

# Tópicos de Análise de Algoritmos

Parte destes slides são adaptações de slides  
do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

# Introdução

CLRS 2.2 e 3.1  
AU 3.3, 3.4 e 3.6

## Exemplo: número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Uma **inversão** é um par  $(i, j)$  de índices de  $p$  tal que  $i < j$  e  $p[i] > p[j]$ .

Entrada:

	1	2	3	4	5	6	7	8	9
$p$	2	4	1	9	5	3	8	6	7

## Exemplo: número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Uma **inversão** é um par  $(i, j)$  de índices de  $p$  tal que  $i < j$  e  $p[i] > p[j]$ .

Entrada:

	1	2	3	4	5	6	7	8	9
$p$	2	4	1	9	5	3	8	6	7

Saída: 11

Inversões:  $(1, 3)$ ,  $(2, 3)$ ,  $(4, 5)$ ,  $(2, 6)$ ,  $(4, 6)$ ,  
 $(5, 6)$ ,  $(4, 7)$ ,  $(4, 8)$ ,  $(7, 8)$ ,  $(4, 9)$  e  $(7, 9)$ .

# Número de inversões

## CONTA-INVERSÕES ( $p, n$ )

```
1   $c \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3      para  $j \leftarrow i + 1$  até  $n$  faça
4          se  $p[i] > p[j]$ 
5              então  $c \leftarrow c + 1$ 
6  devolva  $c$ 
```

# Número de inversões

CONTA-INVERSÕES ( $p, n$ )

```
1   $c \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3      para  $j \leftarrow i + 1$  até  $n$  faça
4          se  $p[i] > p[j]$ 
5              então  $c \leftarrow c + 1$ 
6  devolva  $c$ 
```

Se a execução de cada linha de código consome **1 unidade** de tempo, o consumo total é ...

## Consumo de tempo

Se a execução de cada linha de código consome **1 unidade** de tempo, o consumo total é:

linha	todas as execuções da linha
1	= 1
2	= $n$
3	= $\sum_{i=2}^n i = (n+2)(n-1)/2$
4	= $\sum_{i=1}^{n-1} i = n(n-1)/2$
5	$\leq \sum_{i=1}^{n-1} i = n(n-1)/2$
6	= 1
<b>total</b>	$\leq (3/2)n^2 + n/2 + 1$

## Consumo de tempo

Se a execução de cada linha de código consome **1 unidade** de tempo, o consumo total é:

linha	todas as execuções da linha
1	= 1
2	= $n$
3	= $\sum_{i=2}^n i = (n+2)(n-1)/2$
4	= $\sum_{i=1}^{n-1} i = n(n-1)/2$
5	$\leq \sum_{i=1}^{n-1} i = n(n-1)/2$
6	= 1
<b>total</b>	$\leq (3/2)n^2 + n/2 + 1$

O algoritmo **CONTA-INVERSÕES** consome não mais que  $(3/2)n^2 + n/2 + 1$  unidades de tempo.

## Consumo de tempo

Se a execução de **cada linha de código consome um tempo diferente**, o consumo total é:

linha	todas as execuções da linha	
1	= 1	$\times t_1$
2	= $n$	$\times t_2$
3	= $(n + 2)(n - 1)/2$	$\times t_3$
4	= $n(n - 1)/2$	$\times t_4$
5	$\leq n(n - 1)/2$	$\times t_5$
6	= 1	$\times t_6$
<b>total</b>	$\leq$	<b>?</b>

## Consumo de tempo

Se a execução de **cada linha de código consome um tempo diferente**, o consumo total é:

linha	todas as execuções da linha		
1	=	1	$\times t_1$
2	=	$n$	$\times t_2$
3	=	$(n + 2)(n - 1)/2$	$\times t_3$
4	=	$n(n - 1)/2$	$\times t_4$
5	$\leq$	$n(n - 1)/2$	$\times t_5$
6	=	1	$\times t_6$
<b>total</b>	$\leq$	$(\frac{t_3+t_4+t_5}{2})n^2 + (t_2 + \frac{t_3-t_4-t_5}{2})n + (t_1 - t_3 + t_6)$	
	=	$c_2n^2 + c_1n + c_0,$	

onde  $c_2$ ,  $c_1$  e  $c_0$  são constantes que dependem da máquina.

## Consumo de tempo

Se a execução de **cada linha de código consome um tempo diferente**, o consumo total é:

linha	todas as execuções da linha		
1	=	1	$\times t_1$
2	=	$n$	$\times t_2$
3	=	$(n + 2)(n - 1)/2$	$\times t_3$
4	=	$n(n - 1)/2$	$\times t_4$
5	$\leq$	$n(n - 1)/2$	$\times t_5$
6	=	1	$\times t_6$
<b>total</b>	$\leq$	$(\frac{t_3+t_4+t_5}{2})n^2 + (t_2 + \frac{t_3-t_4-t_5}{2})n + (t_1 - t_3 + t_6)$	
	=	$c_2n^2 + c_1n + c_0,$	

onde  $c_2$ ,  $c_1$  e  $c_0$  são constantes que dependem da máquina.

$n^2$  é para sempre! Está nas entranhas do algoritmo!

# Notação O

Intuitivamente...

$O(f(n)) \approx$  funções que não crescem mais rápido que  $f(n)$   
 $\approx$  funções menores ou iguais a um múltiplo de  $f(n)$

$n^2$        $(3/2)n^2$        $9999n^2$        $n^2/1000$       etc.

crescem todas com a **mesma velocidade**

# Notação O

Intuitivamente...

