

# Comportamento Assintótico de Funções



Adaptado dos slides da  
Prof<sup>a</sup>. Renata Oliveira

# A notação “O”

- A ordem de crescimento do tempo de execução de um algoritmo nos dá uma caracterização da sua eficiência e nos permite comparar o desempenho relativo com outros algoritmos.
- Quando nos concentramos em tamanhos grandes de entrada, que torna relevante apenas a ordem de crescimento do tempo de execução do algoritmo, estamos estudando a **EFICIÊNCIA ASSINTÓTICA** do mesmo, ou seja, estamos preocupados em como o tempo de execução de um algoritmo cresce com o tamanho da entrada crescendo infinitamente.

Ou ainda: Qual o comportamento do algoritmo para entradas cada vez maiores?

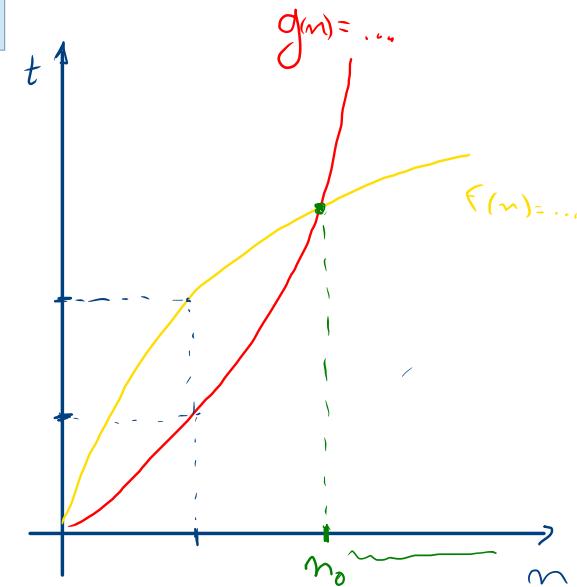
# A notação “O”

- A ordem de magnitude de um algoritmo é dada pela soma das frequências de todas as suas declarações. Ela pode ser extraída diretamente do algoritmo, independente da máquina ou linguagem.
- Existe uma notação matemática adequada para tratar da ordem de magnitude. É chamada de notação O (“O” grande ou Big “O”), e foi sugerida por Knuth.

# Definição

- Uma função  $g(n)$  domina assintoticamente outra função,  $f(n)$ , se existem constantes  $c$  e  $n_0$  (positivas) tais que:

$$\forall n \geq n_0 \quad |f(n)| \leq c |g(n)|$$



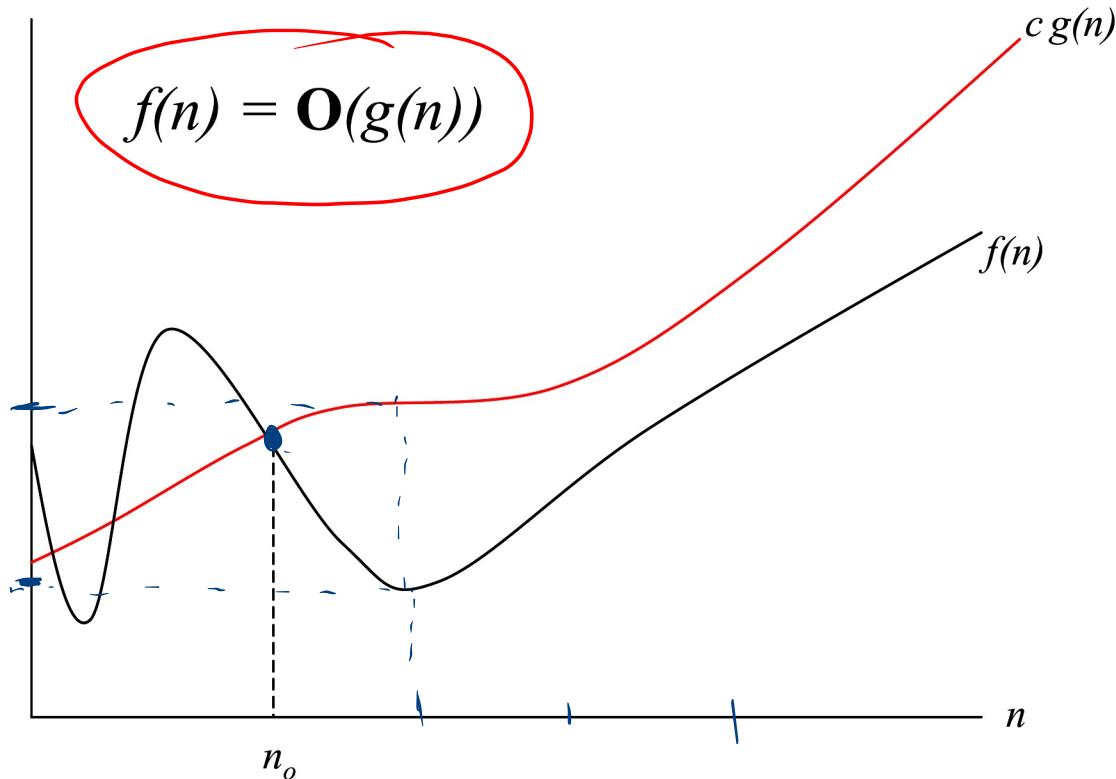
- Neste caso dizemos que  $f(n)$  é  $O(g(n))$  e escrevemos  $f(n) = O(g(n))$

