

# Complexidade de Algoritmos (parte 1)

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados 2 (AE43CP)  
Engenharia de Computação  
Departamento Acadêmico de Informática (Dainf)  
Universidade Tecnológica Federal do Paraná (UTFPR)  
Campus Pato Branco

- Análise Assintótica
  - Conceitos matemáticos
  - Notações
- Taxas de Crescimento

- Por que estudar a complexidade de algoritmos?
  - Qual a diferença entre programa e algoritmo?

- Programa vs. Algoritmo

<b>Programa</b>	<b>Algoritmo</b>
Linguagem concreta (C, Java, Python)	Linguagem abstrata (pseudo-código)
Dependente de sistema operacional	Independente de sistema operacional
Dependente de hardware	Independente de hardware
Avaliação em tempo real (empírica)	Avaliação por estimativa (assintótica)

- O algoritmo deve resolver o problema corretamente e rapidamente (nem sempre isso é possível)
- Algoritmos eficientes
  - Problemas determinísticos (P) vs. não-determinístico (NP)
- Existem problemas para os quais não são conhecidos algoritmos eficientes para resolvê-los
  - Chamados de NP-completos

- Afinal, como mensurar a eficiência de um algoritmo?

- Afinal, como mensurar a eficiência de um algoritmo?
  - Na disciplina "Algoritmos e Estrutura de Dados 1" tivemos uma introdução à análise de complexidade de algoritmos, na qual vimos é importante determinar os recursos necessários para a execução de cada algoritmo:
    - Tempo
    - Espaço

- Lembram do conto abaixo?

"Desenvolvi um novo algoritmo chamado TripleX que leva 14,2 segundos para processar 1.000 números, enquanto o método SimpleX leva 42,1 segundos."
- Você trocaria o SimpleX que roda em sua empresa pelo TripleX?



- A afirmação tem que ser examinada, pois há diversos fatores envolvidos, como características da máquina, linguagem de programação, etc
- É desejável a comparação de algoritmos e não programas
- Análise/complexidade de algoritmos: comparações entre algoritmos de forma independente de
  - Hardware
  - Sistema operacional
  - Linguagem de programação
- A análise/complexidade de algoritmos também tem o objetivo de determinar se o algoritmo analisado é ótimo

- Sabe-se que:
  - Processar 100.000 números leva mais tempo do que 10.000 números
  - Cadastrar 20 itens em um sistema de vendas leva mais tempo do que cadastrar 10
  - Etc
- Na introdução à análise de complexidade em "Algoritmos e Estrutura de Dados I" vimos que podemos estimar a eficiência de um algoritmo em função do tamanho do problema (número de elementos processados):
  - Geralmente, é assumido que  $n$  é o tamanho do problema
  - É calculado o número de operações realizadas sobre os  $n$  elementos (e.g. comparações, somas, subtrações, incrementos, etc)

- Exemplo: TripleX vs. SimpleX
  - TripleX: para uma entrada de tamanho  $n$ , o algoritmo realiza  $n^2 + n$  operações:
    - $f(n) = n^2 + n$
  - SimpleX: para uma entrada de tamanho  $n$ , o algoritmo realiza  $1.000n$  operações:
    - $g(n) = 1.000n$

Tamanho da Entrada	1	10	100	1.000	10.000
$f(n) = n^2 + n$	2	110	10.100	<b>1.001.000</b>	<b>100.010.000</b>
$g(n) = 1.000n$	<b>1.000</b>	<b>10.000</b>	<b>100.000</b>	1.000.000	10.000.000

- partir de  $n = 1.000$ ,  $f(n)$  mantém-se maior e cada vez mais distante de  $g(n)$ 
  - Diz-se que  $f(n)$  cresce mais rápido do que  $g(n)$

## **Análise Assintótica**

- Devemos nos preocupar com a eficiência de algoritmos quando o tamanho de  $n$  for grande
- A análise assintótica de um algoritmo descreve a sua eficiência relativa quando  $n$  torna-se grande
- Comparação de algoritmos: o algoritmo com menor taxa de crescimento rodará mais rápido quando o tamanho do problema for grande
- Também se pode aplicar os conceitos de análise assintótica para a quantidade de memória usada por um algoritmo

- Expoentes:

- $x^a x^b = x^{a+b}$

- $x^a / x^b = x^{a-b}$

- $(x^a)^b = x^{ab}$

- $x^n + x^n = 2x^n$

- $2^n + 2^n = 2^{n+1}$

- Logaritmos (usaremos a base 2, a menos que seja dito o contrário)
  - $x^a = b \Rightarrow \log_x b = a$
  - $\log_a b = \log_c b / \log_c a$ , se  $c > 0$
  - $\log ab = \log a + \log b$
  - $\log a/b = \log a - \log b$
  - $\log a^b = b \log a$

