Análise de Algoritmos

CLRS 2.2 e 3.1
AU 3.3, 3.4 e 3.6

Essas transparências foram adaptadas das transparências do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Notação O

Intuitivamente...

$\mathrm{O}(f(n))$ $\approx$ funções que não crescem mais rápido que $f(n)$

$\approx$ funções menores ou iguais a um múltiplo de $f(n)$

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

crescem todas com a mesma velocidade

Notação O

Intuitivamente...

$\mathrm{O}(f(n))$ $\approx$ funções que não crescem mais rápido que $f(n)$

$\approx$ funções menores ou iguais a um múltiplo de $f(n)$

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

crescem todas com a mesma velocidade

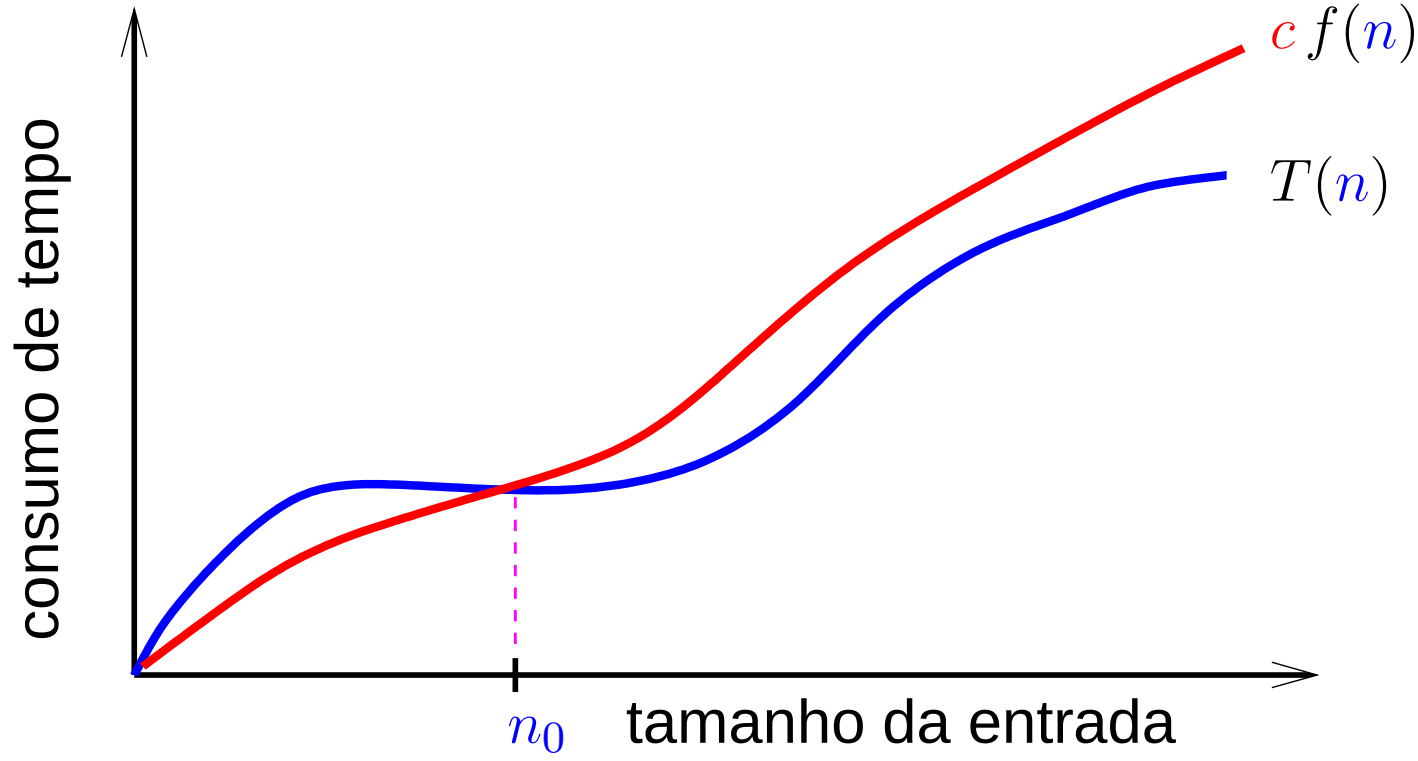
- $n^2 + 99n$ é $\mathrm{O}(n^2)$
- $33n^2$ é $\mathrm{O}(n^2)$
- $9n + 2$ é $\mathrm{O}(n^2)$
- $0{,}00001n^3 - 200n^2$ não é $\mathrm{O}(n^2)$

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros nos reais. Dizemos que $T(n)$ é $\mathrm{O}(f(n))$ se existem constantes positivas c e n_0 tais que

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

para todo $n \geq n_0$.