$$|f(n)| \leq c \cdot |g(n)|$$

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

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

# Definição



Dizemos que  $g(n)$  domina assintoticamente  $f(n)$ , ou que  $g(n)$  é o limite assintótico superior para  $f(n)$

# Exemplo

- Suponha que  $f(n) = (3/2)n^2 + (7/2)n - 4$  e que  $g(n) = n^2$ . A tabela abaixo sugere que  $f(n) \leq 2 g(n)$  para  $n \geq 6$  e, portanto,  $f(n) = O(g(n))$

n	f(n)	g(n)
0	4	0
1	1	1
2	9	4
3	20	9
4	34	16
5	51	25
6	71	36 *
7	94	49 ↓
8	120	64

$$F(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

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

$$c = 2$$

$$|F(n)| \leq 2 |g(n)|$$
$$n \geq 6$$

# Exemplo

provar por indução

- Mostre, usando a definição, que  $3n^2+n$  é  $O(n^2)$

$$f(n) = 3n^2 + n$$

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

$$|f(n)| \leq c |g(n)|$$

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

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

$$|f(n)| \leq c |g(n)|$$

$$3n^2 + n \leq c \cdot n^2$$

$$\text{Supondo } n=1 \text{ e } c=4$$

$$\underline{3 \cdot (1)^2 + 1 = 4 \leq 4 = 4 \cdot (1)^2}$$

$$\text{hipótese: } n = k$$

$$3k^2 + k \leq 4k^2$$

$$\text{passo induutivo: } n = k+1$$

$$\underline{3(k+1)^2 + (k+1) \leq 4(k+1)^2}$$

aplicar a  
hip.

$$\begin{aligned} \underline{3(k+1)^2 + (k+1)} &= \underline{3k^2} + 6k + 3 + \underline{k+1} = (3k^2 + k) + (6k + 4) \leq 4k^2 + \underline{6k} + 4 \dots \\ \dots \leq 4k^2 + \underline{8k} + 4 &= \underline{4(k+1)^2} \end{aligned}$$

OK!

## TAREFA

$$n^2 \in O((n+1)^2)$$

$$F(n) \in O(g(n))$$

$$F(n) = n^2$$

$$g(n) = (n+1)^2$$

$$|n^2| \leq c |(n+1)^2|$$

$$2m^2 + 3 \in O(m^2)$$

$$m^2 \in O(2m^2 + 3)$$

# Análise de Algoritmos

- A análise de um algoritmo conta apenas algumas operações “elementares” (por exemplo, comparações, atribuições, multiplicações, acessos ao disco, etc).
- A medida de custo ou medida de complexidade relata o crescimento assintótico da operação considerada.
- Se  $f$  é uma função de complexidade para um algoritmo  $F$ , então  $\underline{O(f)}$  é considerada a complexidade assintótica ou o comportamento assintótico do algoritmo F.
- A relação de dominação assintótica permite comparar funções de complexidade.

