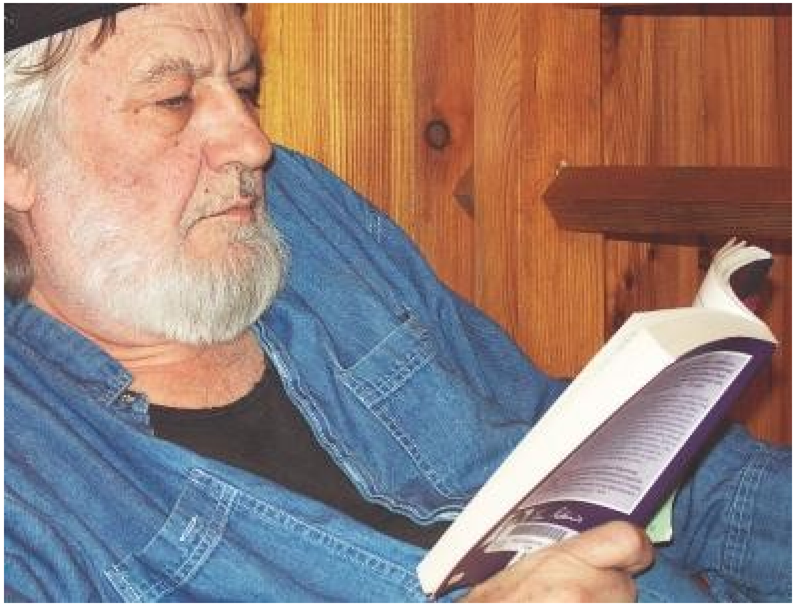
Análise de Algoritmos

Em 1965 Jack Edmonds introduz
a idéia de
Complexidade Assintótica

A idéia da “ordem”



Complexidade Assintótica

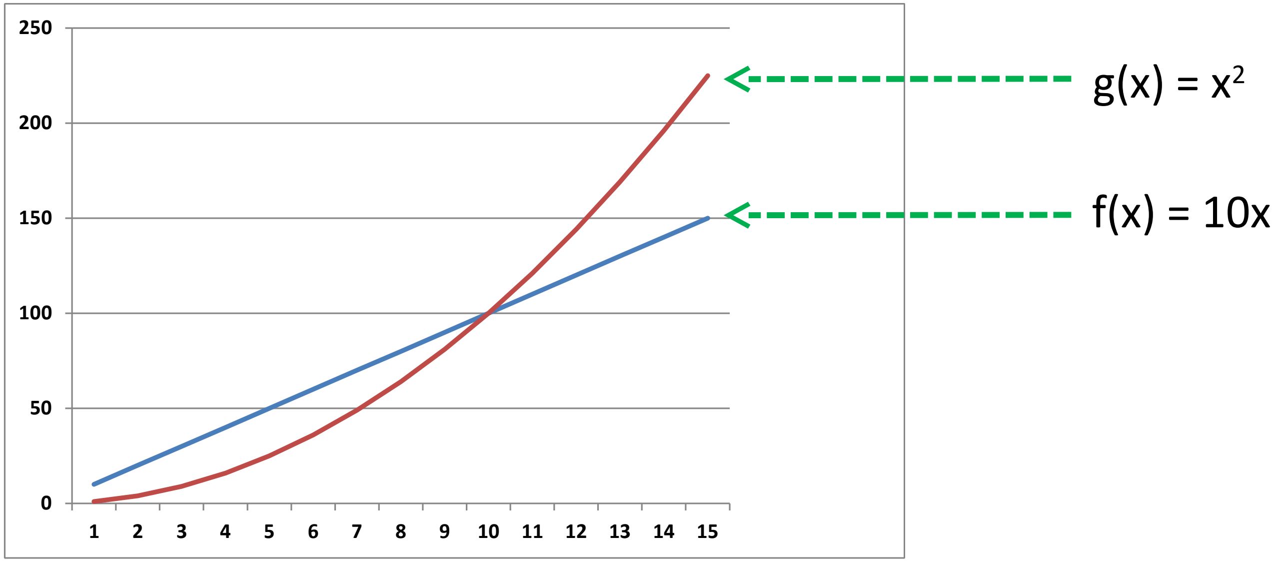
A função T(x) é chamada de
Complexidade Local

A Complexidade Assintótica
fornece limites para T(x)

Complexidade Assintótica

Notação O

Sendo duas funções f(n) e g(n) não negativas, f, g: $N \rightarrow R^m$ $m \geq 0$. Diz-se que f(n) é de ordem g(n), ou simplesmente f(n) é *O*(g(n)) se o crescimento de f(n) é, no máximo, tão rápido quanto o crescimento de g(n).

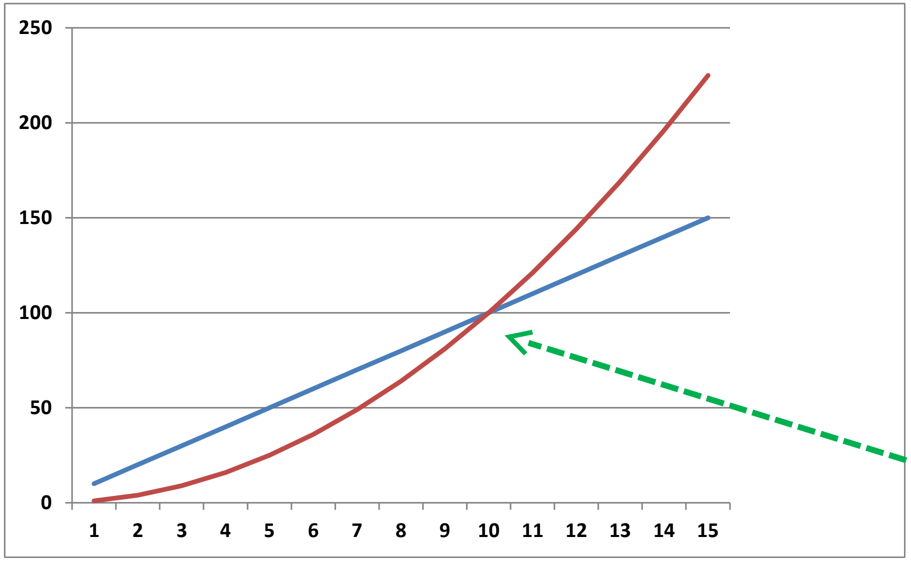


Notação *O*

f(n) é limitada superiormente por um múltiplo real positivo de g(n) para valores grandes de n. Diz-se que **f(n) é *O*(g(n))** quando:

existe uma constante real positiva c e um limite n_0 tais que

$$f(n) \leq cg(n), \text{ para todo valor de } n \geq n_0 .$$

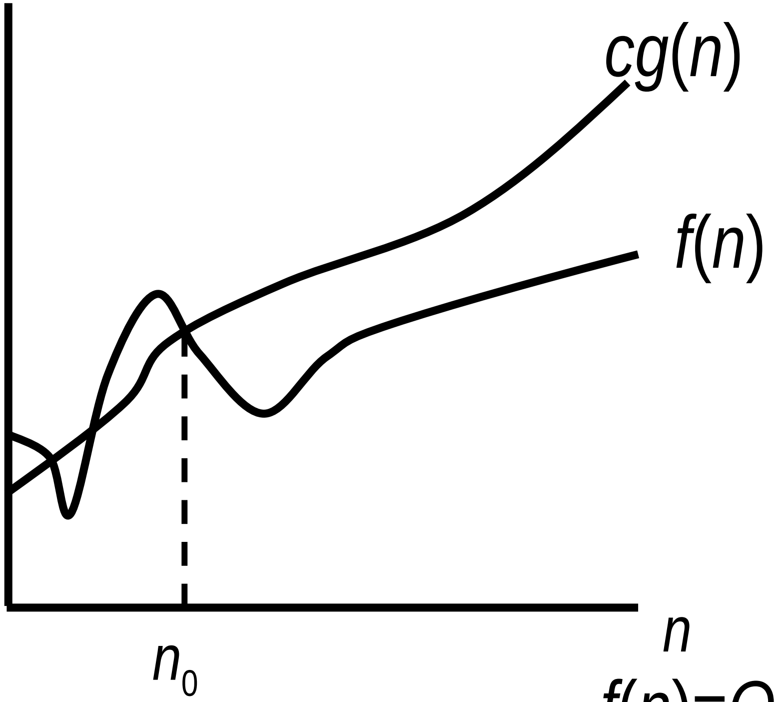


$n_0 = 10$

$c = 1$

Notação O

Esta definição equivale a dizer que $\lim_{n \to \infty} \dfrac{f(n)}{g(n)}$ existe e é finito.



$f(n)=O(g(n))$

Exemplos

$$\lim_{n\to\infty}\frac{\sqrt{n}}{n}=0 \longrightarrow \sqrt{n}=O(n)$$

$$\lim_{n\to\infty}\frac{n}{\sqrt{n}}=\infty \longrightarrow n \text{ não é } O(\sqrt{n})$$

$$\lim_{n\to\infty}\frac{n}{2n}=\frac{1}{2} \longrightarrow n=O(2n)$$

$$\lim_{n\to\infty}\frac{2n}{n}=2 \longrightarrow 2n=O(n)$$

Exemplos

1. 7n-2

   7n-2 é O(n)

   Exige-se $c > 0$ e $n_0 \geq 1$ tal que $7n-2 \leq c \bullet n$ para $n \geq n_0$

   O que é verdade para, por exemplo, $c = 7$ e $n_0 = 1$

2. $3n^3 + 20n^2 + 5$

   $3n^3 + 20n^2 + 5$ é $O(n^3)$

   Exige-se $c > 0$ e $n_0 \geq 1$ tal que $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ para $n \geq n_0$

   O que é verdade para, por exemplo, $c = 4$ e $n_0 = ?$

3. $5n^3 + 2n^2 + 3n$

Exemplos

1. $7n-2$

   $7n-2$ é $O(n)$

   Exige-se $c > 0$ e $n_0 \geq 1$ tal que $7n-2 \leq c \cdot n$ para $n \geq n_0$

   O que é verdade para, por exemplo, $c = 7$ e $n_0 = 1$

2. $3n^3 + 20n^2 + 5$

   $3n^3 + 20n^2 + 5$ é $O(n^3)$

   Exige-se $c > 0$ e $n_0 \geq 1$ tal que $3n^3 + 20n^2 + 5 \leq c \cdot n^3$ para $n \geq n_0$

   O que é verdade para, por exemplo, $c = 4$ e $n_0 = 21$

3. $5n^3 + 2n^2 + 3n$

Exemplos

1. $7n-2$

   $7n-2$ é $O(n)$

   Exige-se $c > 0$ e $n_0 \geq 1$ tal que $7n-2 \leq c \bullet n$ para $n \geq n_0$

   O que é verdade para, por exemplo, $c = 7$ e $n_0 = 1$

2. $3n^3 + 20n^2 + 5$

   $3n^3 + 20n^2 + 5$ é $O(n^3)$

   Exige-se $c > 0$ e $n_0 \geq 1$ tal que $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ para $n \geq n_0$