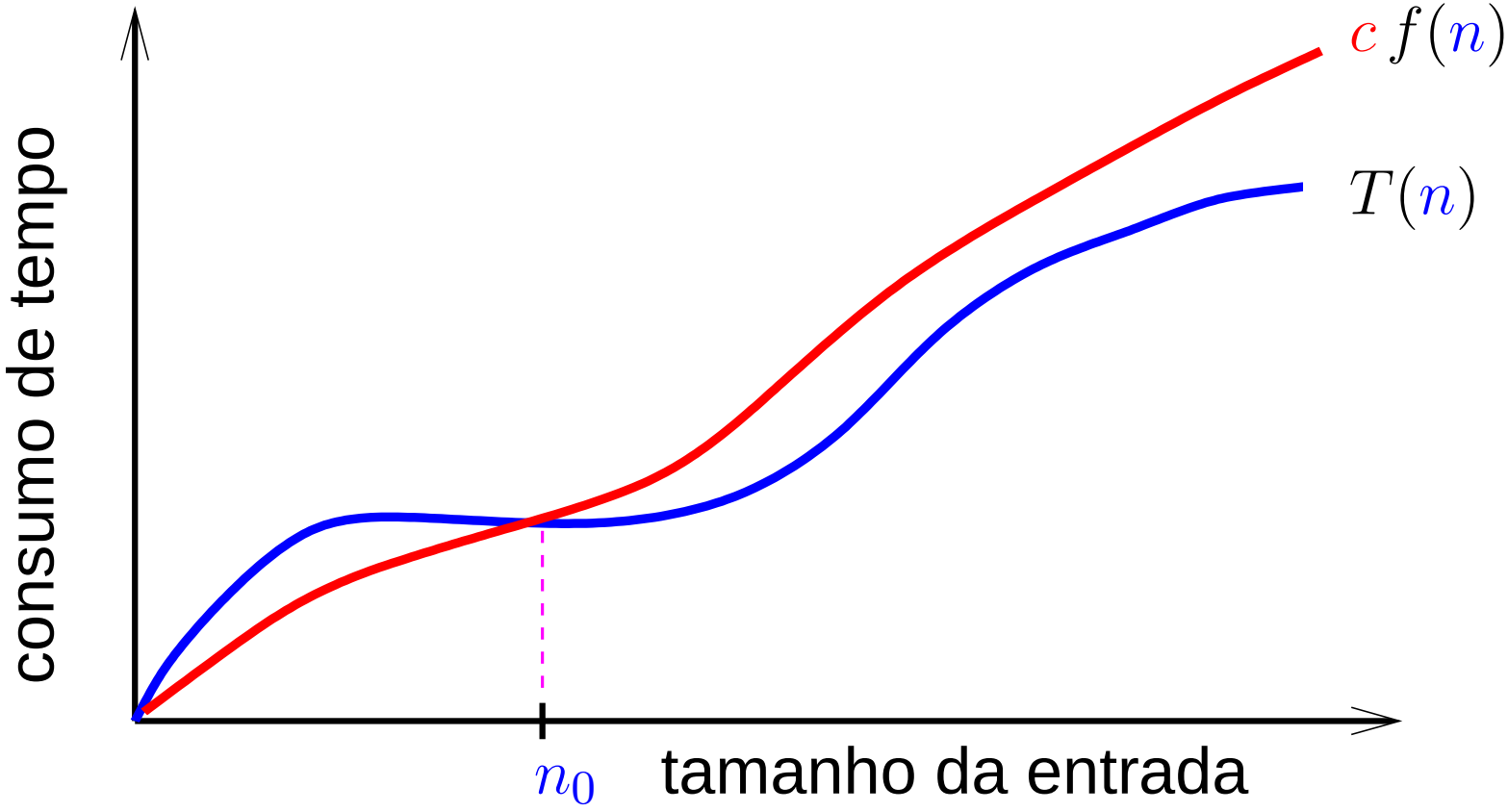


Mais informal

$T(n)$ é $\mathrm{O}(f(n))$ se existe $c > 0$ tal que

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

para todo n suficientemente GRANDE.