- Séries

- $\sum_{i=1}^n 2^i = 2^{n+1} - 1$

- $\sum_{i=1}^n a^i = \frac{a^{n+1} - 1}{a - 1}$

- $\sum_{i=1}^n i = \frac{n(n+1)}{2}$

- $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$

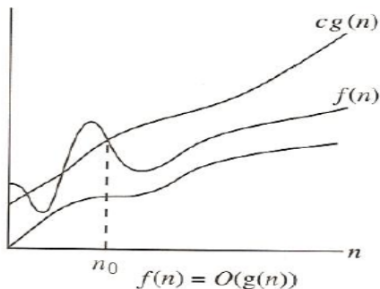


- Notações de Knuth:
  - *Big-oh*:  $O(n)$
  - Ômega:  $\Omega(n)$
  - Teta:  $\Theta(n)$

# Análise Assintótica

Notações: big-oh

- Limite assintótico superior
- Dada duas funções,  $f(n)$  e  $g(n)$ 
  - Uma função  $f(n)$  é da ordem de (*big-oh*)  $g(n)$  ou função  $f(n)$  é  $O(g(n))$  se existirem duas constantes positivas  $c$  e  $n_0$  tais que  $f(n) \leq cg(n)$ , para todo  $n \geq n_0$

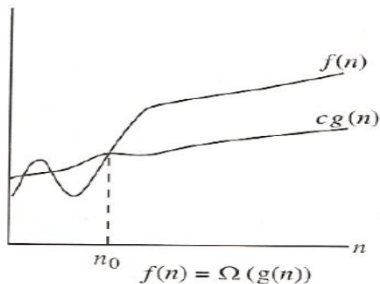


- Exemplos:
  - Seja  $f(n) = (n + 1)^2$ . Logo,  $f(n)$  é  $O(n^2)$ , quando  $n_0 = 1$  e  $c = 4$ , pois  $(n + 1)^2 \leq 4n^2$  para  $n \geq n_0$
  - Seja  $f(n) = 2n^3 + n^2 + 3n + 1$ . Logo, para provar que  $f(n)$  é  $O(n^3)$ , basta provar que  $2n^3 + n^2 + 3n + 1 \leq 6n^3$  para  $n \geq n_0$
- Notação  $O$  é a mais utilizada para a análise de complexidade de algoritmos
- Ao dizer que  $f(n) = O(g(n))$ , tem-se que  $g(n)$  é o limite superior de  $f(n)$
- Exemplo: algoritmo de ordenação *quicksort*
  - O pior cenário é quando o pivô é escolhido de forma que divisão do sub-arranjo fique desbalanceada:  $O(n^2)$

# Análise Assintótica

Notações: ômega

- Limite assintótico inferior
- Dada duas funções,  $f(n)$  e  $g(n)$ 
  - Uma função  $f(n)$  é da ordem de ômega  $g(n)$  ou função  $f(n)$  é  $\Omega(g(n))$  se existirem duas constantes positivas  $c$  e  $n_0$  tais que  $f(n) \geq cg(n)$ , para todo  $n \geq n_0$



# Análise Assintótica

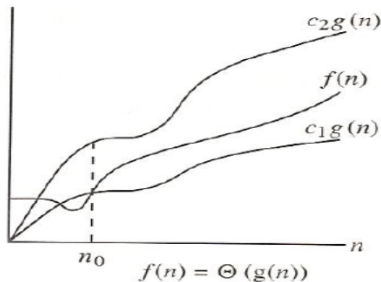
Notações: ômega

- Exemplo: para mostrar que  $f(n) = 3n^3 + 2n^2$  é  $\Omega(n^3)$ , basta fazer  $c = 1$  e  $n_0 = 0$ , e então  $3n^3 + 2n^2 \geq n^3$  para todo  $n \geq n_0$
- Ao dizer que  $f(n) = \Omega(g(n))$ , tem-se que  $g(n)$  é o limite inferior de  $f(n)$
- Exemplo: a quantidade mínima ("o melhor que pode fazer") de operações que o *quicksort* realiza é na ordem de  $n \log n$
- Essa notação é pouca utilizada

# Análise Assintótica

Notações: teta

- Limite assintótico firme
- Dada duas funções,  $f(n)$  e  $g(n)$ 
  - Uma função  $f(n)$  é da ordem de teta  $g(n)$  ou função  $f(n)$  é  $\theta(g(n))$  se existirem constantes positivas  $c_1$ ,  $c_2$  e  $n_0$  tais que  $c_1g(n) \leq f(n) \leq c_2g(n)$  para todo  $n \geq n_0$