   O que é verdade para, por exemplo, $c = 4$ e $n_0 = 21$

3. $5n^3 + 2n^{2} + 3n$

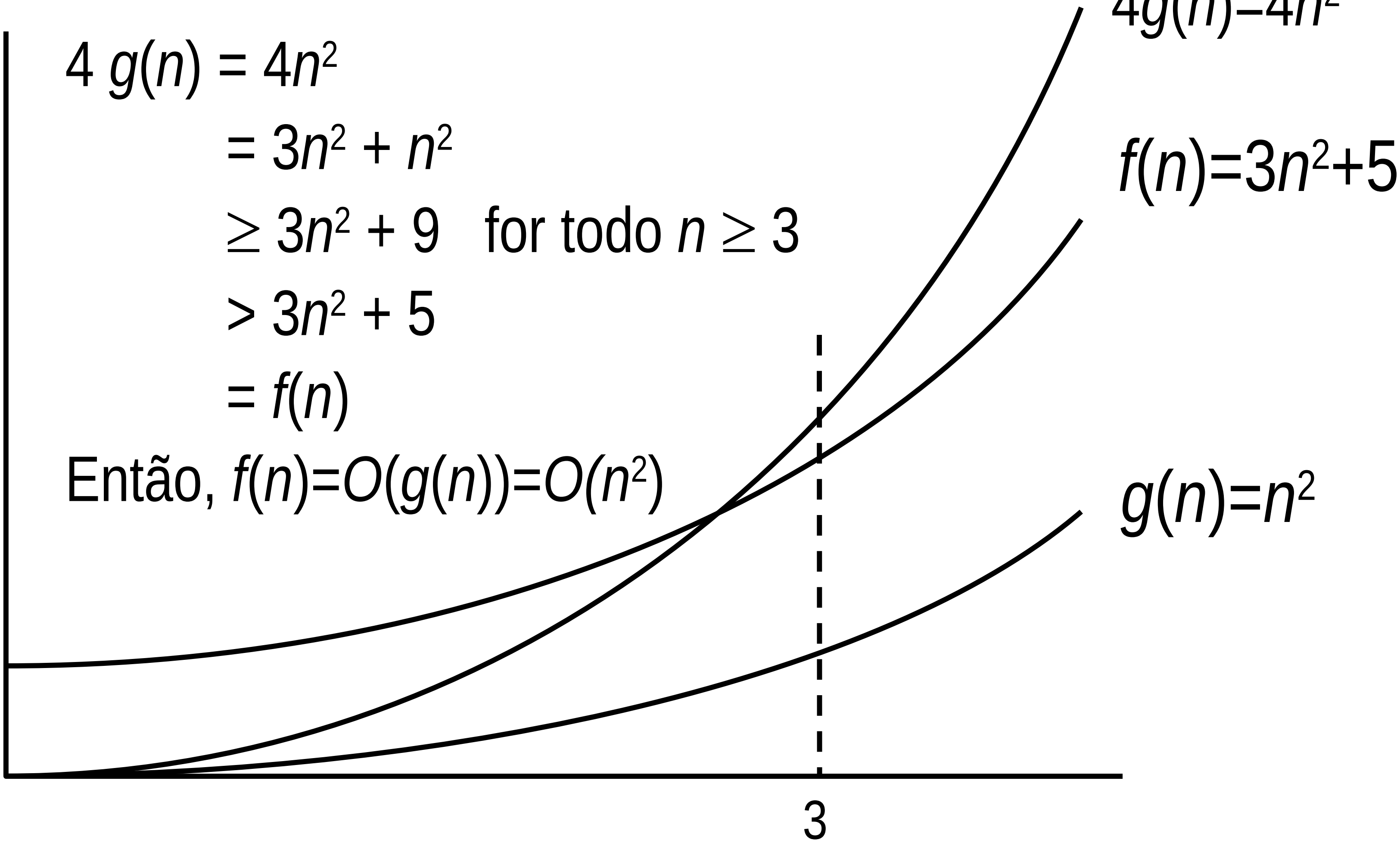
   $5n^3 + 2n^{2} + 3n$ é $O(n^3)$

   O que é verdade para, por exemplo, $c = 6$ e $n_0 = 3$

Exemplos

$$
\begin{aligned}
4\,g(n) &= 4n^2 \\
&= 3n^2 + n^2 \\
&\geq 3n^2 + 9 \quad \text{for todo } n \geq 3 \\
&> 3n^2 + 5 \\
&= f(n)
\end{aligned}
$$

Então, $f(n)=O(g(n))=O(n^2)$



Operações c/ Assintóticas

Regra da Soma

Se um algoritmo A se divide em duas partes independentes, A_1 e A_2, com complexidades dadas por $T_1(n)$ e $T_2(n)$ de *ordem* $O(f(n))$ e $O(g(n))$, respectivamente, então $T(n) = T_1(n) + T_2(n)$ e A será de ordem $O(\max\{ f(n), g(n) \})$.

Regra do Produto

Se $T_1(n)$ e $T_2(n)$ são de ordem $O(f(n))$ e $O(g(n))$, respectivamente, então $T(n) = T_1(n) \times T_2(n)$ é $O(f(n)\, g(n))$.