Exemplo

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

Exemplo

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

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

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

Exemplo

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

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

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

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

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

Uso da notação O

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

"$T(n)$ é $\mathrm{O}(f(n))$" deve ser entendido como "$T(n) \in \mathrm{O}(f(n))$".

"$T(n) = \mathrm{O}(f(n))$" deve ser entendido como "$T(n) \in \mathrm{O}(f(n))$".

"$T(n) \leq \mathrm{O}(f(n))$" é feio.

"$T(n) \geq \mathrm{O}(f(n))$" não faz sentido!

"$T(n)$ é $g(n) + \mathrm{O}(f(n))$" significa que existe constantes positivas c e n_0 tais que

$$T(n) \leq g(n) + c\, f(n)$$

para todo $n \geq n_0$.

Nomes de classes O

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

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 ← 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
```

Número de inversões

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	consumo de todas as execuções da linha
1	?
2	?
3	?
4	?
5	?
6	?
total	?

Número de inversões

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	consumo de todas as execuções da linha
1	$\mathrm{O}(1)$
2	$\mathrm{O}(n)$
3	$\mathrm{O}(n^2)$
4	$\mathrm{O}(n^2)$
5	$\mathrm{O}(n^2)$
6	$\mathrm{O}(1)$
total	$\mathrm{O}(3n^2 + n + 2) = \mathrm{O}(n^2)$

Conclusão

O algoritmo CONTA-INVERSÕES consome $\mathrm{O}(n^2)$ unidades de tempo.

Também escreve-se

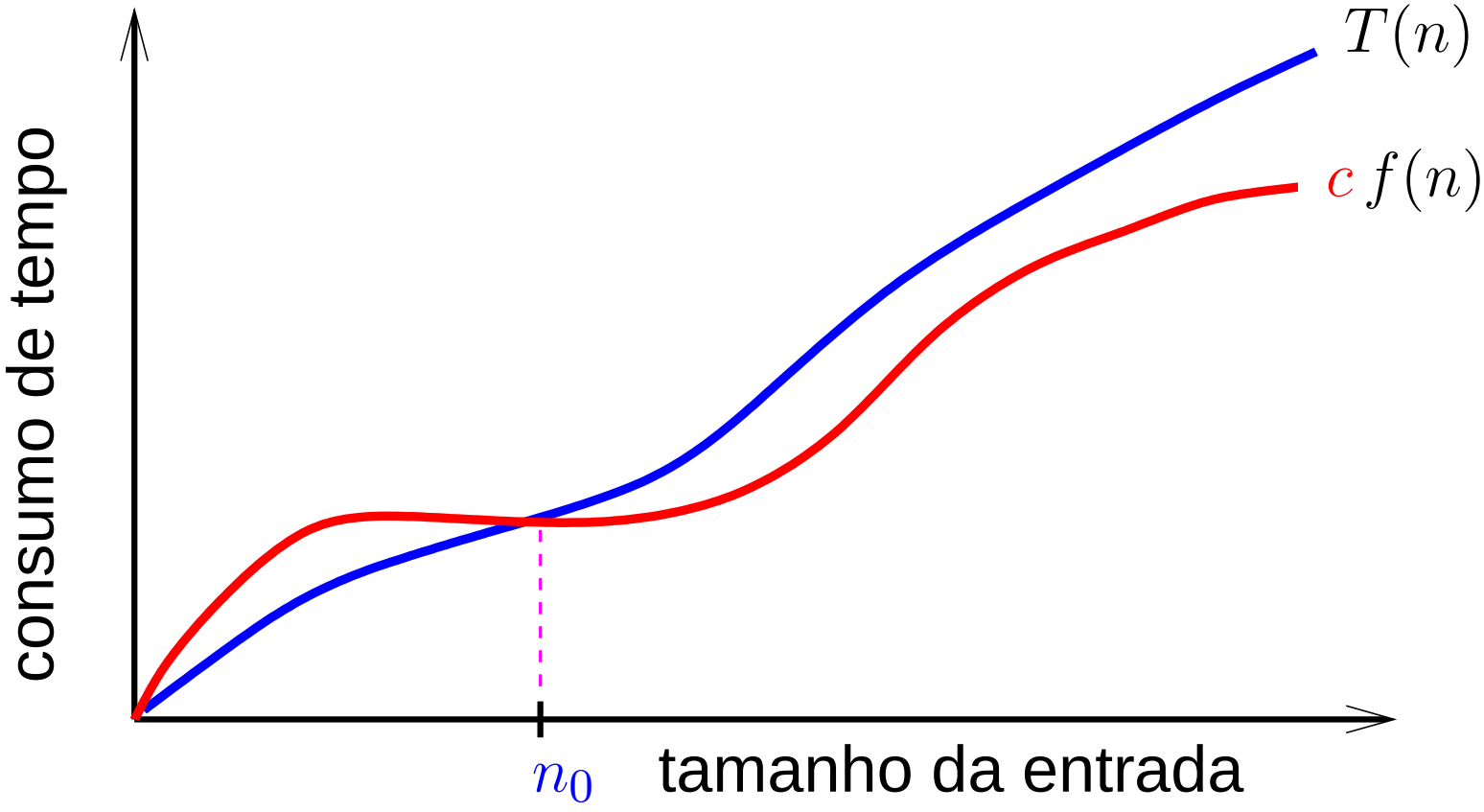
O algoritmo CONTA-INVERSÕES consome tempo $\mathrm{O}(n^2)$.

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 $n \geq n_0$.

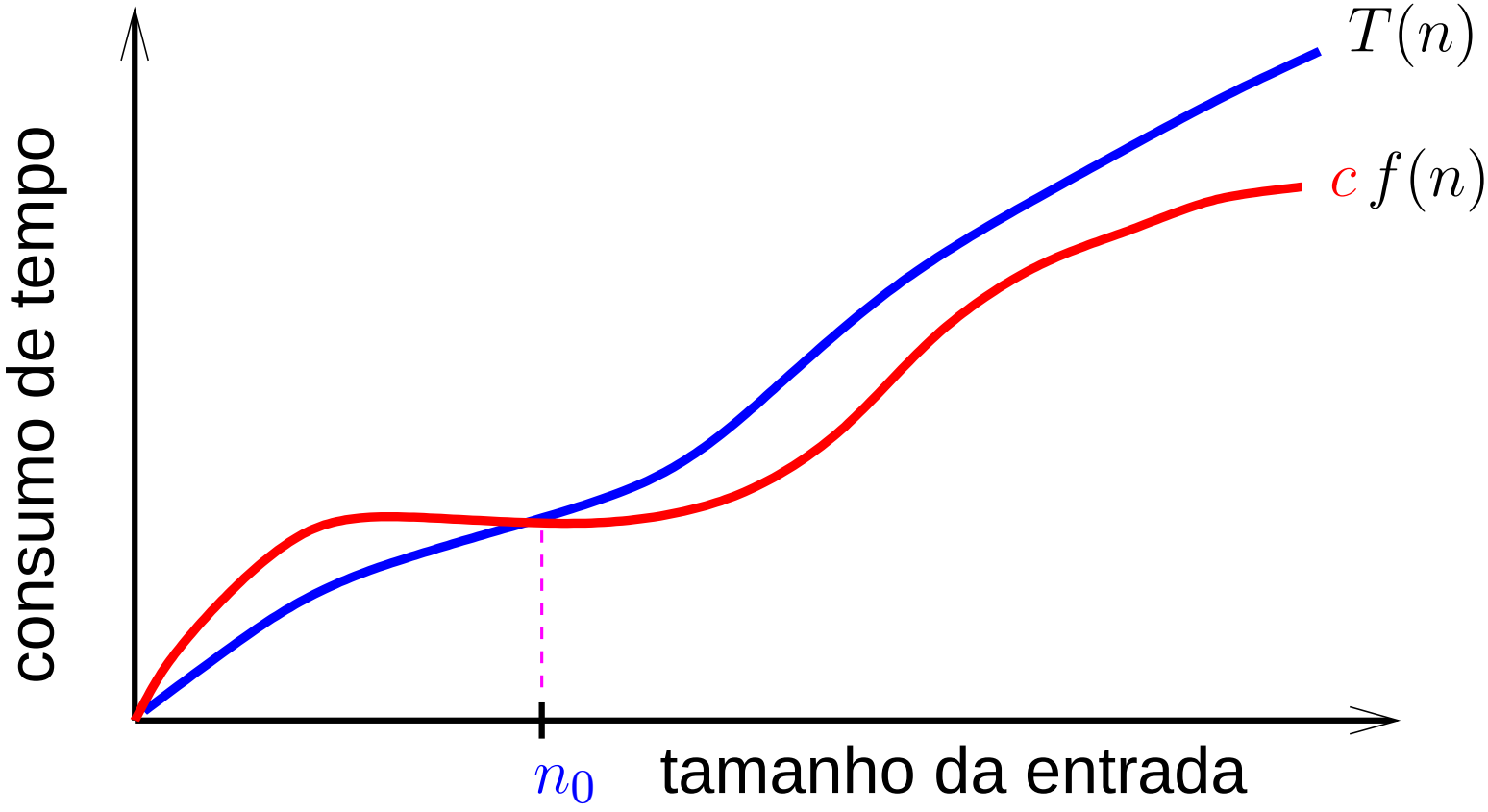


Mais informal

$T(n) = \Omega(f(n))$ 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 $\mathrm{O}(n^2)$ e também $\Omega(n^2)$.

Exemplo 2

O consumo de tempo do CONTA-INVERSÕES é $\mathrm{O}(n^2)$ 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
```

