

## MC458 — Projeto e Análise de Algoritmos I

Cid Carvalho de Souza   Cândida Nunes da Silva  
Orlando Lee

3 de março de 2016

### Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram **direta ou indiretamente** com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Eis a lista de “colaboradores” (**em ordem alfabética**):
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

### Antes de mais nada. . .

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- Esses slides são o fruto de um trabalho colaborativo de vários professores.
- Nunca é demais enfatizar que o material é apenas um **guia** e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial, “Cormen” e “Manber”).

Orlando Lee

### Complexidade assintótica de algoritmos

- Como dito anteriormente, na maior parte desta disciplina, estaremos nos concentrando na **análise de pior caso** e no **comportamento assintótico** dos algoritmos (instâncias de **tamanho grande**).
- Considere o algoritmo **Ordena-Por-Inserção (Insertionsort)**. Veremos que ele tem complexidade (de **pior caso**) igual a uma função quadrática  $an^2 + bn + c$ , onde  $a, b, c$  são constantes absolutas.
- O estudo assintótico nos permite “jogar para debaixo do tapete” os valores destas constantes, i.e., aquilo que independe do tamanho da entrada (neste caso os valores de  $a, b$  e  $c$ ).
- **Por que podemos fazer isso?**

## Análise assintótica de funções quadráticas

Considere a função quadrática  $3n^2 + 10n + 50$ :

$n$	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

Como se vê,  $3n^2$  é o termo dominante quando  $n$  é grande.

De um modo geral, podemos nos concentrar nos termos dominantes e esquecer os demais.

## Notação assintótica

- Usando notação assintótica, dizemos que o algoritmo Ordena-Por-Inserção tem complexidade de tempo de pior caso  $\Theta(n^2)$ .
- Isto quer dizer duas coisas:
  - a complexidade de tempo é limitada (superiormente) assintoticamente por algum polinômio da forma  $an^2$ , e
  - para todo  $n$  suficientemente grande, existe alguma instância de tamanho  $n$  que consome tempo pelo menos  $dn^2$ , para alguma constante positiva  $d$ .
- Mais adiante discutiremos em detalhes o uso da notação assintótica em análise de algoritmos.

## Algoritmos recursivos

"To understand recursion, we must first understand recursion."

(anônimo)

- O que é o paradigma de **divisão-e-conquista**?
- Como mostrar a corretude de um algoritmo recursivo?
- Como analisar o consumo de tempo de um algoritmo recursivo?
- O que é uma **fórmula de recorrência**?
- O que significa **resolver** uma fórmula de recorrência?

## Recursão e o paradigma de divisão-e-conquista

- Um **algoritmo recursivo** encontra a saída para uma instância de entrada de um problema chamando a si mesmo para resolver instâncias menores deste mesmo problema.
- Algoritmos de **divisão-e-conquista** possuem três etapas em cada nível de recursão:
  - 1 **Divisão**: o problema é dividido em subproblemas semelhantes ao problema original, porém tendo com entrada instâncias de tamanho menor;
  - 2 **Conquista**: cada subproblema é resolvido recursivamente a menos que o tamanho de sua entrada seja suficientemente "pequeno", situação na qual ele é resolvido diretamente;
  - 3 **Combinação**: as soluções dos subproblemas são combinadas para obter uma solução do problema original.

## Exemplo de divisão-e-conquista: Mergesort

- Mergesort é um algoritmo para resolver o problema de ordenação e um exemplo clássico do uso do paradigma de **divisão-e-conquista**. (*to merge = intercalar*)
- Descrição do Mergesort em alto nível;
  - Divisão**: divida o vetor com  $n$  elementos em dois subvetores de tamanho  $\lfloor n/2 \rfloor$  e  $\lceil n/2 \rceil$ , respectivamente.
  - Conquista**: ordene os dois vetores **recursivamente** usando o Mergesort;
  - Combinação**: intercale os dois subvetores para obter um vetor ordenado usando o algoritmo Intercala.

