Verão 2019 - TÓPICOS DE PROGRAMAÇÃO

Prof. Dr. Leônidas O. Brandão (coordenador)

Profa. Dra. Patrícia Alves Pereira (ministrante)

Prof. Bernardo (Monitor)

Crescimento de funções

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

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
```

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
total	$\leq$	$(3/2)n^2 + n/2 + 1$

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

Limite superior: notação O

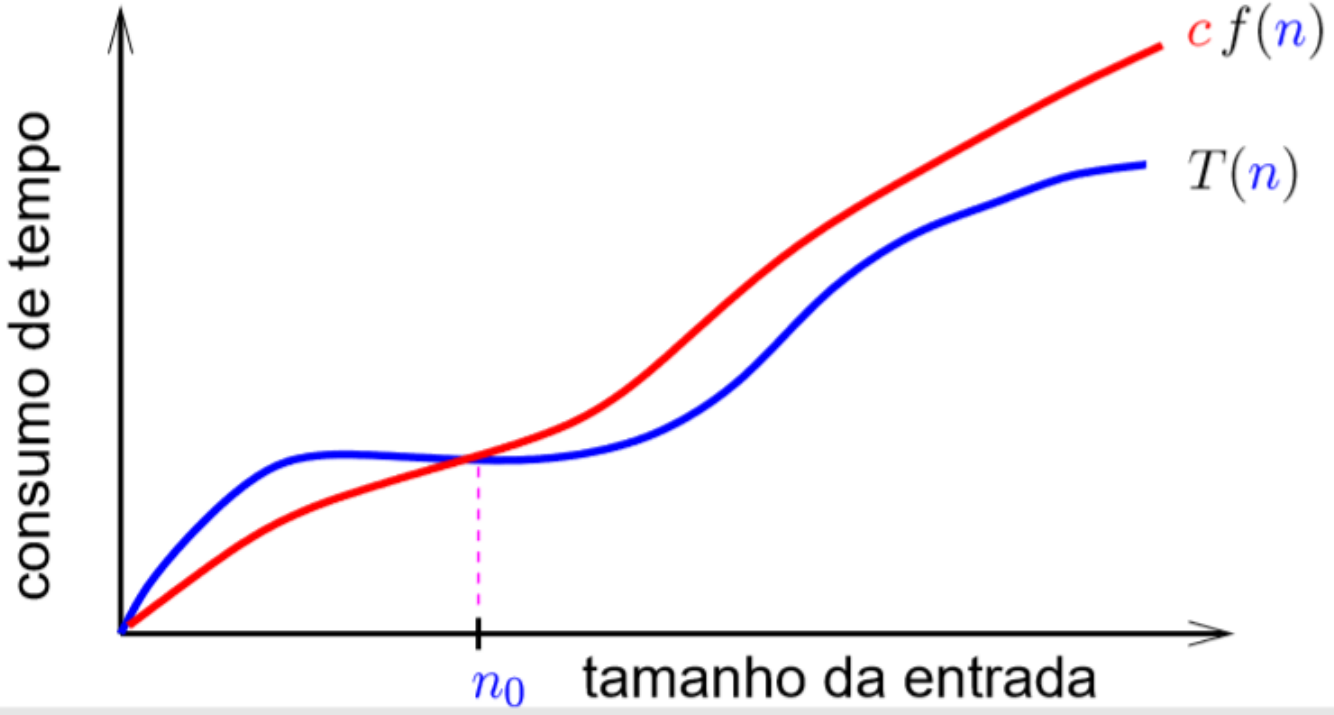
- O (Ó grande)
- limite superior da complexidade
- usada para especificar o limite superior da complexidade de um algoritmo

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

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

para todo $n \geq n_0$.



A notação O(f(n)) diz que existe um ponto n_0 tal que para todos os tamanhos de entrada *n* superiores a n_0, o custo é inferior a algum múltiplo de f(n), ou seja, f(n) cresce mais rapidamente que T(n).

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