$O(f(n)) \approx$  funções que não crescem mais rápido que  $f(n)$   
 $\approx$  funções menores ou iguais a um múltiplo de  $f(n)$

$n^2$        $(3/2)n^2$        $9999n^2$        $n^2/1000$       etc.

crescem todas com a **mesma velocidade**

- ▶  $n^2 + 99n$  é  $O(n^2)$
- ▶  $33n^2$  é  $O(n^2)$
- ▶  $9n + 2$  é  $O(n^2)$
- ▶  $0,00001n^3 - 200n^2$  **não é**  $O(n^2)$

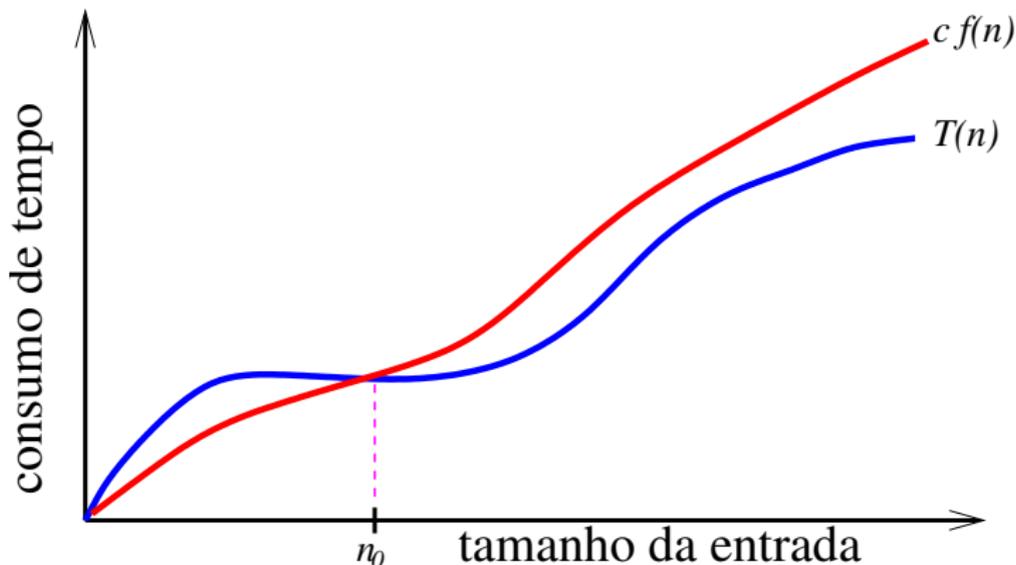
## Definição

Sejam  $T(n)$  e  $f(n)$  funções dos inteiros nos reais.

Dizemos que  $T(n)$  é  $O(f(n))$  se existem constantes positivas  $c$  e  $n_0$  tais que

$$0 \leq T(n) \leq c f(n)$$

para todo  $n \geq n_0$ .

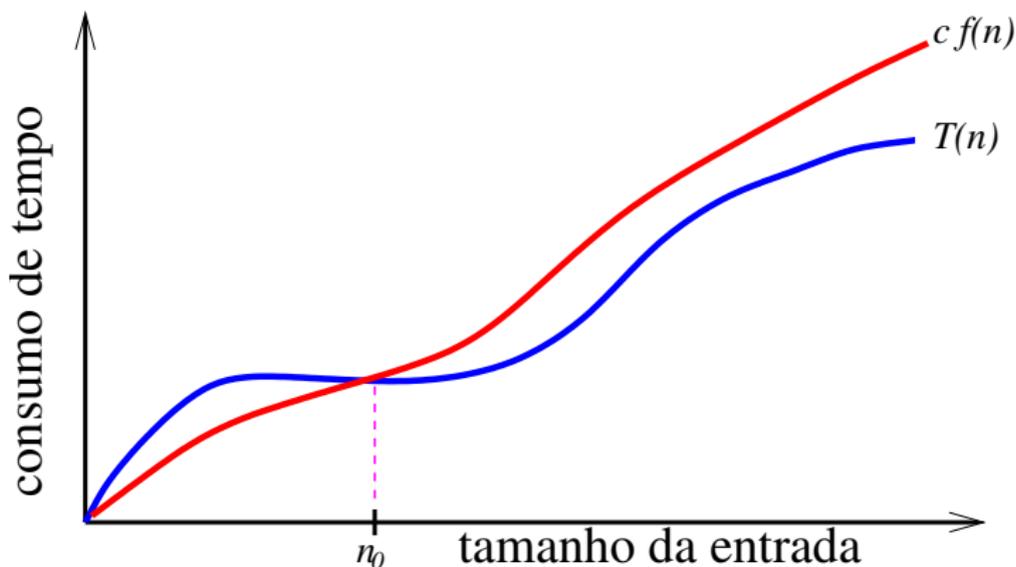


## Mais informal

$T(n)$  é  $O(f(n))$  se existe  $c > 0$  tal que

$$T(n) \leq c f(n)$$

para todo  $n$  suficientemente GRANDE.



## Exemplos

$T(n)$  é  $O(f(n))$  lê-se “ $T(n)$  é O de  $f(n)$ ” ou  
“ $T(n)$  é da ordem de  $f(n)$ ”

## Exemplos

$T(n)$  é  $O(f(n))$  lê-se “ $T(n)$  é O de  $f(n)$ ” ou  
“ $T(n)$  é da ordem de  $f(n)$ ”

### Exemplo 1

$10n^2$  é  $O(n^3)$ .