Exemplo 2

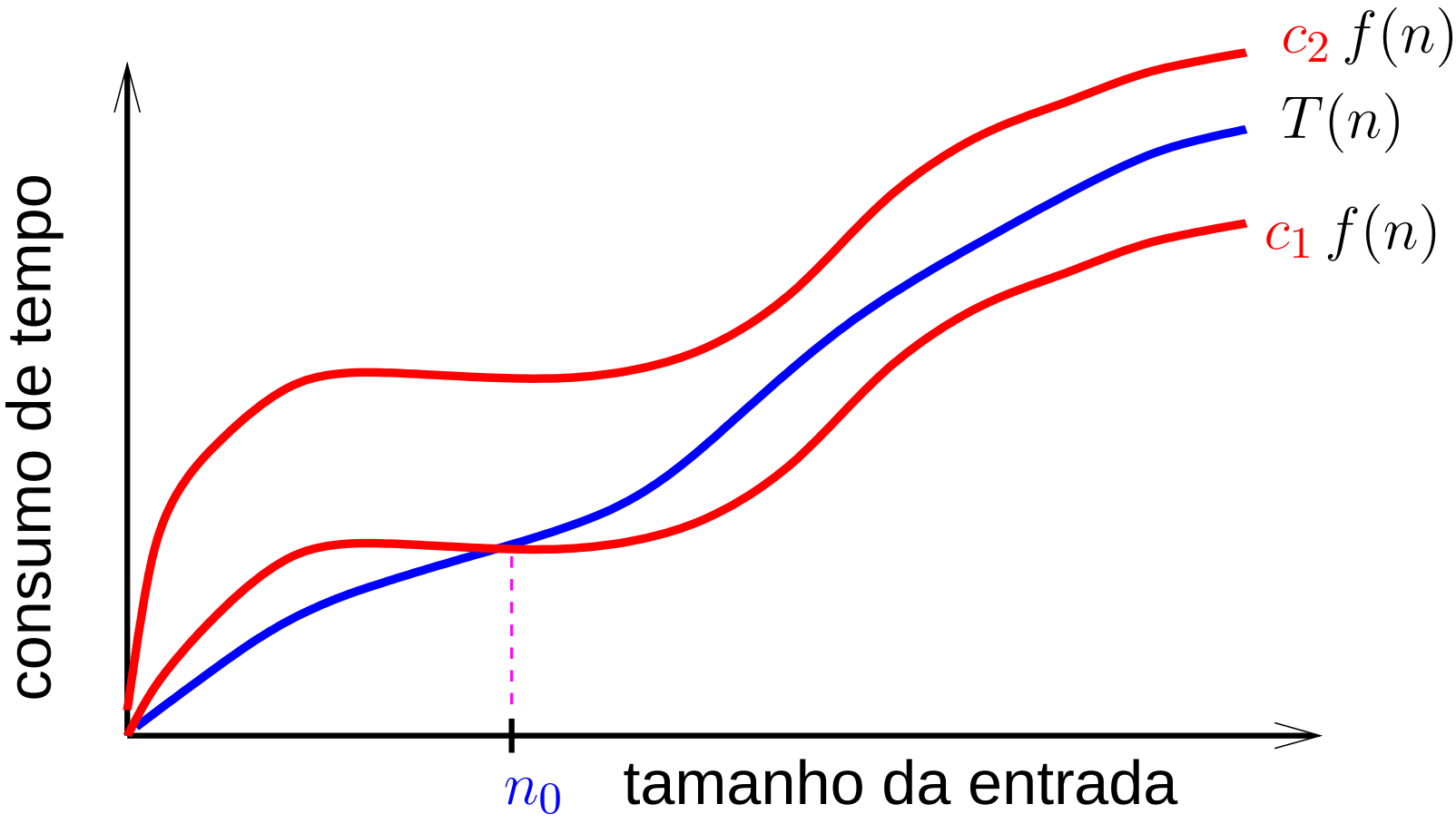
O consumo de tempo do CONTA-INVERSÕES é $\mathrm{O}(n^2)$ e também $\Omega(n^2)$.