crescem todas com a mesma velocidade

Por exemplo, dizer que o espaço alocado pela execução de um algoritmo é $O(n^2)$, significa dizer que a quantidade de memória usada é no máximo uma função quadrática da entrada.

$n^2 + 99n$ é $\mathrm{O}(n^2)$

$33n^2$ é $\mathrm{O}(n^2)$

$9n + 2$ é $\mathrm{O}(n^2)$

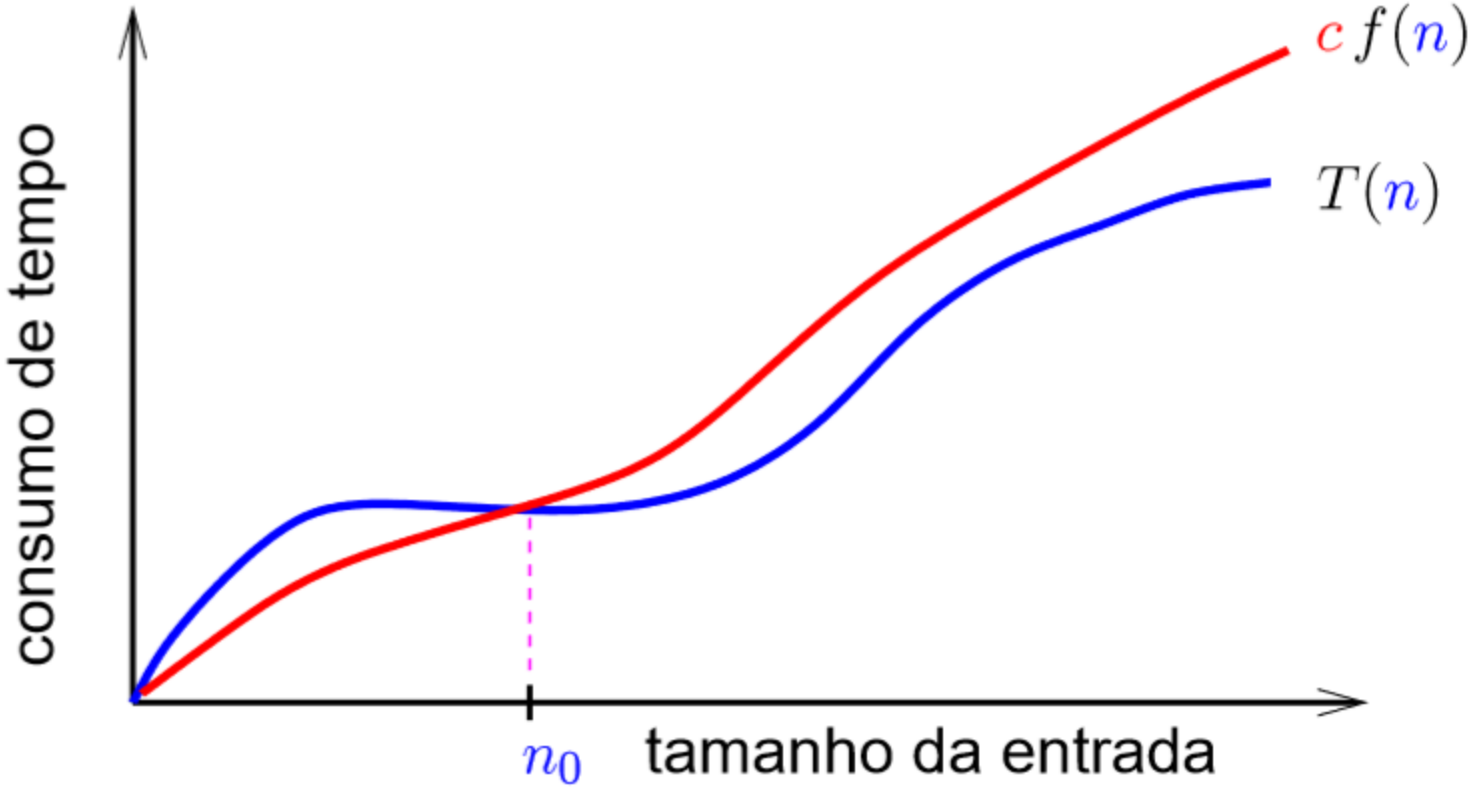
$0{,}00001n^3 - 200n^2$ não é $\mathrm{O}(n^2)$

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 $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplo 1

$10n^2$ é $\mathrm{O}(n^3)$.

Exemplos

$T(n)$ é $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplo 1

$10n^2$ é $\mathrm{O}(n^3)$.

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

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

Exemplo 3

$20n^3 + 10n \log n + 5$ 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
```

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	$O(1)$
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 $O(n^2)$.