## Exemplos

$T(n)$  é  $O(f(n))$  lê-se “ $T(n)$  é O de  $f(n)$ ” ou  
“ $T(n)$  é da ordem de  $f(n)$ ”

### Exemplo 1

$10n^2$  é  $O(n^3)$ .

**Prova:** Para  $n \geq 1$ , temos que  $0 \leq 10n^2 \leq 10n^3$ .

## Exemplos

$T(n)$  é  $O(f(n))$  lê-se “ $T(n)$  é O de  $f(n)$ ” ou  
“ $T(n)$  é da ordem de  $f(n)$ ”

### Exemplo 1

$10n^2$  é  $O(n^3)$ .

**Prova:** Para  $n \geq 1$ , temos que  $0 \leq 10n^2 \leq 10n^3$ .

**Outra prova:** Para  $n \geq 10$ , temos  $0 \leq 10n^2 \leq n \times n^2 = 1n^3$ .

## Exemplos

$T(n)$  é  $O(f(n))$  lê-se “ $T(n)$  é O de  $f(n)$ ” ou  
“ $T(n)$  é da ordem de  $f(n)$ ”

### Exemplo 1

$10n^2$  é  $O(n^3)$ .

**Prova:** Para  $n \geq 1$ , temos que  $0 \leq 10n^2 \leq 10n^3$ .

**Outra prova:** Para  $n \geq 10$ , temos  $0 \leq 10n^2 \leq n \times n^2 = 1n^3$ .

### Exemplo 2

$\lg n$  é  $O(n)$ .

## Exemplos

$T(n)$  é  $O(f(n))$  lê-se “ $T(n)$  é O de  $f(n)$ ” ou  
“ $T(n)$  é da ordem de  $f(n)$ ”

### Exemplo 1

$10n^2$  é  $O(n^3)$ .

**Prova:** Para  $n \geq 1$ , temos que  $0 \leq 10n^2 \leq 10n^3$ .

**Outra prova:** Para  $n \geq 10$ , temos  $0 \leq 10n^2 \leq n \times n^2 = 1n^3$ .

### Exemplo 2

$\lg n$  é  $O(n)$ .

**Prova:** Para  $n \geq 1$ , tem-se que  $\lg n \leq 1n$ .

## Mais exemplos

### Exemplo 3

$20n^3 + 10n \log n + 5$  é  $O(n^3)$ .

## Mais exemplos

### Exemplo 3

$20n^3 + 10n \log n + 5$  é  $O(n^3)$ .

Prova: Para  $n \geq 1$ , tem-se que

$$20n^3 + 10n \lg n + 5 \leq 20n^3 + 10n^3 + 5n^3 = 35n^3.$$

## Mais exemplos

### Exemplo 3

$20n^3 + 10n \log n + 5$  é  $O(n^3)$ .

Prova: Para  $n \geq 1$ , tem-se que

$$20n^3 + 10n \lg n + 5 \leq 20n^3 + 10n^3 + 5n^3 = 35n^3.$$

Outra prova: Para  $n \geq 10$ , tem-se que

$$20n^3 + 10n \lg n + 5 \leq 20n^3 + n n \lg n + n \leq 20n^3 + n^3 + n^3 = 22n^3.$$

## Uso da notação $O$

$O(f(n)) = \{T(n) : \text{existem } c \text{ e } n_0 \text{ tq } T(n) \leq cf(n), n \geq n_0\}$

“ $T(n)$  é  $O(f(n))$ ” deve ser entendido como “ $T(n) \in O(f(n))$ ”.

“ $T(n) = O(f(n))$ ” deve ser entendido como “ $T(n) \in O(f(n))$ ”.

“ $T(n) \leq O(f(n))$ ” é feio.

“ $T(n) \geq O(f(n))$ ” não faz sentido!

“ $T(n)$  é  $g(n) + O(f(n))$ ” significa que existem constantes positivas  $c$  e  $n_0$  tais que

$$T(n) \leq g(n) + cf(n)$$

para todo  $n \geq n_0$ .

## Nomes de classes $O$

classe	nome
$O(1)$	constante
$O(\lg n)$	logarítmica
$O(n)$	linear
$O(n \lg n)$	$n \log n$
$O(n^2)$	quadrática
$O(n^3)$	cúbica
$O(n^k)$ com $k \geq 1$	polinomial
$O(2^n)$	exponencial
$O(a^n)$ com $a > 1$	exponencial

# Número de inversões

CONTA-INVERSÕES ( $p, n$ )

```
1   $c \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3      para  $j \leftarrow i + 1$  até  $n$  faça
4          se  $p[i] > p[j]$ 
5              então  $c \leftarrow c + 1$ 
6  devolva  $c$ 
```

# Número de inversões

CONTA-INVERSÕES ( $p, n$ )

```
1   $c \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3      para  $j \leftarrow i + 1$  até  $n$  faça
4          se  $p[i] > p[j]$ 
5              então  $c \leftarrow c + 1$ 
6  devolva  $c$ 
```

linha    consumo de todas as execuções da linha

---

1	?
2	?
3	?
4	?
5	?
6	?

---

total    ?

# Número de inversões

CONTA-INVERSÕES ( $p, n$ )

```
1   $c \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3      para  $j \leftarrow i + 1$  até  $n$  faça
4          se  $p[i] > p[j]$ 
5              então  $c \leftarrow c + 1$ 
6  devolva  $c$ 
```

linha    consumo de todas as execuções da linha

---

1         $O(1)$

2         $O(n)$

3         $O(n^2)$

4         $O(n^2)$

5         $O(n^2)$

6         $O(1)$

---

**total**     $O(3n^2 + n + 2) = O(n^2)$

## Conclusão

O algoritmo **CONTA-INVERSÕES** consome  $O(n^2)$  unidades de tempo.