Resumo: Notação O

1. Se $\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \Re^{+}$ então $f(n) \in O(g(n))$ e $g(n) \in O(f(n))$

2. Se $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ então $f(n) \in O(g(n))$ e $g(n) \notin O(f(n))$

3. Se $\lim_{n \to \infty} \frac{f(n)}{g(n)} = +\infty$ então $f(n) \notin O(g(n))$ e $g(n) \in O(f(n))$

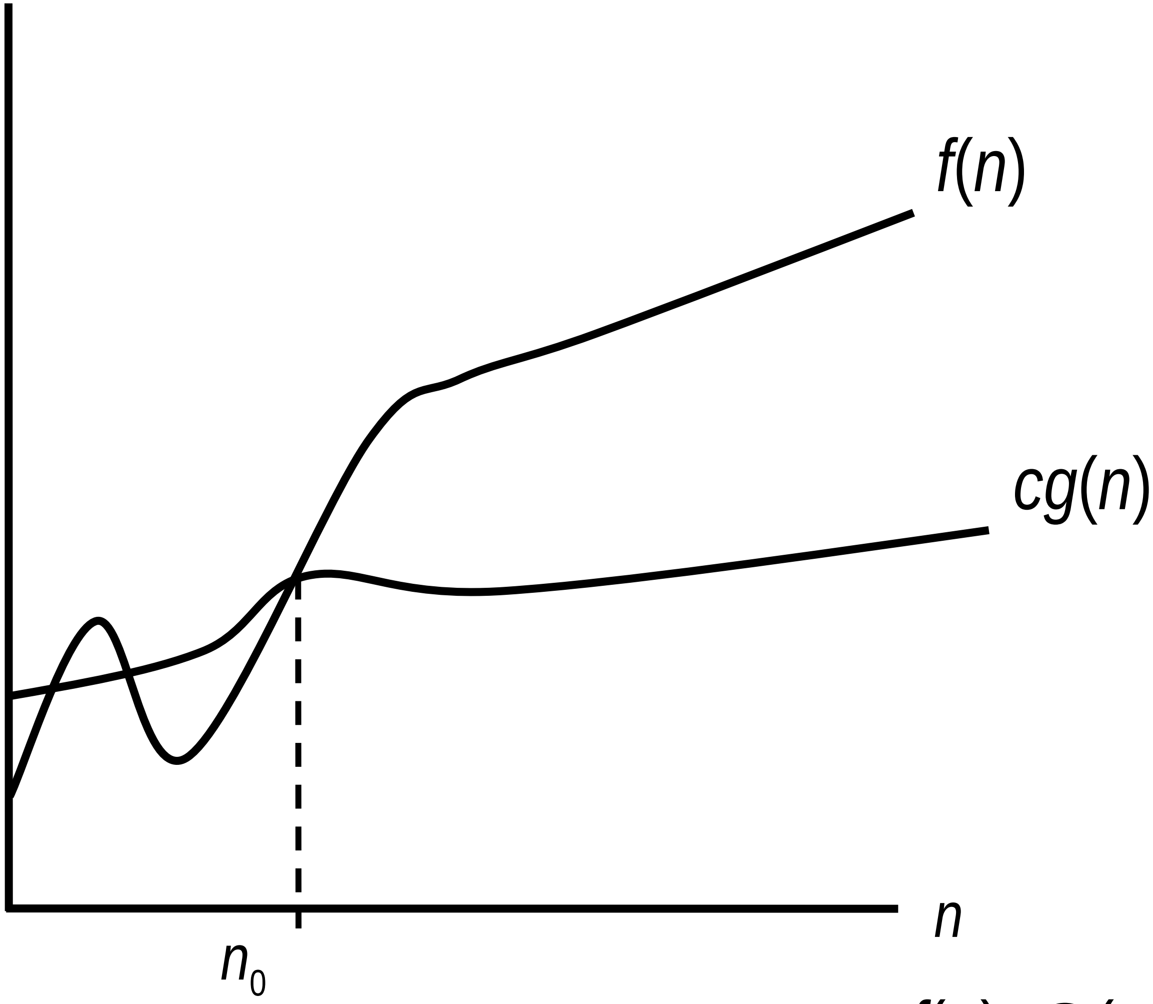
Notação Ω

Diz-se que **f(n) é Ω(g(n))** quando:

existe uma constante real positiva *d* e um limite n_0 tais que

$f(n) \geq d\, g(n)$, para todo valor de $n \geq n_0$.