Limite inferior: a notação Ω

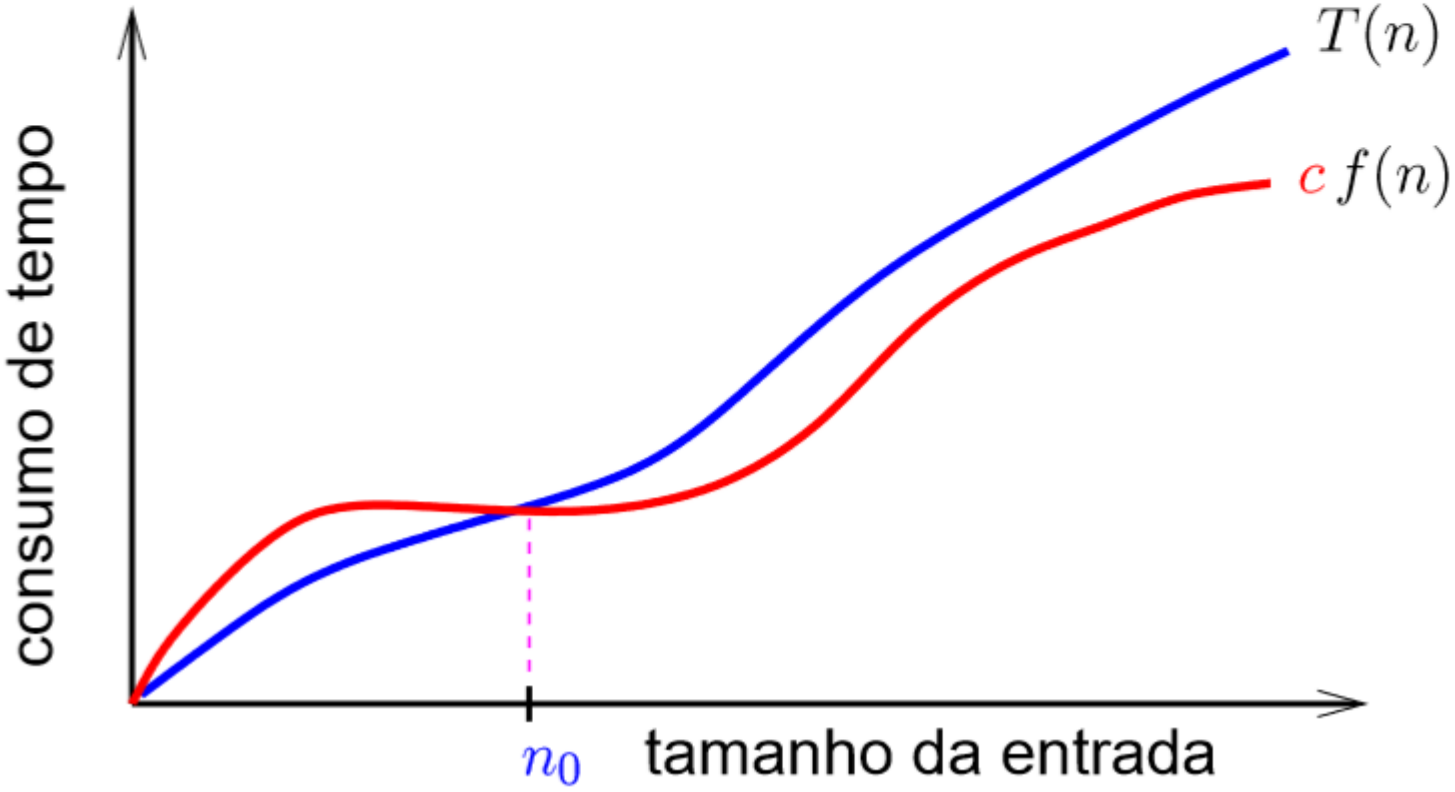
- Grande Omega
- limite inferior da complexidade
- notação usada para especificar o limite inferior da complexidade de um algoritmo

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 $n \geq n_0$.