Também escreve-se

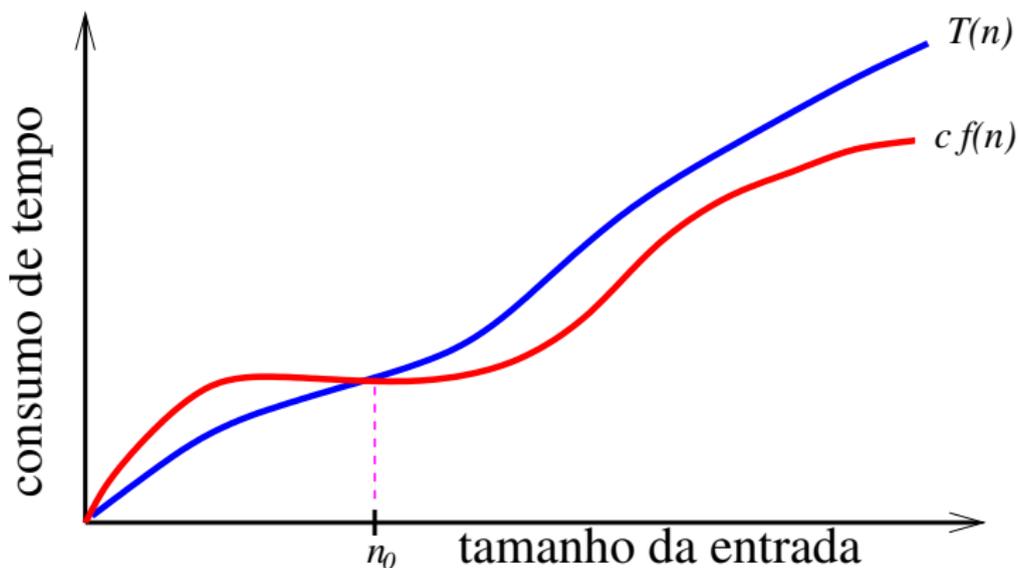
O algoritmo **CONTA-INVERSÕES** consome tempo  $O(n^2)$ .

## Notação Omega

Dizemos que  $T(n)$  é  $\Omega(f(n))$  se existem constantes positivas  $c$  e  $n_0$  tais que

$$c f(n) \leq T(n)$$

para todo  $n \geq n_0$ .

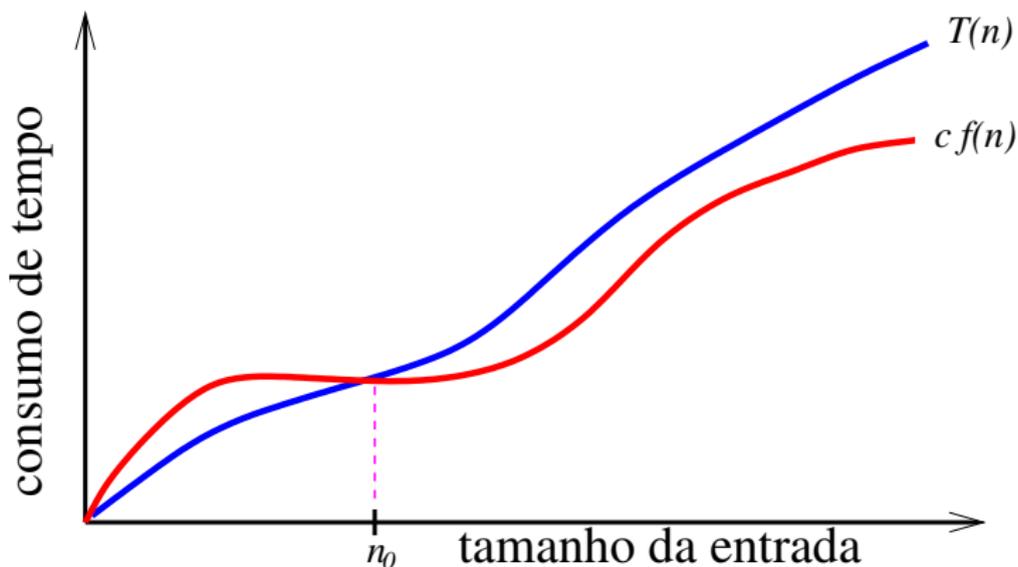


## Mais informal

$T(n) = \Omega(f(n))$  se existe  $c > 0$  tal que

$$c f(n) \leq T(n)$$

para todo  $n$  **suficientemente GRANDE**.



# Exemplos

## Exemplo 1

Se  $T(n) \geq 0.001n^2$  para todo  $n \geq 8$ , então  $T(n)$  é  $\Omega(n^2)$ .

# Exemplos

## Exemplo 1

Se  $T(n) \geq 0.001n^2$  para todo  $n \geq 8$ , então  $T(n)$  é  $\Omega(n^2)$ .

**Prova:** Aplique a definição com  $c = 0.001$  e  $n_0 = 8$ .

## Exemplo 2

O consumo de tempo do **CONTA-INVERSÕES** é  $O(n^2)$  e também  $\Omega(n^2)$ .

**CONTA-INVERSÕES** ( $p, n$ )

```
1   $c \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3      para  $j \leftarrow i + 1$  até  $n$  faça
4          se  $p[i] > p[j]$ 
5              então  $c \leftarrow c + 1$ 
6  devolva  $c$ 
```

## Exemplo 2

O consumo de tempo do **CONTA-INVERSÕES** é  $O(n^2)$  e também  $\Omega(n^2)$ .