## Mergesort

**Relembrando**: o objetivo é reorganizar  $A[p \dots r]$ , com  $p \leq r$ , em ordem crescente.

```
MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
3          MERGESORT( $A, p, q$ )
4          MERGESORT( $A, q + 1, r$ )
5          INTERCALA( $A, p, q, r$ )
```

	$p$			$q$				$r$	
A	66	33	55	44	99	11	77	22	88

## Mergesort

**Relembrando**: o objetivo é reorganizar  $A[p \dots r]$ , com  $p \leq r$ , em ordem crescente.

```
MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
3          MERGESORT( $A, p, q$ )
4          MERGESORT( $A, q + 1, r$ )
5          INTERCALA( $A, p, q, r$ )
```

	$p$			$q$				$r$	
A	33	44	55	66	99	11	77	22	88

## Mergesort

**Relembrando**: o objetivo é reorganizar  $A[p \dots r]$ , com  $p \leq r$ , em ordem crescente.

```
MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
3          MERGESORT( $A, p, q$ )
4          MERGESORT( $A, q + 1, r$ )
5          INTERCALA( $A, p, q, r$ )
```

	$p$			$q$				$r$	
A	33	44	55	66	99	11	22	77	88

## Mergesort

**Relembrando:** o objetivo é reorganizar  $A[p \dots r]$ , com  $p \leq r$ , em ordem crescente.

```

MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3        MERGESORT( $A, p, q$ )
4        MERGESORT( $A, q+1, r$ )
5        INTERCALA( $A, p, q, r$ )
    
```

	$p$			$q$				$r$	
A	11	22	33	44	55	66	77	88	99

## Corretude do Mergesort

```

MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3        MERGESORT( $A, p, q$ )
4        MERGESORT( $A, q+1, r$ )
5        INTERCALA( $A, p, q, r$ )
    
```

O algoritmo está correto?

A corretude do algoritmo **Mergesort** apoia-se na corretude do algoritmo **Intercala** e pode ser demonstrada **por indução** em  $n := r - p + 1$ .

Aprenderemos como fazer provas por indução mais adiante.

## Complexidade do Mergesort

```

MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3        MERGESORT( $A, p, q$ )
4        MERGESORT( $A, q+1, r$ )
5        INTERCALA( $A, p, q, r$ )
    
```

linha	consumo de tempo
1	?
2	?
3	?
4	?
5	?

$T(n) = ?$

## Complexidade do Mergesort

```

MERGESORT( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3        MERGESORT( $A, p, q$ )
4        MERGESORT( $A, q+1, r$ )
5        INTERCALA( $A, p, q, r$ )
    
```

linha	consumo de tempo
1	$\Theta(1)$
2	$\Theta(1)$
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	$\Theta(n)$

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) + \Theta(2)$

## Complexidade do Mergesort

- Obtemos o que chamamos de **fórmula de recorrência** (i.e., uma fórmula que define uma função em termos dela mesma).

$$T(1) = \Theta(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \text{ para } n = 2, 3, 4, \dots$$

- Em geral, ao aplicar o paradigma de **divisão-e-conquista**, chega-se a um algoritmo recursivo cuja complexidade  $T(n)$  é dada por uma **fórmula de recorrência**.
- É preciso então **resolver** a recorrência! Mas, o que significa resolver uma recorrência?
- Significa encontrar uma “fórmula fechada” para  $T(n)$ .
- No caso,  $T(n) = \Theta(n \lg n)$ . Ou seja, o consumo de tempo do **Mergesort** é  $\Theta(n \lg n)$  no pior caso.
- Veremos depois como resolver recorrências.

## Crescimento de funções

## Notação Assintótica

- Vamos expressar complexidade através de funções em variáveis que descrevam o tamanho de instâncias do problema. Exemplos:
  - Problemas de aritmética: número de bits (ou bytes) dos inteiros.
  - Problemas em grafos: número de vértices e/ou arestas
  - Problemas de ordenação de vetores: tamanho do vetor.
  - Busca em textos: número de caracteres do texto ou padrão de busca.
- Vamos supor que funções que expressam complexidade são sempre positivas, já que estamos contando o número de operações.

## Comparação de Funções

- Vamos comparar funções assintoticamente, ou seja, para valores grandes, desprezando constantes multiplicativas e termos de menor ordem.

Por que queremos fazer isto? Considere as funções

$$n^2, 2n^2, 30n^2, 100n^2, cn^2.$$

Obviamente, quanto maior a constante  $c$  associada, maior é o valor da função. Entretanto, todas elas têm a mesma **velocidade de crescimento**. Podemos **ignorar** o valor de  $c$ .