# Teorema

- Se  $A(n) = a_m n^m + \dots + a_1 n + a_0$  é um polinômio de grau  $m$ , então  $A(n) = O(n^m)$ .

$$A = a_m n^m + b \cdot n^{m-1} + c \cdot n^{m-2} + \dots + d$$

$$A = O(n^m)$$

$$f(n) = 2n^2 + n + 1$$

$n$	$2n^2$	$n$	1	$2n^2 + n + 1$
1	2	1	1	4
2	8	2	1	11
3	18	3	1	22
4	32	4	1	37
⋮	⋮	⋮	⋮	⋮
10	200	10	1	211

$$O(n^2)$$

# Exemplo:

- A função de complexidade de um algoritmo de ordenação A é dada por

$$f(n) = \frac{n^3 + 2n^2 + n + 1}{5} = \frac{n^3}{5} + \frac{2n^2}{5} + \frac{n}{5} + \frac{1}{5}$$


- Logo, dizemos que este algoritmo tem complexidade  $O(n^3)$ .
- Isto quer dizer que para valores muito grandes de  $n$  poderíamos desconsiderar a outra parte do polinômio ( $2n^2 + n + 1$ ) pois ela não influenciaria tanto no crescimento de  $f(n)$ .

# Importante:

- Um algoritmo é  $O(g(n))$  significa que se ele está rodando em um computador com valores crescentes de  $n$ , os tempos de execução resultantes serão sempre menores que  $|g(n)|$ .
- Quando tentamos determinar a ordem de complexidade de  $f(n)$ , estamos procurando pela menor  $g(n)$  tal que  $f(n) = O(g(n))$ .

$F(n)$  é  $O(g(n))$

$$|F(n)| \leq C * |g(n)|$$

# Propriedades:

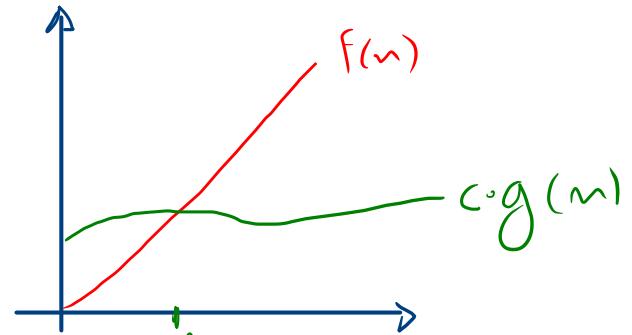
- $f(n) = O(f(n))$   $f(n) = n^2$   
 $n^2 = O(n^2)$
- $c \cdot O(f(n)) = O(f(n))$   $c = \text{constante}$   $2 \cdot O(f(n)) = O(f(n))$
- $O(f(n)) + O(f(n)) = O(f(n))$   $O(n^2 + O(n^2)) = O(n^2)$
- $O(O(f(n))) = O(f(n))$
- $O(f(n) + g(n)) = O(\max(f(n), g(n)))$   $O(n^3 + n^2) = O(n^3)$
- $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$   $O(n^3) \cdot O(n^2) = O(n^3 \cdot n^2)$
- $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n)) = g(n) \cdot O(f(n))$

# Outras notações assintóticas

- Notação Omega ( $\Omega$ ):

Limite assintótico inferior  $f(n) = \Omega(g(n))$  se existirem constantes positivas  $c$  e  $n_0$  tais que:

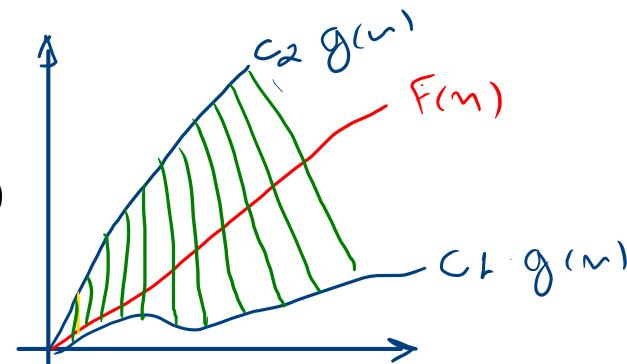
$$|f(n)| \geq c |g(n)|, \forall n \geq n_0$$



- Notação Teta ( $\Theta$ ):

$f(n) = \Theta(g(n))$  se existirem constantes positivas  $c_1$ ,  $c_2$ ,  $n_0$  tais que:

$$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|, \forall n \geq n_0$$



# Propriedades das funções O, θ, Ω

- Considere  $\# = O, \theta, \Omega$ 
  - $f(n) = \#(g(n))$  e  $g(n) = \#(h(n)) \Rightarrow f(n) = \#(h(n))$
  - $f(n) = \#(f(n))$
  - $f(n) = \theta(g(n))$  se e somente se  $f(n) = O(g(n))$  e  $f(n) = \Omega(g(n))$
- Resultados Importantes:
  - $f(n)$  é  $O(g(n)) \Rightarrow 0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$
  - $f(n)$  é  $\Omega(g(n)) \Rightarrow 0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq \infty$
  - $f(n)$  é  $\theta(g(n)) \Rightarrow 0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$