Assintótica Ω



$f(n)=\Omega(g(n))$

Exemplos

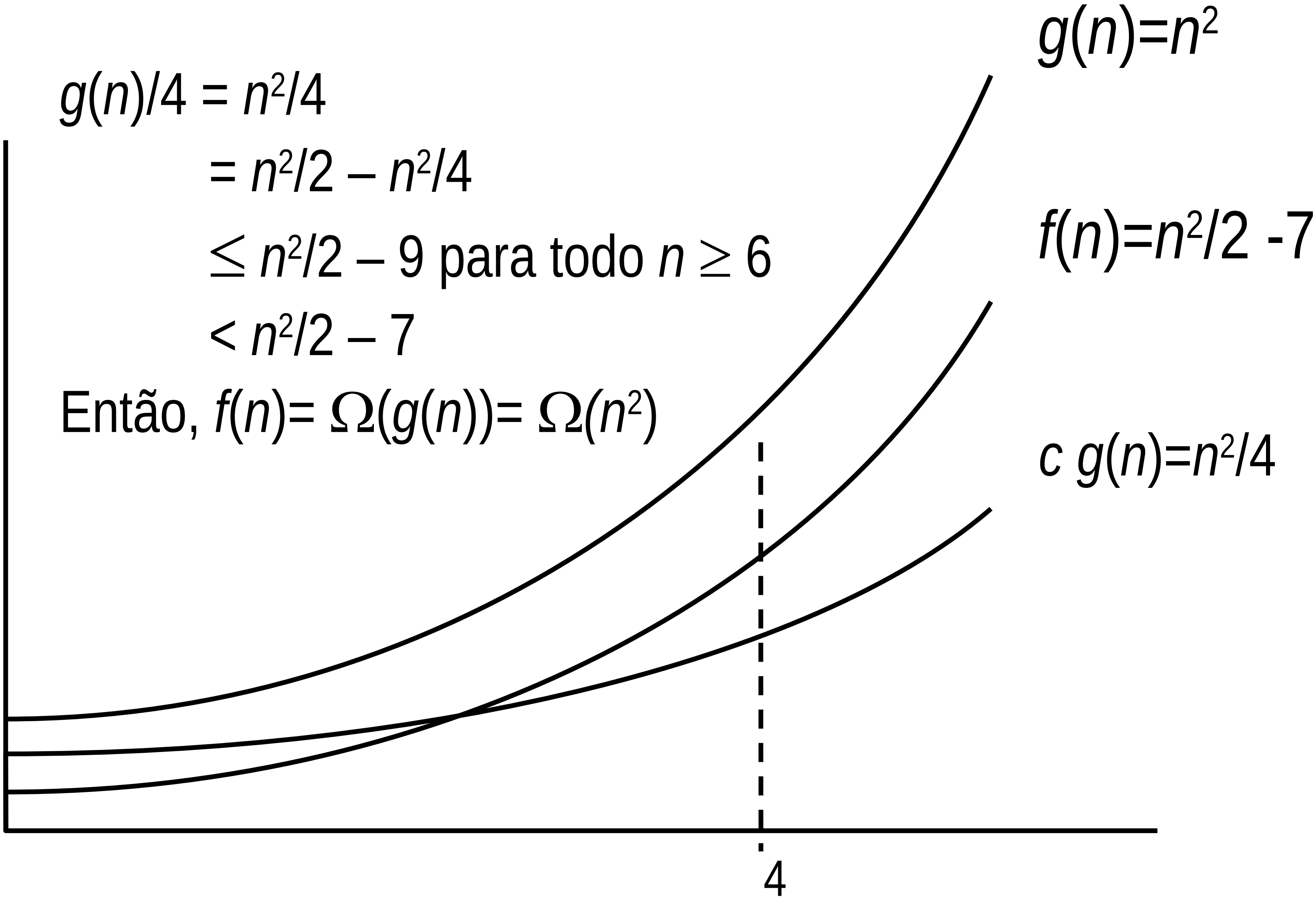
$g(n)/4 = n^2/4$

$= n^2/2 - n^2/4$

$\leq n^2/2 - 9$ para todo $n \geq 6$

$< n^2/2 - 7$

Então, $f(n) = \Omega(g(n)) = \Omega(n^2)$



Exemplos

$f(n) = 100n + 5$ é $\Omega(g(n^2))$?

Para encontrar c, n_0 tal que $0 \leq cn^2 \leq 100n + 5$

$100n + 5 \leq 100n + 5n$ (para $n \geq 1$) $= 105n$

Como n é positivo

$cn^2 \leq 105n \quad \Rightarrow cn \leq 105 \ \Rightarrow n \leq 105/c$

Contudo n não pode ser menor que uma constante!

Notação Θ

Diz-se que que **f(n)** é **Θ (g(n))**, ou que **f(n)** é da ordem exata de **g(n),** se **f(n)** é tanto ***O*(g(n))** como **Ω(g(n))**.

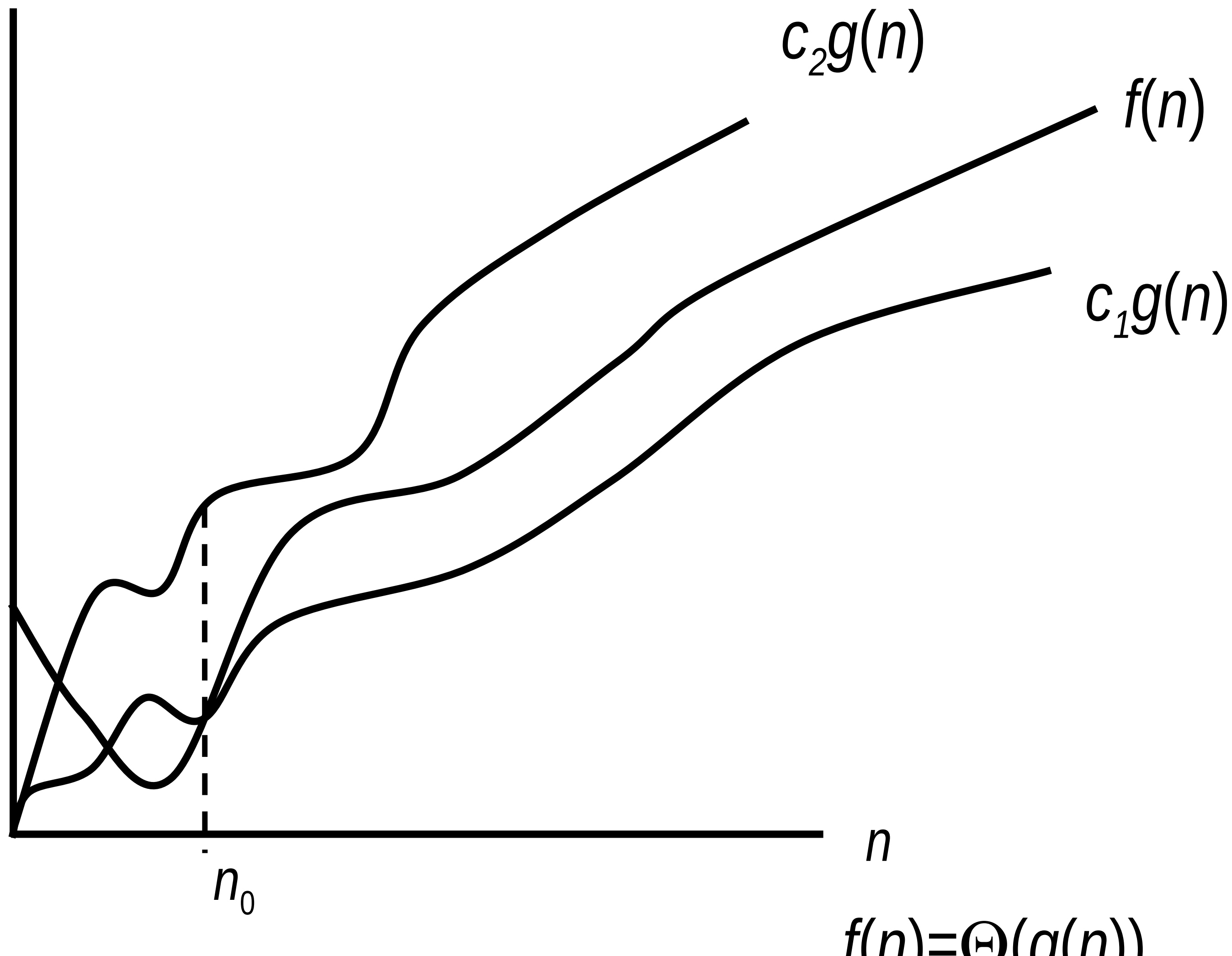
Formalmente, **$\Theta(g(n)) = O\,(g(n)) \cap \Omega(g(n))$**, o que equivale dizer que existem constantes **c, d, n_1 e n_2** tais que:

$$f(n) \le cg(n), \text{ para todo } n \ge n_1$$

e

$$f(n) \ge dg(n), \text{ para todo } n \ge n_2$$

Assintótica Θ



Notação o

Diz-se que f(n) é *o*(g(n)), $n \to \infty$, quando para toda constante positiva ε, existe uma constante n_0 tal que

$f(n) < \varepsilon\, g(n)$, para **$n \geq n_0$**

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Notação ω

Diz-se que f(n) é $\omega(g(n))$, $n \to \infty$, quando para toda constante positiva ε, existe uma constante n_0 tal que

$f(n) > \varepsilon\, g(n)$, para **$n \geq n_0$**

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

Resumo Assintóticas

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \in [0, \infty) \longrightarrow \boxed{f(n) \text{ é } O(g(n))}$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} \in [0, \infty) \longrightarrow \boxed{f(n) = \Omega(g(n))}$$

$$f(n) = \Theta(g(n)) \longrightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$$

Exercícios

Fatorial_iterativo(*n*)

```
{    se (n == 0) então escrever (“1”);
     senão
     {       x=1;
             para i=1 até n fazer
                     x = x * i;
             Escrever( x );       }
 }
```

Resposta

Fatorial_iterativo(*n*)
se (*n* == 0) então escrever (“1”);
senão
x=1;
para i=1 até n fazer
x = x * i;
Escrever(x);

1

1

n

n+1

Pior Caso e Melhor Caso:

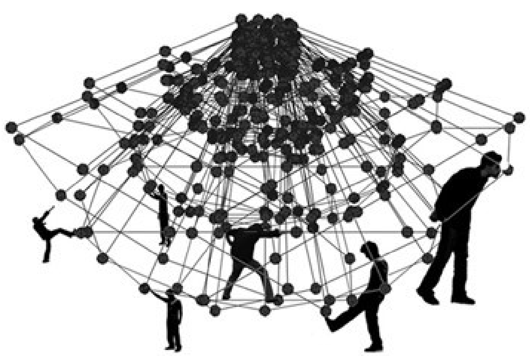
T(*n*) = 1 +1 + (n+1) + n = 2n + 3

Portanto, tanto no pior como no melhor caso, o algoritmo é *O*(*n*).

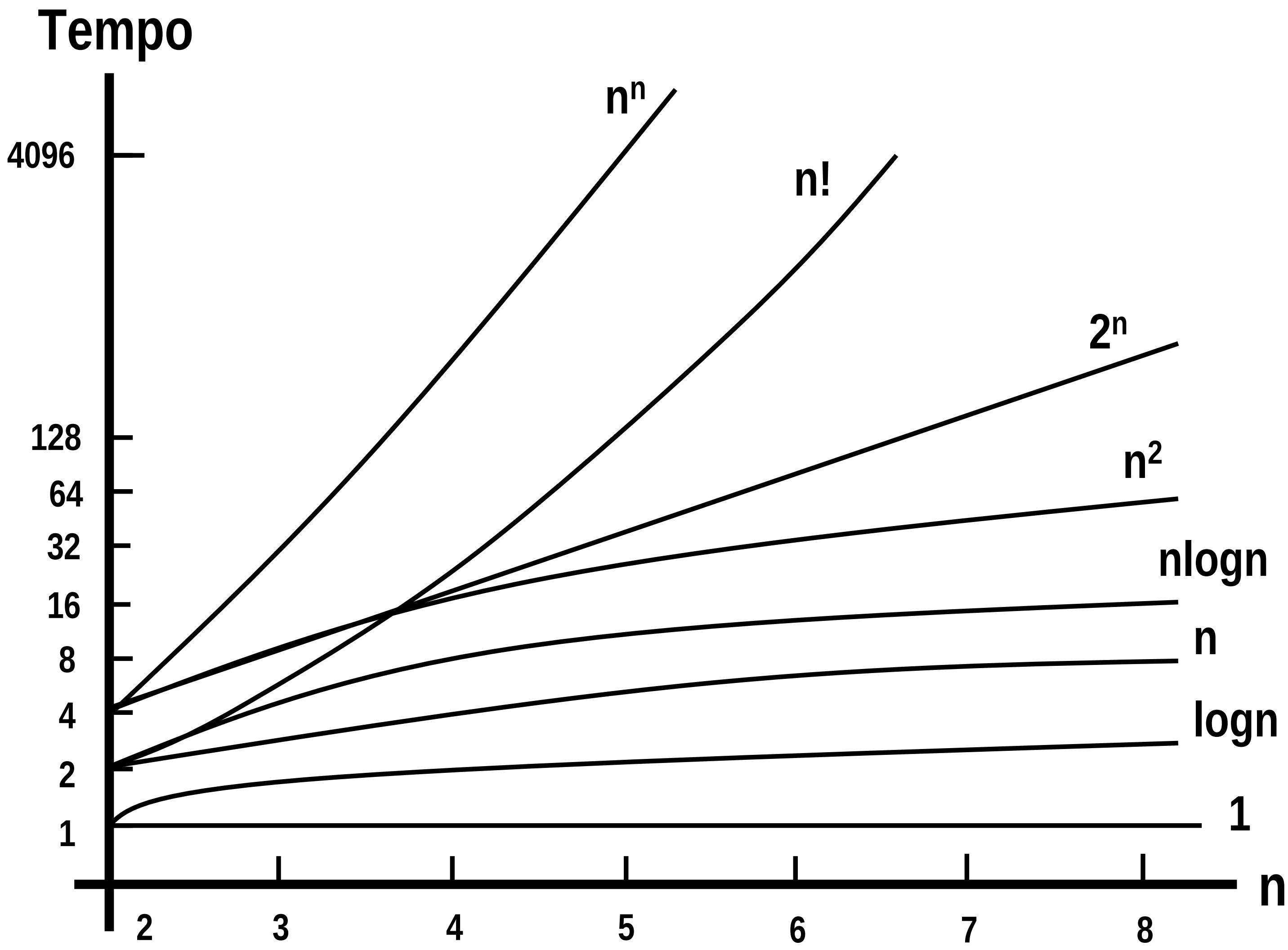
Assim, temos que o algoritmo é Θ(n)