**CONTA-INVERSÕES** ( $p, n$ )

```
1  c ← 0
2  para i ← 1 até n - 1 faça
3      para j ← i + 1 até n faça
4          se p[i] > p[j]
5              então c ← c + 1
6  devolva c
```

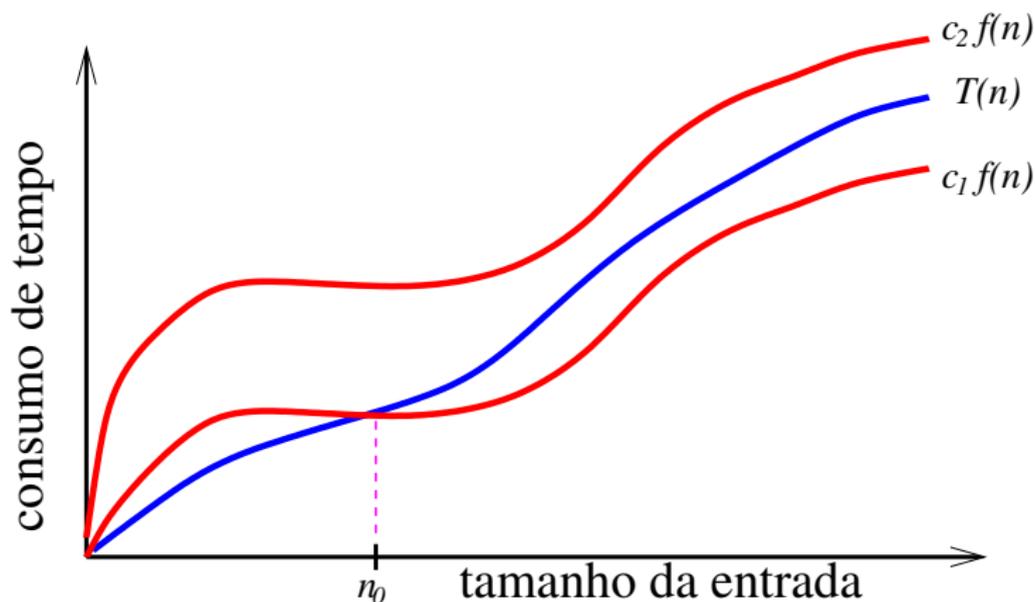
linha	todas as execuções da linha
1	= 1
2	= $n$
3	= $(n + 2)(n - 1)/2$
4	= $n(n - 1)/2$
5	≥ 0
6	= 1
<b>total</b>	<b>≥ <math>n^2 + n = \Omega(n^2)</math></b>

## Notação Theta

Sejam  $T(n)$  e  $f(n)$  funções dos inteiros no reais.

Dizemos que  $T(n)$  é  $\Theta(f(n))$  se

$T(n)$  é  $O(f(n))$  e  $T(n)$  é  $\Omega(f(n))$ .





# Intuitivamente

Comparação **assintótica**, ou seja, para  $n$  **ENORME**.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $O(f(n))$
$T(n) \geq f(n)$	$T(n)$ é $\Omega(f(n))$
$T(n) = f(n)$	$T(n)$ é $\Theta(f(n))$

## Tamanho máximo de problemas

Suponha que cada operação consome 1 microsegundo ( $1\mu s$ ).

consumo de tempo ( $\mu s$ )	Tamanho máximo de problemas ( $n$ )		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20n \lceil \lg n \rceil$	4096	166666	7826087
$2n^2$	707	5477	42426
$n^4$	31	88	244
$2^n$	19	25	31

Michael T. Goodrich e Roberto Tamassia,  
*Projeto de Algoritmos*, Bookman.

## Crescimento de algumas funções

$n$	$\lg n$	$\sqrt{n}$	$n \lg n$	$n^2$	$n^3$	$2^n$
2	1	1,4	2	4	8	4
4	2	2	8	16	64	16
8	3	2,8	24	64	512	256
16	4	4	64	256	4096	65536
32	5	5,7	160	1024	32768	4294967296
64	6	8	384	4096	262144	$1,8 \cdot 10^{19}$
128	7	11	896	16384	2097152	$3,4 \cdot 10^{38}$
256	8	16	1048	65536	16777216	$1,1 \cdot 10^{77}$
512	9	23	4608	262144	134217728	$1,3 \cdot 10^{154}$
1024	10	32	10240	1048576	$1,1 \cdot 10^9$	$1,7 \cdot 10^{308}$

## Nomes de classes $\Theta$

classe	nome
$\Theta(1)$	constante
$\Theta(\log n)$	logarítmica
$\Theta(n)$	linear
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	quadrática
$\Theta(n^3)$	cúbica
$\Theta(n^k)$ com $k \geq 1$	polinomial
$\Theta(2^n)$	exponencial
$\Theta(a^n)$ com $a > 1$	exponencial

## Palavras de Cautela

Suponha que  $\mathcal{A}$  e  $\mathcal{B}$  são algoritmos para um mesmo problema.

Suponha que o consumo de tempo de  $\mathcal{A}$  é “essencialmente”  $100n$

e que o consumo de tempo de  $\mathcal{B}$  é “essencialmente”  $n \log_{10} n$ .

## Palavras de Cautela

Suponha que  $\mathcal{A}$  e  $\mathcal{B}$  são algoritmos para um mesmo problema.  
Suponha que o consumo de tempo de  $\mathcal{A}$  é “essencialmente”  $100n$   
e que o consumo de tempo de  $\mathcal{B}$  é “essencialmente”  $n \log_{10} n$ .