# Análise Assintótica

Notações: teta

- Exemplo: para mostrar que  $f(n) = \frac{n^3}{3}$  é  $\Theta(n^3)$ , podemos fazer  $c_1 = \frac{1}{6}$  e  $c_2 = \frac{2}{3}$  para  $n \geq 0$
- Quando  $f(n) = \Theta(g(n))$ , isso significa que  $f(n)$  é  $O(g(n))$  e  $\Omega(g(n))$
- Exemplo: algoritmo de ordenação *heapsort*, onde o custo mínimo e máximo é  $n \log n$ , ou seja,  $\Theta(n \log n)$

- O uso das notações permite comparar a taxa de crescimento das funções correspondentes aos algoritmos
- Exemplo: Para 2 algoritmos quaisquer, considere as funções de eficiência correspondentes  $f(n) = 1.000n$  e  $g(n) = n^2$ 
  - A primeira cresce mais rapidamente para valores pequenos de  $n$
  - A segunda cresce mais rapidamente e finalmente será uma função maior, sendo que o ponto de mudança é  $n = 1.000$
  - Em uma comparação entre ambas funções, podemos dizer que  $f(n) = O(g(n))$  e  $g(n) = \Omega(f(n))$



- Na análise de algoritmos, algumas vezes, apenas uma função referente à quantidade de instruções é obtida
  - Exemplo: para uma função que executa  $2n^3 + 4n^2 + 3n + 5$  instruções em seu "pior cenário", dizemos que a sua complexidade é de  $O(n^3)$

Não faz sentido comparar pontos isolados das funções, pois é  $n^3$  que representa a gradeza da função

- O cálculo de instruções será abordado na próxima aula

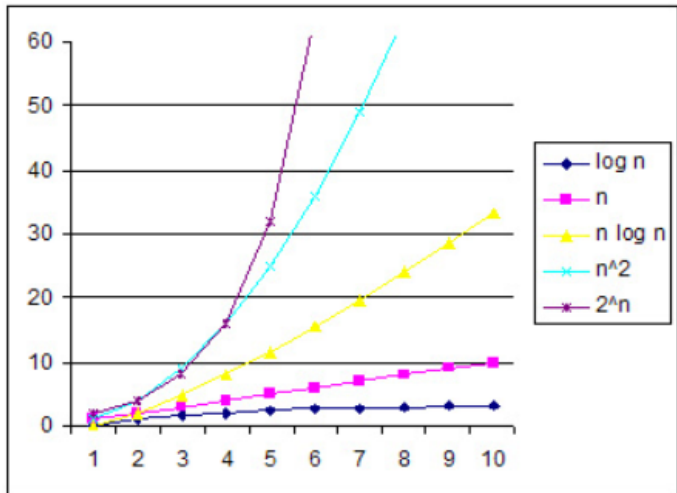
## Taxas de Crescimento

- Funções e taxas de crescimento mais comuns:

$c$	constante
$\log n$	logarítmica
$n$	linear
$n \log n$	linear
$n^2$	quadrática
$n^3$	cúbica
$2^n$	exponencial
$a^n$	exponencial





# Taxas de Crescimento

- Crescimentos de algumas funções



- Se a função  $T(x)$  é um polinômio de grau  $n$ , então  $T(x) = \Theta(x^n)$
- Comparação de funções para  $T_1(n) = O(f(n))$  e  $T_2(n) = O(g(n))$ 
  - $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$
  - $T_1(n) * T_2(n) = O(f(n) * g(n))$
  - $f(n) = \Theta(g(n))$  se e somente se  $g(n) = \Theta(f(n))$
  - $f(n) = O(g(n))$  se e somente se  $g(n) = \Omega(f(n))$
- Não se diz que  $T(n) = O(10n^3)$  ou que  $T(n) = O(10n^3 + 5n^2 + n)$ 
  - Diz-se apenas  $T(n) = O(n^3)$

- Apesar de às vezes ser importante, não é comum incluir constantes ou termos de menor ordem em taxas de crescimento:
  - Queremos medir a taxa de crescimento da função, o que torna os "termos menores" irrelevantes
  - As constantes também dependem do tempo exato de cada operação
  - como ignoramos os custos reais das operações, ignoramos também as constantes

-  Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Clifford, S.  
*Algoritmos: teoria e prática.*  
Elsevier, 2012.
-  Horowitz, E., Sahni, S. Rajasekaran, S.  
*Computer Algorithms.*  
Computer Science Press, 1998.
-  Szwarcfiter, J.; Markenzon, L.  
*Estruturas de Dados e Seus Algoritmos.*  
LTC, 2010.
-  Ziviani, M.  
*Projetos de Algoritmos: com implementações em Pascal e C.*  
Thomson, 2004.