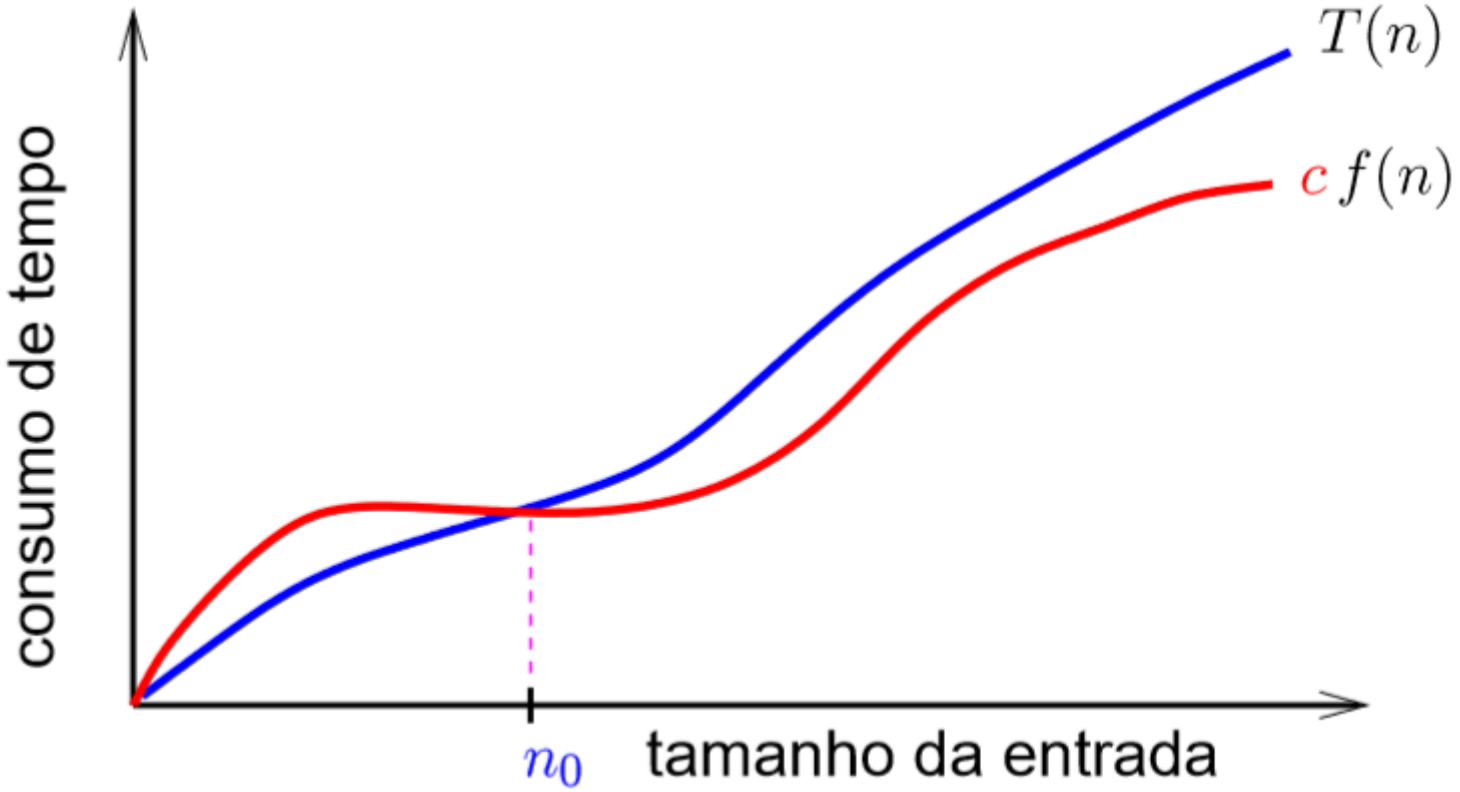
Essa definição diz que, a partir de um certo valor n_0, o custo T(n) é maior que f(n), multiplicado por um certo fator constante. Assim T(n) cresce mais rapidamente que f(n).

Mais informal

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

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

para todo n suficientemente GRANDE.