$100n$  é  $\Theta(n)$  e  $n \log_{10} n$  é  $\Theta(n \lg n)$ .

Logo,  $\mathcal{A}$  é **assintoticamente** mais eficiente que  $\mathcal{B}$ .

## Palavras de Cautela

Suponha que  $\mathcal{A}$  e  $\mathcal{B}$  são algoritmos para um mesmo problema.  
Suponha que o consumo de tempo de  $\mathcal{A}$  é “essencialmente”  $100n$   
e que o consumo de tempo de  $\mathcal{B}$  é “essencialmente”  $n \log_{10} n$ .

$100n$  é  $\Theta(n)$  e  $n \log_{10} n$  é  $\Theta(n \lg n)$ .

Logo,  $\mathcal{A}$  é **assintoticamente** mais eficiente que  $\mathcal{B}$ .

$\mathcal{A}$  é mais eficiente que  $\mathcal{B}$  para  $n \geq 10^{100}$ .

$10^{100}$  = um *googol*

$\approx$  número de átomos no universo observável

= número **ENORME**

# Palavras de Cautela

## Conclusão:

Lembre das constantes e termos de baixa ordem que estão “escondidos” na notação assintótica.

Em geral um algoritmo que consome tempo  $\Theta(n \lg n)$ , e com fatores constantes razoáveis, é bem eficiente.

Um algoritmo que consome tempo  $\Theta(n^2)$  pode, algumas vezes, ser satisfatório.

Um algoritmo que consome tempo  $\Theta(2^n)$  é dificilmente aceitável.

Do ponto de vista de AA, eficiente = polinomial.

# Número de inversões

Você sabe fazer um algoritmo mais rápido para o problema do número de inversões?

# Número de inversões

Você sabe fazer um algoritmo mais rápido para o problema do número de inversões?

Note que o número de inversões pode ser  $\Theta(n^2)$ .

Portanto, para isso, não podemos contar de uma em uma as inversões, como faz o algoritmo que vimos hoje.

Temos que ser mais espertos...

# Número de inversões

Você sabe fazer um algoritmo mais rápido para o problema do número de inversões?

Note que o número de inversões pode ser  $\Theta(n^2)$ .

Portanto, para isso, não podemos contar de uma em uma as inversões, como faz o algoritmo que vimos hoje.

Temos que ser mais espertos...

**Ideia:** vamos ordenar e contar ao mesmo tempo!

# Número de inversões

Você sabe fazer um algoritmo mais rápido para o problema do número de inversões?

Note que o número de inversões pode ser  $\Theta(n^2)$ .

Portanto, para isso, não podemos contar de uma em uma as inversões, como faz o algoritmo que vimos hoje.

Temos que ser mais espertos...

**Ideia:** vamos ordenar e contar ao mesmo tempo!

**Resultado:** um algoritmo  $O(n \lg n)$  para o problema do número de inversões de uma permutação!

# Número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Queremos um algoritmo  $O(n \lg n)$  para o problema.

O número de inversões pode ser  $\Theta(n^2)$ .

Portanto, não podemos contar de uma em uma as inversões, como faz o algoritmo anterior.

## Número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Queremos um algoritmo  $O(n \lg n)$  para o problema.

O número de inversões pode ser  $\Theta(n^2)$ .

Portanto, não podemos contar de uma em uma as inversões, como faz o algoritmo anterior.

**Ideia:** Vamos ordenar e contar ao mesmo tempo!

A ordenação ajuda a contar várias inversões de uma só vez.

## Número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Queremos um algoritmo  $O(n \lg n)$  para o problema.

O número de inversões pode ser  $\Theta(n^2)$ .

Portanto, não podemos contar de uma em uma as inversões, como faz o algoritmo anterior.

**Ideia:** Vamos ordenar e contar ao mesmo tempo!

A ordenação ajuda a contar várias inversões de uma só vez.

Que algoritmo de ordenação usaremos?

## Número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Queremos um algoritmo  $O(n \lg n)$  para o problema.

O número de inversões pode ser  $\Theta(n^2)$ .

Portanto, não podemos contar de uma em uma as inversões, como faz o algoritmo anterior.

**Ideia:** Vamos ordenar e contar ao mesmo tempo!

A ordenação ajuda a contar várias inversões de uma só vez.

Que algoritmo de ordenação usaremos?

Duas opções: o MERGESORT e o HEAPSORT.

Qual deles parece mais adequado?

## Número de inversões

**Problema:** Dada uma permutação  $p[1..n]$ , determinar o número de inversões em  $p$ .

Queremos um algoritmo  $O(n \lg n)$  para o problema.

O número de inversões pode ser  $\Theta(n^2)$ .

Portanto, não podemos contar de uma em uma as inversões, como faz o algoritmo anterior.

**Ideia:** Vamos ordenar e contar ao mesmo tempo!

A ordenação ajuda a contar várias inversões de uma só vez.

Que algoritmo de ordenação usaremos?

Duas opções: o MERGESORT e o HEAPSORT.

Qual deles parece mais adequado?

