloor z/2 \rfloor$ 
7   retorna  $x$ 
```

Suponha que  $y$  e  $z$  tenham  $n$  bits.

- entrada e saída da função :  $\Theta(1)$
- linha 2:  $\Theta(1)$
- linhas 4, 5 e 6:  $\Theta(1)$  cada vez que são executados
- laço custa  $O(n)$

Portanto, a multiplicação leva tempo  $O(n)$  (pela regra da soma e do produto)

## Bubblesort

```
1 Algoritmo BUBBLESORT( $A[1..n]$ )
2   para  $i \leftarrow 1$  até  $n - 1$  faça
3     para  $j \leftarrow 1$  até  $n - i$  faça
4       se  $A[j] > A[j + 1]$  então
5         Troca  $A[j]$  com  $A[j + 1]$ 
```

- entrada e saída da função :  $\Theta(1)$
- linha 5:  $O(1)$
- if:  $O(1)$
- laço interno:  $O(n)$
- laço externo:  $O(n)$

$$O(n)O(n) = O(n^2).$$

Análise foi folgada?

Pode-se provar que é  $\Omega(n^2)$ , concluindo então  $\Theta(n^2)$ .

## Truque de Análise

Ao invés de usar o método passo-a-passo de análise:

- Identificar a **operação fundamental** usada no algoritmo, e observar que o tempo de execução é uma constante múltipla do número de operações fundamentais executadas
  - Vantagem: não precisa fazer uma análise linha-a-linha
- Analisar o **número exato** dessa operação
  - Vantagem: trabalhar com números ao invés de notação assintótica
  - Vantagem: menos chance de fazer análise folgada

## Exemplo

No BUBBLESORT, a operação fundamental é a comparação (“se”).

Assim, tempo de execução será  $O$  do número de comparações.

- A linha 4 usa 1 comparação
- O laço interno usa  $n - i$  comparações
- O laço externo usa  $\sum_{i=1}^{n-1} (n - i)$  comparações

Então

$$\begin{aligned} \sum_{i=1}^{n-1} (n - i) &= n(n - 1) - \sum_{i=1}^{n-1} i \\ &= n(n - 1) - n(n - 1)/2 \\ &= n(n - 1)/2. \end{aligned}$$

## Observações e Pegadinhas

O algoritmo de multiplicação leva tempo  $O(n)$ .

O que isso significa? Tenha cuidado com

- Suposições escondidas:  $n$  é o número de bits. Modelo RAM: +, -, \* e / custam  $\Theta(1)$ .
- O algoritmo de multiplicação leva tempo  $O(n^2)$  não é uma afirmação falsa!
- As constantes multiplicativas escondidas: podem fazer o algoritmo tornar-se ruim na prática.