O consumo de tempo do CONTA-INVERSÕES é $O(n^2)$ 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	$\geq$	$n^2+n = \Omega(n^2)$

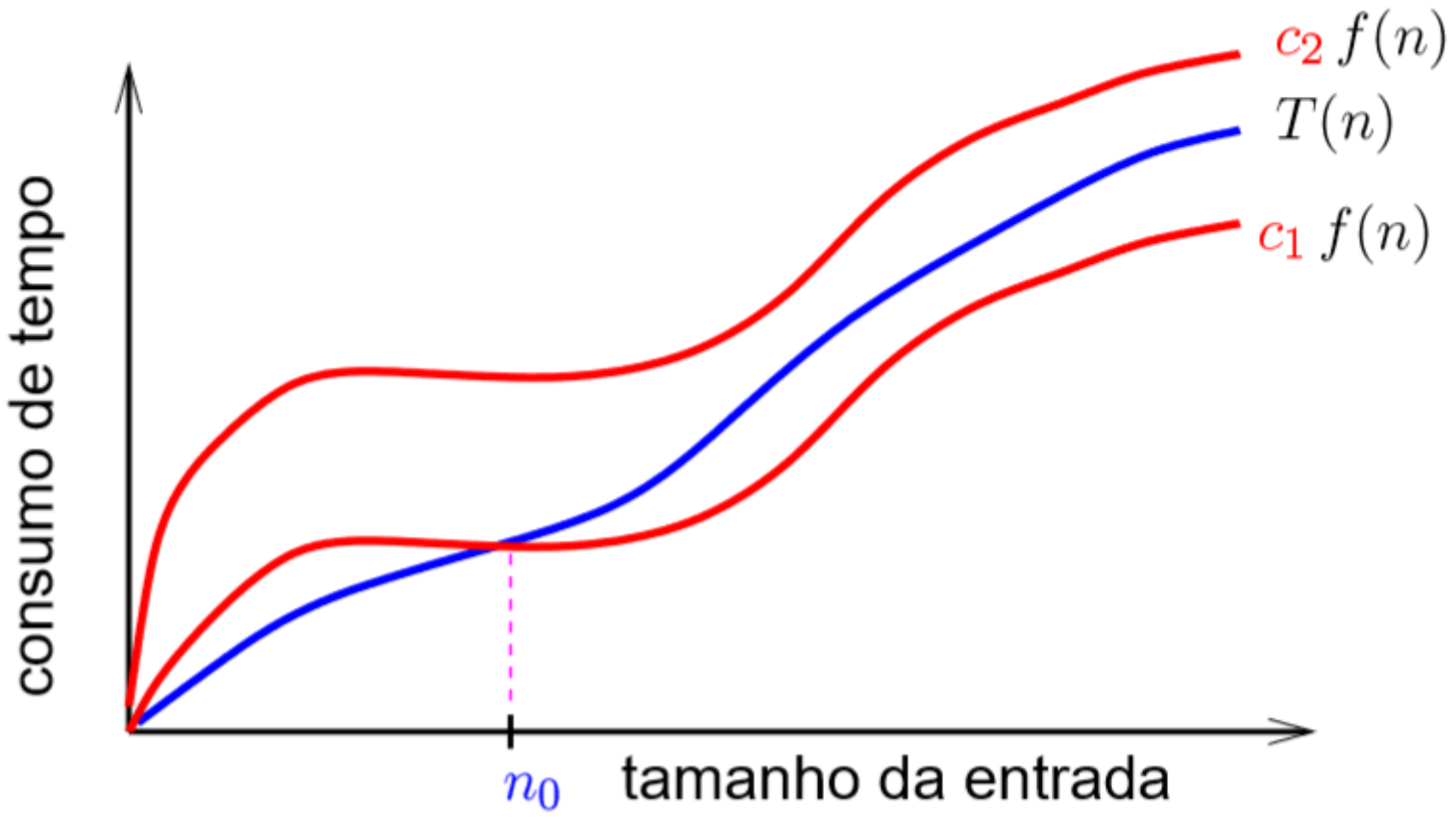
Complexidade exata: a notação Θ Theta

- notação usada para especificar exatamente a complexidade de um algoritmo

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 $\Omega(f(n))$.