**Resposta:** o MERGESORT.

# Merge-Sort

Rearranja  $A[p..r]$ , 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 ( $A, p, q, r$ )
```

Método: Divisão e conquista.

# Intercalação

**Problema:** Dados  $A[p..q]$  e  $A[q+1..r]$  crescentes, rearranjar  $A[p..r]$  de modo que ele fique em ordem crescente.

Para que valores de  $q$  o problema faz sentido?

Entra:

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

# Intercalação

**Problema:** Dados  $A[p..q]$  e  $A[q+1..r]$  crescentes, rearranjar  $A[p..r]$  de modo que ele fique em ordem crescente.

Para que valores de  $q$  o problema faz sentido?

Entra:

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

Sai:

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

# Intercalação

INTERCALA ( $A, p, q, r$ )

```
0   ▷  $B[p..r]$  é um vetor auxiliar
1   para  $i \leftarrow p$  até  $q$  faça
2        $B[i] \leftarrow A[i]$ 
3   para  $j \leftarrow q + 1$  até  $r$  faça
4        $B[r + q + 1 - j] \leftarrow A[j]$ 
5    $i \leftarrow p$ 
6    $j \leftarrow r$ 
7   para  $k \leftarrow p$  até  $r$  faça
8       se  $B[i] \leq B[j]$ 
9           então  $A[k] \leftarrow B[i]$ 
10               $i \leftarrow i + 1$ 
11       senão  $A[k] \leftarrow B[j]$ 
12               $j \leftarrow j - 1$ 
```

## Adaptação do Merge-Sort

Conta o número de inversões de  $A[p..r]$ , com  $p \leq r$ , e rearranja  $A[p..r]$  em ordem crescente.