Por exemplo, considere as funções  $n^3$  e  $cn^2$ .

Para  $n \leq c$ , temos que  $n^3 \leq cn^2$ , mas a partir de  $n > c$  a função  $n^3$  cresce mais rapidamente que a função  $cn^2$ .

## Comparação de Funções

- Vamos comparar funções assintoticamente, ou seja, para valores grandes, desprezando constantes multiplicativas e termos de menor ordem.

	$n = 100$	$n = 1000$	$n = 10^4$	$n = 10^6$	$n = 10^9$
$\log n$	2	3	4	6	9
$n$	100	1000	$10^4$	$10^6$	$10^9$
$n \log n$	200	3000	$4 \cdot 10^4$	$6 \cdot 10^6$	$9 \cdot 10^9$
$n^2$	$10^4$	$10^6$	$10^8$	$10^{12}$	$10^{18}$
$100n^2 + 15n$	$1,0015 \cdot 10^6$	$1,00015 \cdot 10^8$	$\approx 10^{10}$	$\approx 10^{14}$	$\approx 10^{20}$
$2^n$	$\approx 1,26 \cdot 10^{30}$	$\approx 1,07 \cdot 10^{301}$	?	?	?

## Classe O

### Definição:

$O(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in O(g(n))$ , então  $f(n)$  cresce no máximo tão rapidamente quanto  $g(n)$ .

### Exemplo:

$$\frac{1}{2}n^2 - 3n \in O(n^2).$$

Valores de  $c$  e  $n_0$  que satisfazem a definição são

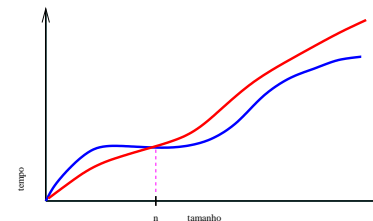
$$c = \frac{1}{2} \text{ e } n_0 = 7.$$

## Classe O

### Definição:

$O(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in O(g(n))$ , então  $f(n)$  cresce no máximo tão rapidamente quanto  $g(n)$ .



## Classe O

### Definição:

$O(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in O(g(n))$ , então  $f(n)$  cresce no máximo tão rapidamente quanto  $g(n)$ .

### Exemplo:

O que é a classe  $O(1)$ ?

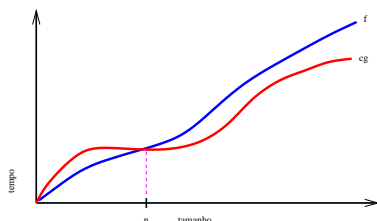
É classe de funções que são limitadas por alguma constante.

## Classe $\Omega$

### Definição:

$\Omega(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in \Omega(g(n))$ , então  $f(n)$  cresce no mínimo tão lentamente quanto  $g(n)$ .



## Classe $\Omega$

### Definição:

$\Omega(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in \Omega(g(n))$ , então  $f(n)$  cresce no mínimo tão lentamente quanto  $g(n)$ .

### Exemplo:

$\frac{1}{2}n^2 - 3n \in \Omega(n^2).$

Valores de  $c$  e  $n_0$  que satisfazem a definição são

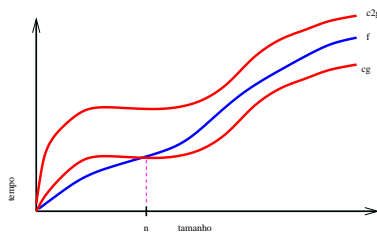
$$c = \frac{1}{14} \text{ e } n_0 = 7.$$

## Classe $\Theta$

### Definição:

$\Theta(g(n)) = \{f(n) : \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in \Theta(g(n))$ , então  $f(n)$  cresce tão rapidamente quanto  $g(n)$ .



## Classe $\Theta$

### Definição:

$\Theta(g(n)) = \{f(n) : \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in \Theta(g(n))$ , então  $f(n)$  cresce tão rapidamente quanto  $g(n)$ .