# Exercícios

1. Mostre, por definição, que :

- a)  $\sqrt{n}$  é  $O(n^2)$
- b)  $n^2$  é  $O((n+1)^2)$

2. João desenvolveu um algoritmo cuja função de complexidade é  $f(n) = 2n^2+3$ . Mostre, por definição, que  $f(n)$  é  $O(n^2)$ .

# Exercícios

- José, na análise de quatro algoritmos para criptografia de dados,  $A_1, A_2, A_3$  e  $A_4$  encontrou as seguintes expressões para o número de operações matemáticas realizadas:

$$A_1: f(n) = \log n^{10}$$

$$A_2: f(n) = 2n$$

$$A_3: f(n) = 10n$$

$$A_4: f(n) = 4\sqrt[4]{n}$$

Ajude-o a demonstrar o seguinte

- $A_1$  é  $\Theta(\log n)$
- $A_2$  é  $\Omega(n^m)$  para  $m > 1$
- $A_3$  é  $O(n \log n)$
- $A_4$  é  $\Omega(\ln n)$

# Classes de comportamento assintótico

- $f = O(1)$  complexidade constante
  - O uso do algoritmo independe de  $n$ .
  - $O(c)$  para  $c \in R$  é  $O(1)$
  - Comandos de atribuição, leitura, escrita.
- $f = O(\log(n))$  complexidade logarítmica
  - O custo de aplicar o algoritmo a problemas com  $n$  suficientemente grande pode ser limitado por uma função do tipo  $K \cdot \log n$   $K \in R$ .
- $f = O(n)$  complexidade linear
  - O custo do algoritmo cresce linearmente com o tamanho da entrada.

# Classes de comportamento assintótico

- $f = O(n \log n)$
- $f = O(n^2)$  complexidade quadrática
- $f = O(n^3)$  complexidade cúbica
- $f = O(cn)^c$ , constante, complexidade exponencial
- $f = O(n!)$  complexidade fatorial

$O(1) < O(\log \log n) < O(\log n) < O(n^{1/10}) < O(\sqrt{n}) <$   
 $O(n) < \dots < O(n \log n) < O(n^2) < O(n^5) < O(2^n) <$   
 $O(n!)$

# Análise de algoritmos: Alguns princípios

- 1.O tempo de execução de um comando de atribuição de leitura e ou de escrita pode ser considerado  $O(1)$ .
- 2.O tempo de execução de uma SEQUÊNCIA de comandos é determinado pelo maior tempo de execução de qualquer comando da sequência
- 3.O tempo de execução de um comando de DECISÃO é composto pelo tempo dos comandos executados dentro da condição, mais o tempo para avaliar a condição.
- 4.O tempo para executar um laço é a soma do tempo de execução do corpo do laço mais o tempo de avaliar a condição de parada, multiplicado pelo número de iterações do laço.

# Análise de algoritmos: Alguns princípios

## 5.PROGRAMAS COM PROCEDIMENTOS NÃO RECURSIVOS

- Cada procedimento é tratado separadamente, iniciando pelos procedimentos que não chamam outros procedimentos.
- A seguir, computar o tempo dos procedimentos que chamam os procedimentos que não chamam outros procedimentos, utilizando os tempos já avaliados.
- Repetir o processo até chegar no procedimento principal.

# Exemplo 1: Encontrar o maior e o menor elemento de um vetor

```
void maxmin(A:vetor;)
{
    int i;
    max = A[0];
    min = A[0];
    for (i = 1; i < n;i++)
    {
        if (A[i] > max) max = A[i];
        if (A[i] < min) min = A[i];
    }
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9

# Exemplo 2: Ordenar os elementos de um vetor