```
CONTA-E-ORDENA ( $A, p, r$ )
1   se  $p \geq r$ 
2       então devolva 0
3       senão  $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
4            $c \leftarrow$  CONTA-E-ORDENA ( $A, p, q$ ) +
5                       CONTA-E-ORDENA ( $A, q + 1, r$ ) +
6                       CONTA-E-INTERCALA ( $A, p, q, r$ )
7       devolva  $c$ 
```

Método: Divisão e conquista.

## Contagem na intercalação

CONTA-E-INTERCALA ( $A, p, q, r$ )

1 para  $i \leftarrow p$  até  $q$  faça

2      $B[i] \leftarrow A[i]$

3 para  $j \leftarrow q + 1$  até  $r$  faça

4      $B[r + q + 1 - j] \leftarrow A[j]$

5  $i \leftarrow p$

6  $j \leftarrow r$

7  $c \leftarrow 0$

▷ inicializa o contador

8 para  $k \leftarrow p$  até  $r$  faça

9     se  $B[i] \leq B[j]$

10         então  $A[k] \leftarrow B[i]$

11              $i \leftarrow i + 1$

12         senão  $A[k] \leftarrow B[j]$

13              $j \leftarrow j - 1$

14          $c \leftarrow c + (q - i + 1)$

▷ conta inversões

15 devolva  $c$

# Simulação

A

<i>p</i>	22	33	55	77	<i>q</i>	99	11	44	66	<i>r</i>	88
----------	----	----	----	----	----------	----	----	----	----	----------	----

B

--	--	--	--	--	--	--	--	--	--	--	--

$c = 0$

# Simulação



$c = 0$

# Simulação



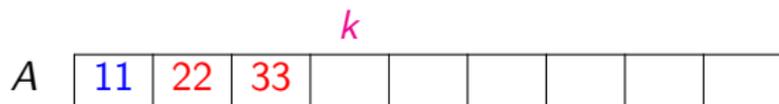
$$c = 0 + 5 = 5$$

# Simulação



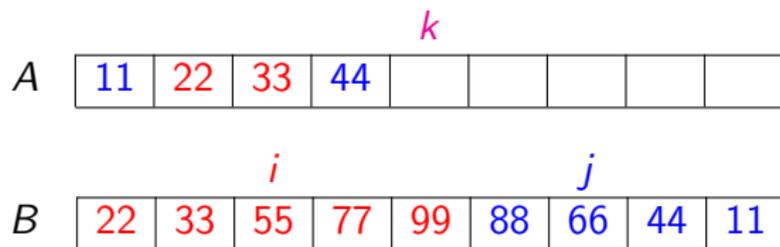
$c = 5$

# Simulação



$c = 5$

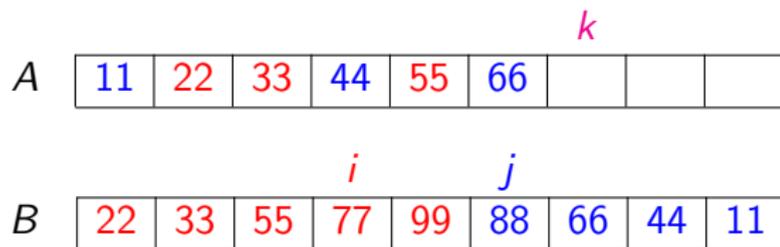
# Simulação



$$c = 5 + 3 = 8$$



# Simulação



$$c = 8 + 2 = 10$$

# Simulação

A

11	22	33	44	55	66	77		
----	----	----	----	----	----	----	--	--

*k*

B

22	33	55	77	99	88	66	44	11
----	----	----	----	----	----	----	----	----

*i*      *j*

$$c = 10$$

# Simulação

A

11	22	33	44	55	66	77	88	
----	----	----	----	----	----	----	----	--

*k*

*i = j*

B

22	33	55	77	99	88	66	44	11
----	----	----	----	----	----	----	----	----

$$c = 10 + 1 = 11$$

# Simulação

A

11	22	33	44	55	66	77	88	99
----	----	----	----	----	----	----	----	----

B

22	33	55	77	99	88	66	44	11
----	----	----	----	----	----	----	----	----

*j*   *i*

$c = 11$

## Contagem na intercalação

CONTA-E-INTERCALA ( $A, p, q, r$ )

1 para  $i \leftarrow p$  até  $q$  faça

2      $B[i] \leftarrow A[i]$

3 para  $j \leftarrow q + 1$  até  $r$  faça

4      $B[r + q + 1 - j] \leftarrow A[j]$

5  $i \leftarrow p$

6  $j \leftarrow r$

7  $c \leftarrow 0$

▷ inicializa o contador

8 para  $k \leftarrow p$  até  $r$  faça

9     se  $B[i] \leq B[j]$

10         então  $A[k] \leftarrow B[i]$

11              $i \leftarrow i + 1$

12         senão  $A[k] \leftarrow B[j]$

13              $j \leftarrow j - 1$

14          $c \leftarrow c + (q - i + 1)$

▷ conta inversões

15 devolva  $c$

## Consumo de tempo

Quanto tempo consome em função de  $n := r - p + 1$ ?

linha	consumo de todas as execuções da linha
1	$O(n)$
2	$O(n)$
3	$O(n)$
4	$O(n)$
5-7	$O(1)$
8	$O(n)$
9	$O(n)$
10-14	$O(n)$
15	$O(1)$
<b>total</b>	$O(7n + 2) = O(n)$

## Conclusão

O algoritmo **CONTA-E-INTERCALA** consome  $O(n)$  unidades de tempo.

Também escreve-se

O algoritmo **CONTA-E-INTERCALA** consome tempo  $O(n)$ .

# Análise do Conta-E-Ordena

Seja  $T(n)$  o tempo consumido pelo **CONTA-E-ORDENA**.

## Análise do Conta-E-Ordena

Seja  $T(n)$  o tempo consumido pelo CONTA-E-ORDENA.

Vale a seguinte recorrência para  $T(n)$ :

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

## Análise do Conta-E-Ordena

Seja  $T(n)$  o tempo consumido pelo CONTA-E-ORDENA.

Vale a seguinte recorrência para  $T(n)$ :

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

**Solução:**  $T(n) = O(n \lg n)$ .

Prova?

## Análise do Conta-E-Ordena

Seja  $T(n)$  o tempo consumido pelo CONTA-E-ORDENA.

Vale a seguinte recorrência para  $T(n)$ :

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

**Solução:**  $T(n) = O(n \lg n)$ .

Prova?

Considera-se a recorrência simplificada

$$T(n) = 2 T(n/2) + n$$

definida apenas para  $n$  potência de 2.

## Análise do Conta-E-Ordena

Seja  $T(n)$  o tempo consumido pelo CONTA-E-ORDENA.

Vale a seguinte recorrência para  $T(n)$ :

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

**Solução:**  $T(n) = O(n \lg n)$ .

Prova?

Considera-se a recorrência simplificada

$$T(n) = 2 T(n/2) + n$$

definida apenas para  $n$  potência de 2.

Prova-se por indução em  $n$  que  $T(n) = n + n \lg n = O(n \lg n)$ .

## Prova

**Afirmção:** A recorrência

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2T(n/2) + n & \text{se } n \geq 2, n \text{ potência de } 2 \end{cases}$$

tem como solução  $T(n) = n + n \lg n$ .

## Prova

**Afirmação:** A recorrência

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2T(n/2) + n & \text{se } n \geq 2, n \text{ potência de } 2 \end{cases}$$

tem como solução  $T(n) = n + n \lg n$ .

**Prova:** Por indução em  $n$ .

Base:  $n = 1$        $T(1) = 1 = 1 + 1 \cdot 0 = 1 + 1 \lg 1$ .

# Prova

**Afirmação:** A recorrência

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2 T(n/2) + n & \text{se } n \geq 2, n \text{ potência de } 2 \end{cases}$$

tem como solução  $T(n) = n + n \lg n$ .

**Prova:** Por indução em  $n$ .

Base:  $n = 1$        $T(1) = 1 = 1 + 1 \cdot 0 = 1 + 1 \lg 1$ .

Passo:  $n \geq 2$

$$\begin{aligned} T(n) &= 2 T(n/2) + n \\ &= 2 (n/2 + (n/2) \lg(n/2)) + n && \text{por indução} \\ &= 2n + n \lg(n/2) \\ &= 2n + n(\lg n - 1) \\ &= n + n \lg n. \end{aligned}$$

## Resolução de recorrências

Mas como descobrimos que  $T(n) = n + n \lg n$ ?

## Resolução de recorrências

Mas como descobrimos que  $T(n) = n + n \lg n$ ? No chute!

## Resolução de recorrências

Mas como descobrimos que  $T(n) = n + n \lg n$ ? No chute!

Uma maneira de se obter um “chute” de solução de recorrência é desenrolando a recorrência.

## Resolução de recorrências

Mas como descobrimos que  $T(n) = n + n \lg n$ ? No chute!

Uma maneira de se obter um “chute” de solução de recorrência é desenrolando a recorrência.

$$\begin{aligned}T(n) &= 2 T(n/2) + n \\&= 2(2 T(n/2^2) + n/2) + n = 2^2 T(n/2^2) + 2n \\&= 2^2(2 T(n/2^3) + n/2^2) + 2n = 2^3 T(n/2^3) + 3n \\&= 2^3(2 T(n/2^4) + n/2^3) + 3n = 2^4 T(n/2^4) + 4n \\&= \dots \\&= 2^k T(n/2^k) + kn,\end{aligned}$$

onde  $k = \lg n$ . Disso concluímos que

$$T(n) = n + n \lg n.$$