Funções Complexidade

Função	Nome	Função	Nome
C	Constante	n^2	Quadrada
Log n	Logarítmica	n^3	Cúbica
$\log^2 n$	Log quadrada	2^n	Exponencial
n	Linear	n!	Fatorial



Crescimento



Crescimento

Melhor

- $O(1)$ constante
- $O(\log n)$ logarítmica
- $O(n)$ linear
- $O(n \log n)$ log linear
- $O(n^2)$ quadrática
- $O(n^3)$ cúbica
- $O(2^n)$ exponencial

Pior

Exemplo

Sejam 5 algoritmos $A_1 ... A_5$ que resolvem um mesmo problema com um tempo de 1 ms em cada operação

$T_k(n)$é a complexidade do algoritmo *K* na entrada *n*

n	A_1 $T(n)=n$	A_2 $T(n)=n\log n$	A_3 $T(n)=n^2$	A_4 $T(n)=n^3$	A_5 $T(n)=2^n$
16	0,016s	0,064s	0,256s	4s	1m4s
32	0,032s	0,16s	1s	33s	46d
512	0,512s	9s	4m22s	1d13h	10^{137} Séculos

Algumas fórmulas úteis

Série Aritmética	Série Geométrica	Série Harmônica
$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \equiv O(n^2)$	$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1} \equiv O(x^n)$	$\sum_{i=1}^{n} \frac{1}{k} = \ln n + O(1) \equiv O(\ln n)$
$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \equiv O\left(\frac{n^3}{3}\right)$	$\sum_{i=0}^{n} \frac{i}{2^i} = 2$	$\sum_{i=1}^{n} \frac{1}{k} = \ln n + O(1) \equiv O(\ln n)$

$$n! = \sqrt{2\pi\ n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

Aproximação de Stirling

Algumas fórmulas úteis

$$\sum_{i=1}^{n} \log i$$

$$S_n = \log 1 + \log 2 + \log 3 + \ldots + \log n$$

$$S_n = \log(1.2.3\ldots n)$$

$$S_n = \log(n!)$$

$$\log(n!) = \Theta(n \log n) \qquad n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

Algumas fórmulas úteis

$$\sum_{i=1}^{n-1} \frac{1}{i\left(i+1\right)} = \qquad \frac{1}{i\left(i+1\right)} = \frac{1}{i} - \frac{1}{i+1}$$

$$S_n = \sum_{i=1}^{n-1} \left(\frac{1}{i} - \frac{1}{i+1} \right) = \left(1 - \frac{1}{2} \right) + \left(\frac{1}{2} - \frac{1}{3} \right) + \left(\frac{1}{3} - \frac{1}{4} \right) + \ldots + \left(\frac{1}{n-1} - \frac{1}{n} \right)$$

$$S_n = 1 - \frac{1}{n}$$

Exercício

Considere o seguinte método para ordenar *n* números em uma lista *L*. O algoritmo faz uma busca linear e encontra o menor número na posição *i* de *L* e permuta este número com o que está na primeira posição de *L*. Em seguida encontra o 2º menor elemento e troca com o que está na segunda posição de *L*. Esta etapa é realizada para os primeiro *n*-1 elementos de *L*. Escreva o pseudocódigo para este algoritmo. Prove que ele ordena corretamente a lista (são necessários apenas *n*-1 passos no laço). Forneça a complexidade do algoritmo em notação assintótica.

Comp. em tempo de um Problema

Assumindo que o problema possa ser solucionado por algoritmos:

O ***limite superior*** da complexidade em tempo de um problema refere-se ao ***tempo no pior caso do melhor algoritmo*** para resolver este problema.

Comp. em tempo de um Problema

Exemplo: Ordenação por comparação

Algoritmos:

Mergesort – $O(n\log n)$

Quicksort – $O(n^2)$

O limite superior da complexidade da ordenação é $O(n\log n)$.

Comp. em tempo de um Problema

Assumindo que o problema possa ser solucionado por algoritmos:

O ***limite inferior*** da complexidade em tempo de um problema refere-se à melhor complexidade possível.