```
void Ordena (vetor A)
1  {
2      int i, j, min, x;
3      for (i = 1; i <= n; i++)
4      {
5          min = i;
6          for ( j = (i+1); j <= n; j++ )
7              if (A[j-1] < A[min-1] )
8                  min := j;
9          x = A[min-1];
10         A[min-1] = A[i-1];
11         A[i-1] = x;
12     }
13 }
```

# Análise de algoritmos: Alguns princípios

## 6.PROGRAMAS COM PROCEDIMENTOS RECURSIVOS

- Para cada procedimento recursivo, associar uma função de custo  $f(n)$  desconhecida ( $n$  mede o tamanho dos argumentos)
- Obter uma relação de recorrência.
- Calcular a complexidade da relação de recorrência.

# Recursão

- Uma definição na qual o item que está sendo definido aparece como parte da definição é chamada definição indutiva ou definição recursiva.
- Definições recursivas são compostas de duas partes:
  - Uma base, onde alguns casos simples do item que está sendo definido são dados explicitamente.
  - Um passo indutivo ou recursivo, onde outros casos do item que está sendo definido são dados em termos dos casos anteriores.

# Sequências Recursivas

- Uma sequência S é uma lista de objetos que são enumerados segundo alguma ordem; existe o 1º objeto, um segundo e assim por diante.
- $S(k)$  denota o K-ésimo objeto da sequência
- Uma sequência é definida recursivamente, explicitando-se seu primeiro valor (ou seus primeiros valores) e, a partir daí, definindo-se outros valores na sequência em termos dos valores iniciais.

# Exemplo

- Seja a sequência S definida recursivamente por:

$$\begin{cases} S(1) = 2 \\ S(n) = 2S(n-1) \text{ para } n \geq 2 \end{cases}$$

- Qual o valor de  $S(1)$ ? E de  $S(2)$ ? E de  $S(3)$ ? E de  $S(5)$ ?

# Relações de Recorrência

- As relações de recorrência são utilizadas para descrever algoritmos recursivos.
- Uma recorrência é uma equação ou desigualdade que descreve uma função em termos de entradas menores.
- Existem três métodos principais para resolver relações de recorrência.
  - Método da Substituição
  - Método da Iteração
  - Método Master ou Mestre

# Método da Substituição

- “Chute Inicial”
- Indução matemática para provar que o chute estava correto.

$$\left\{ \begin{array}{l} T(n) = 3T(n-1) + 1 \\ T(1) = 1 \end{array} \right.$$

# Método da Iteração

- A ideia é expandir (iterar) a recorrência e expressá-la como uma soma de termos dependentes apenas de  $n$  e das condições iniciais. Técnicas para avaliar somas podem então ser usadas para encontrar a solução.

Exemplos:

$$\begin{cases} T(n) = T(n-1) + 1 \\ T(1) = 1 \end{cases}$$

$$\begin{cases} T(n) = T(n/2)+1 \\ T(1) = 0 \end{cases}$$

$$\begin{cases} T(n) = T(n-1) + n \\ T(1) = 1 \end{cases}$$

# Método Master (ou Mestre)

- O método Máster provê uma “receita” para resolver recorrências do tipo  $T(n) = a T(n/b) + f(n)$  com  $a \geq 1$ ,  $b > 1$  constantes e  $f(n)$  função assintoticamente positiva.
- A ideia da recorrência é a divisão de um problema de tamanho  $n$  em  $a$  subproblemas, cada um de tamanho  $n/b$ .
- Os  $a$  subproblemas são resolvidos de forma recursiva, cada um com tempo  $T(n/b)$ .
- O custo de dividir o problema e combinar os resultados dos subproblemas é descrito pela função  $f(n)$ .

# O Teorema Máster:

- Seja  $T(n) = a T(n/b) + f(n)$   $a \geq 1$ ,  $b > 1$ ,  $f(n) \geq 0$   
Então  $T(n)$  pode ser limitada assintoticamente como segue:
  - i. Se  $f(n) = O(n^{(\log_b a) - \varepsilon})$  para alguma constante  $\varepsilon > 0$  então  $T(n) = \Theta(n^{\log_b a})$
  - ii. Se  $f(n) = \Theta(n^{\log_b a})$  então  $T(n) = \Theta(n^{\log_b a} \cdot \log n)$
  - iii. Se  $f(n) = \Omega(n^{(\log_b a) + \varepsilon})$  para alguma constante  $\varepsilon > 0$  e  $a.f(n/b) \leq c.f(n)$  para alguma constante  $c < 1$  então  $T(n) = \Theta(f(n))$

Exemplos:

$$\left\{ \begin{array}{l} T(n) = 9T(n/3) + n \\ T(1) = 1 \end{array} \right.$$

$$\left\{ \begin{array}{l} T(n) = T(2n/3) + 1 \\ T(1) = 1 \end{array} \right.$$