linha		todas as execuções da linha
1	$=$	1
2	$=$	n
3	$=$	$(n+2)(n-1)/2$
4	$=$	$n(n-1)/2$
5	$\geq$	0
6	$=$	1
total	$\geq$	$n^2 + n \;=\; \Omega(n^2)$

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

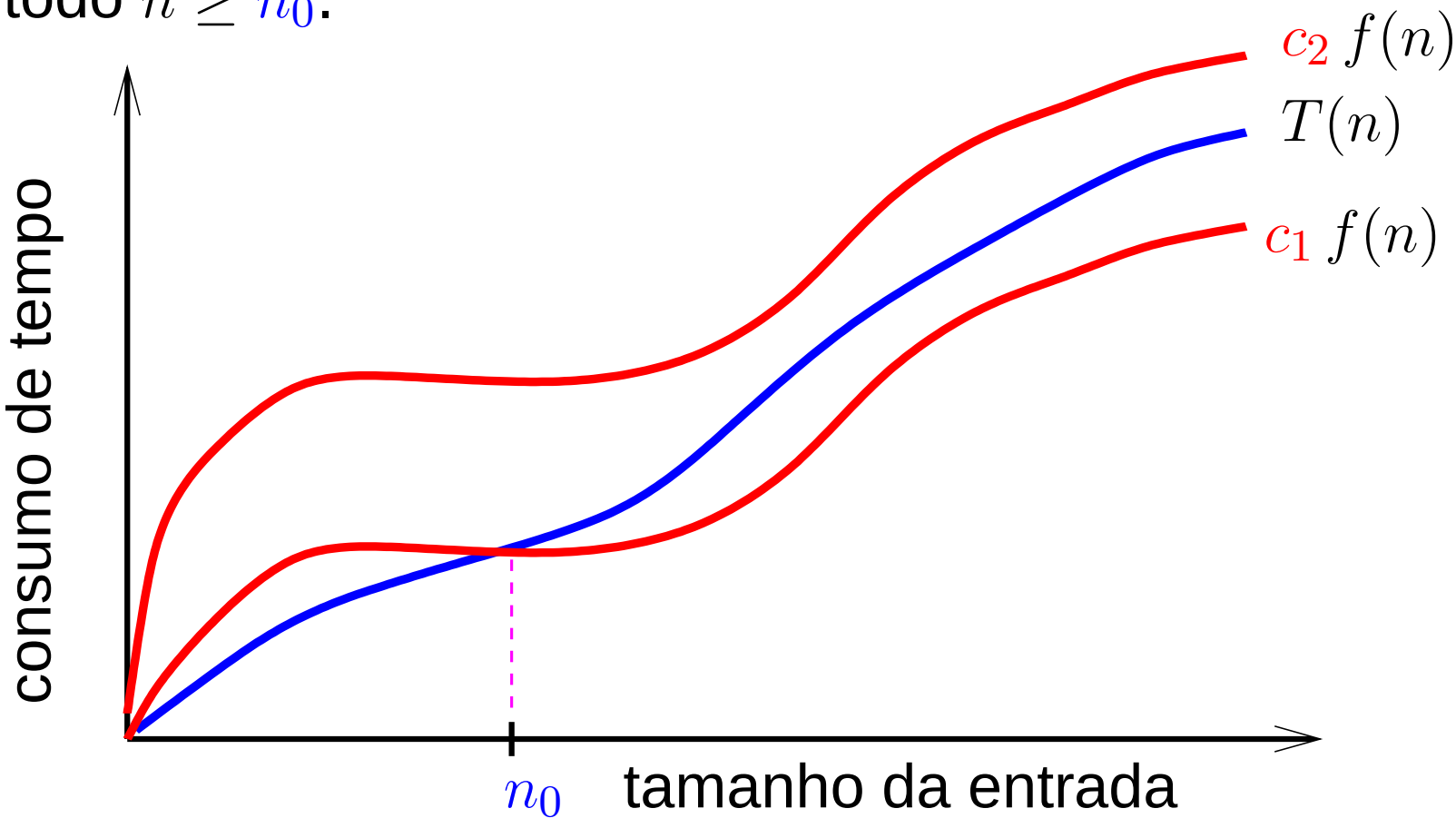


Notação Theta

Dizemos que $T(n)$ é $\Theta(f(n))$ se se existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1\, f(n) \;\leq\; T(n) \;\leq\; c_2\, f(n)$$

para todo $n \geq n_0$.



Intuitivamente

Comparação assintótica, ou seja, para n ENORME.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $\mathrm{O}(f(n))$
$T(n) \geq f(n)$	$T(n)$ 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 (μs)	Tamanho máximo de problemas (n)		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20\,n\,\lceil \lg n \rceil$	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 10^{19}
128	7	11	896	16384	2097152	3,4 10^{38}
256	8	16	1048	65536	16777216	1,1 10^{77}
512	9	23	4608	262144	134217728	1,3 10^{154}
1024	10	32	10240	1048576	1,1 10^{9}	1,7 10^{308}

Nomes de classes Θ

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" $100\,n$ 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" $100\,n$ e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

$100\,n$ 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" $100\,n$ e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

$100\,n$ 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.

Exercício da aula passada

```
1   f1 (x, y)
2       se x = 1 ou y = 1
3           devolva 0
4       senão
5           devolva f1(x − 1, y) + f1(x, y − 1) + xy
```

```
1   f2 (x, y)
2       para i ← 1 até x faça
3           t[i, 1] ← 0
4       para j ← 2 até y faça
5           t[1, j] ← 0
6       para i ← 2 até x faça
7           para j ← 2 até y faça
8               t[i, j] ← t[i − 1, j] + t[i, j − 1] + ij
9       devolva t[x, y]
```

Exercício da aula passada

Para que valores de x e y você acha que dá para começar a sentir diferença no tempo consumido por estas funções?

Exercício da aula passada

Para que valores de x e y você acha que dá para começar a sentir diferença no tempo consumido por estas funções?