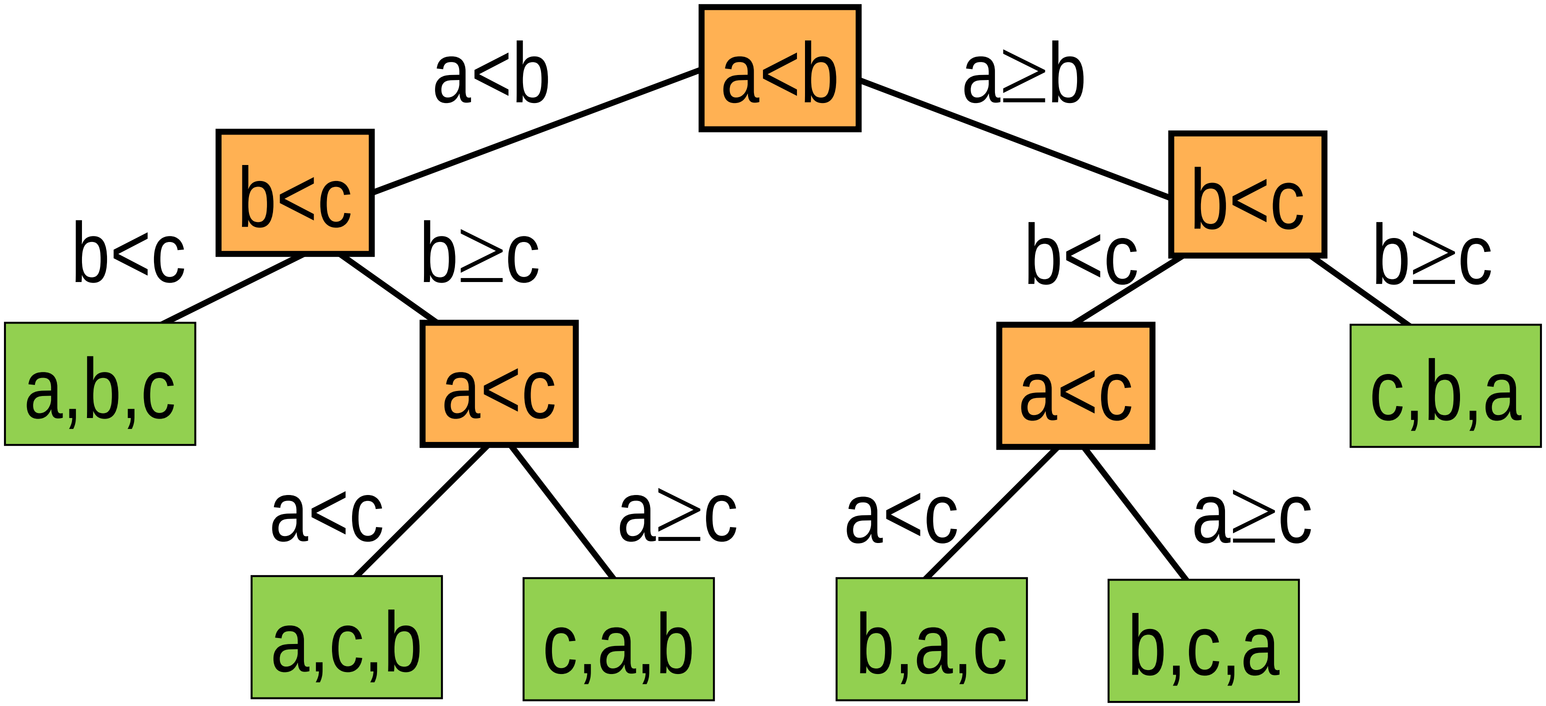
É um resultado teórico que determina não ser possível desenvolver um algoritmo ***cuja complexidade de pior caso*** seja menor que um certo limite estabelecido.

Comp. em tempo de um Problema

Exemplo: Ordenação por comparação

Ordenar três números inteiros a, b e c.

Árvore de Decisão



Comp. em tempo de um Problema

A complexidade é a profundidade de uma folha na árvore de decisão.

A **complexidade de pior caso** é a **maior profundidade** de uma folha **=** altura da árvore.

Comp. em tempo de um Problema

Relação entre o número de folhas e a altura de uma árvore binária completa é

folhas = 2^{altura}

Número de **folhas = *n*!,** onde *n* é o número de elementos a serem ordenados.

Comp. em tempo de um Problema

Como a árvore de decisão não é necessariamente completa temos que:

$$n! \leq 2^{\text{altura}} \Rightarrow \log_2 n! \leq \log_2 2^{\text{altura}} \Rightarrow \text{altura} \geq \lceil \log_2 n! \rceil$$

Aproximação de Stirling

Comp. em tempo de um Problema

Aproximação de Stirling

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\lceil \log_2(n!) \rceil \approx \log_2[\sqrt{2\pi n}] + \log_2[(\frac{n}{e})^n]$$

$$= O\left(n \log_2(\frac{n}{e})\right)$$

$$= O\left(n \log_2(n)\right)$$

Comp. em tempo de um Problema

A complexidade de pior caso é dada pela altura da árvore. Assim:

$\boldsymbol{O(n\log_2 n)}$

Portanto, o limite inferior para o problema é $O(n\log n)$.

Comp. em tempo de um Problema

Existem limites inferiores naturais, como, por exemplo, o tamanho da entrada uma vez que o algoritmo deverá ler sua entrada.

Exemplo: limite inferior para soma de matrizes e para multiplicação de matrizes quadradas de ordem n é $\Omega(n^2)$.

Algoritmo Ótimo

Um limite inferior para um problema *P*, a função *l*, é tal que a complexidade de pior caso de qualquer algoritmo que resolva *P* é $\Omega(l)$.

Se existir algoritmo *A* tal que *A* é $O(l)$, então *A* é dito ***algoritmo ótimo para P***.

Exercício

MergeSort é ótimo para a ordenação por comparações?

QuickSort é ótimo para a ordenação por comparações?