### Exemplo:

$\frac{1}{2}n^2 - 3n \in \Theta(n^2).$

Valores de  $c_1$ ,  $c_2$  e  $n_0$  que satisfazem a definição são

$$c_1 = \frac{1}{14}, c_2 = \frac{1}{2} \text{ e } n_0 = 7.$$

## Classe $o$

### Definição:

$o(g(n)) = \{f(n) : \text{para toda constante positiva } c, \text{ existe uma constante } n_0 > 0 \text{ tal que } 0 \leq f(n) < cg(n), \text{ para todo } n \geq n_0\}.$

Informalmente, dizemos que, se  $f(n) \in o(g(n))$ , então  $f(n)$  cresce mais lentamente que  $g(n)$ .

### Exemplo:

$$1000n^2 \in o(n^3)$$

Para cada valor de  $c$ , um  $n_0$  que satisfaz a definição é

$$n_0 = \left\lceil \frac{1000}{c} \right\rceil + 1.$$

## Classe $\omega$

### Definição:

$\omega(g(n)) = \{f(n) : \text{para toda constante positiva } c, \text{ existe uma constante } n_0 > 0 \text{ tal que } 0 \leq cg(n) < f(n), \text{ para todo } n \geq n_0.\}$

Informalmente, dizemos que, se  $f(n) \in \omega(g(n))$ , então  $f(n)$  cresce mais rapidamente que  $g(n)$ .

### Exemplo:

$$\frac{1}{1000}n^2 \in \omega(n)$$

Para cada valor de  $c$ , um  $n_0$  que satisfaz a definição é

$$n_0 = \lceil 1000c \rceil + 1.$$

## Definições equivalentes

$$\begin{aligned} f(n) \in o(g(n)) &\text{ se } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0. \\ f(n) \in O(g(n)) &\text{ se } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty. \\ f(n) \in \Theta(g(n)) &\text{ se } 0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty. \\ f(n) \in \Omega(g(n)) &\text{ se } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0. \\ f(n) \in \omega(g(n)) &\text{ se } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty. \end{aligned}$$

## Propriedades das Classes

### Transitividade:

Se  $f(n) \in O(g(n))$  e  $g(n) \in O(h(n))$ , então  $f(n) \in O(h(n))$ .  
Se  $f(n) \in \Omega(g(n))$  e  $g(n) \in \Omega(h(n))$ , então  $f(n) \in \Omega(h(n))$ .  
Se  $f(n) \in \Theta(g(n))$  e  $g(n) \in \Theta(h(n))$ , então  $f(n) \in \Theta(h(n))$ .  
Se  $f(n) \in o(g(n))$  e  $g(n) \in o(h(n))$ , então  $f(n) \in o(h(n))$ .  
Se  $f(n) \in \omega(g(n))$  e  $g(n) \in \omega(h(n))$ , então  $f(n) \in \omega(h(n))$ .



## Propriedades das Classes

### Reflexividade:

$$f(n) \in O(f(n)).$$

$$f(n) \in \Omega(f(n)).$$

$$f(n) \in \Theta(f(n)).$$

### Simetria:

$$f(n) \in \Theta(g(n)) \text{ se, e somente se, } g(n) \in \Theta(f(n)).$$

### Simetria Transposta:

$$f(n) \in O(g(n)) \text{ se, e somente se, } g(n) \in \Omega(f(n)).$$

$$f(n) \in o(g(n)) \text{ se, e somente se, } g(n) \in \omega(f(n)).$$

## Convenção

Exemplo 1: usaremos expressões como  $f(n) = O(g(n))$ . Ela significa a mesma coisa que  $f(n) \in O(g(n))$ , ou seja, a função  $f(n)$  pertence à classe de funções  $O(g(n))$ . O mesmo se aplica para  $\Omega$  e  $\Theta$ .

Exemplo 2: Mergesort tem complexidade de tempo  $O(n \lg n)$ . Isto significa que a função  $T(n)$  que mede a complexidade de tempo (de pior caso) do Mergesort pertence à classe de funções  $O(n \lg n)$ .

Exemplo 3: Mergesort tem complexidade de tempo  $\Theta(n \lg n)$ . Isto significa que a complexidade de tempo de Mergesort é  $O(n \lg n)$  e existe alguma instância que requer tempo  $\Omega(n \lg n)$ .

Por outro lado, Mergesort tem complexidade de tempo  $O(n^2)$  mas não  $\Omega(n^2)$ .

## Exemplos

Faça a comparação assintótica das seguintes funções:

- $n \log n$
- $2^\pi$
- $2^n$
- $n$
- $n^2$
- $\log n$
- $100n^2 + 15n$