- Valores menores que 30? 8 respostas.
- Entre 30 e 100? 12 respostas.
- Maiores que 100? 4 respostas.

Exercício da aula passada

Para que valores de x e y você acha que dá para começar a sentir diferença no tempo consumido por estas funções?

- Valores menores que 30? 8 respostas.
- Entre 30 e 100? 12 respostas.
- Maiores que 100? 4 respostas.

Uma frase a se pensar...

$f1$ funciona para x e y maiores que 100.

Exercício da aula passada

Matriz t inicializada com -1 em todas as posições.

```
1   f3 (x, y)
2       se t[x,y] ≠ -1
3           devolva t[x,y]
4       se x = 1 ou y = 1
5           r ← 0
6       senão
7           r ← f3(x-1,y) + f3(x,y-1) + xy
8       t[x,y] ← r
9       devolva r
```

MEMOIZAÇÃO

Exercício da aula passada

```
1    f4 (x, y)
4        para j ← 1 até y faça
5            t[j] ← 0
6        para i ← 2 até x faça
7            para j ← 2 até y faça
8                t[j] ← t[j] + t[j − 1] + ij
9        devolva t[y]
```

Mais econômica em relação à memória.

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...

Idéia: 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...

Idéia: vamos ordenar e contar ao mesmo tempo!

Método: divisão e conquista.

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...

Idéia: vamos ordenar e contar ao mesmo tempo!

Método: divisão e conquista.

Resultado: um algoritmo $\mathrm{O}(n \lg n)$ 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 $\mathrm{O}(n \lg n)$ 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 $\mathrm{O}(n \lg n)$ 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.

Idéia: 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 $\mathrm{O}(n \lg n)$ 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.

Idéia: 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 $\mathrm{O}(n \lg n)$ 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.

Idéia: 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 $\mathrm{O}(n \lg n)$ 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.

Idéia: 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.