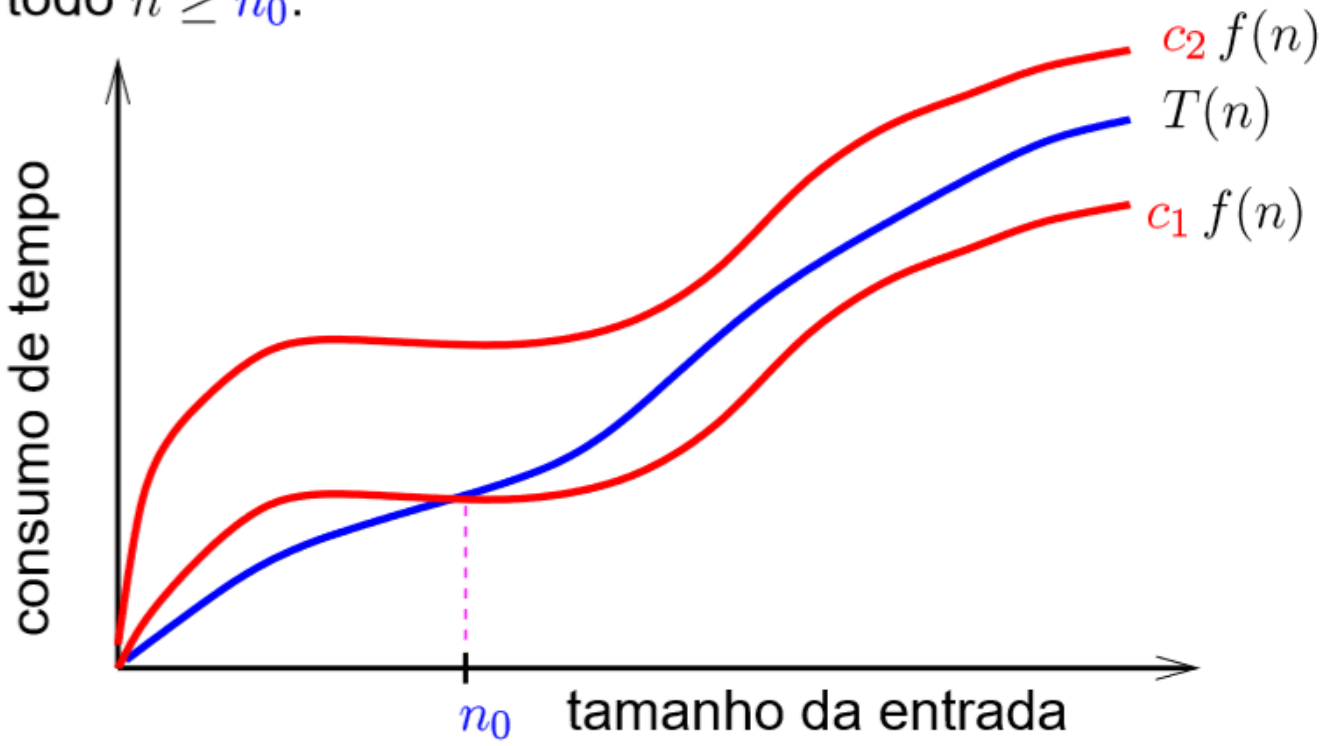


Notação Theta

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 $n \geq n_0$.



- Se um algoritmo A é Θ(f(n)), ele é ao mesmo tempo O(f(n)) e Ω(f(n)). Portanto, sua complexidade cresce tão rapidamente quanto a função f(n). f é uma medição exata da taxa de evolução da complexidade do algoritmo A.

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 $\Theta(f(n))$

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

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.

Praticando...

Qual é a complexidade superior das funções abaixo?

$$F(n) = n^2 + 10n + 20$$

$$F(n) = 250$$

$$F(n) = \lceil n / 3 \rceil$$

Praticando...

Qual é a complexidade superior das funções abaixo?

$$F(n) = n^2 + 10n + 20 = O(n^2)$$

$$F(n) = 250 = O(1)$$

$$F(n) = \lceil n / 3 \rceil = O(n)$$

Praticando...

É verdade que $2^n = \mathrm{O}(n)$?

É verdade que $n = \mathrm{O}(\lg n)$?

Justifique.

Extra: Praticando mais um pouquinho...

É verdade que $n + \sqrt{n}$ é $\mathrm{O}(n)$?

É verdade que n é $\mathrm{O}(\sqrt{n})$?

É verdade que $n^{2/3}$ é $\mathrm{O}(\sqrt{n})$?

É verdade que $\sqrt{n} + 1000$ é $\mathrm{O}(n)$?