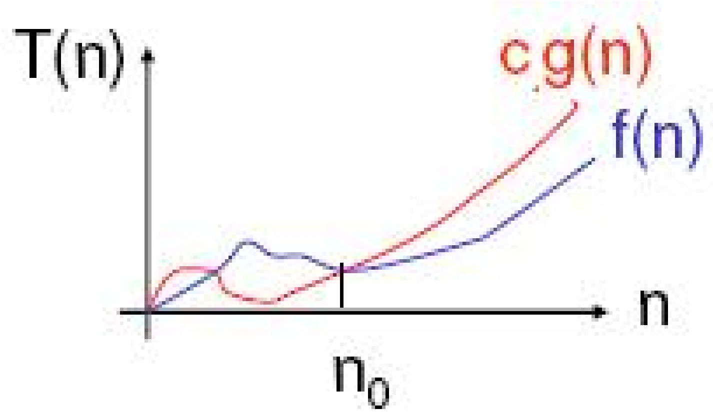
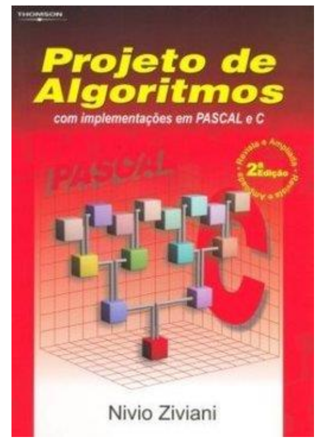
Unidade 4 - Crescimento Assintótico de Funções

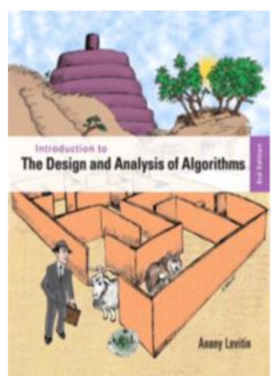


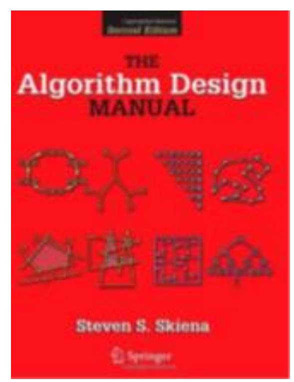
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

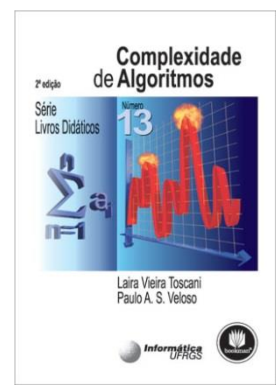
Bibliografia

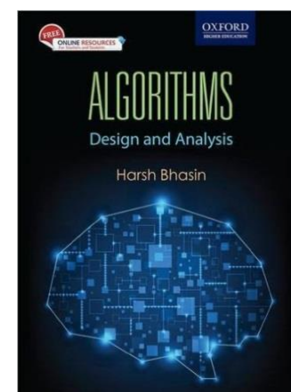
- Algoritmos – Teoria e Prática – Cormen – Segunda Edição – Editora Campus, 2002
- Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015
- Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012
- The Algorithm Design Manual – Steven S. Skiena, Springer, 2008
- Complexidade de Algoritmos – Série Livros Didáticos – UFRGS
- Algorithms – Design and Analysis – Harsh Bhasin – Oxford University Press – 2015
- Projeto de Algoritmos – Nivio Ziviani – Pioneira Informática - 1993











Introdução

- Para a execução de uma tarefa computacional, talvez o mais importante seja projetar um **algoritmo correto**.
- Um algoritmo é dito correto se ele atende à especificação da **tarefa requerida**;
- Entretanto, a despeito de ser correto, um algoritmo pode ter execução **impraticável**.
- Por exemplo, a pesquisa linear em um array é correta, mas **impraticável** se o array tiver $\mathbf{10^{10}}$ elementos.

Exemplo

- Pesquisa Linear em um array com $\mathbf{10^{10}}$ elementos;
- Tempo para se processar um elemento do array: $\mathbf{10^{-6}}$ segundos;
- No pior caso, serão necessários **10.000** segundos ou cerca de 3 horas para se buscar o elemento arbitrário no array.

Assim, algoritmos devem ser corretos mas também eficientes . . .

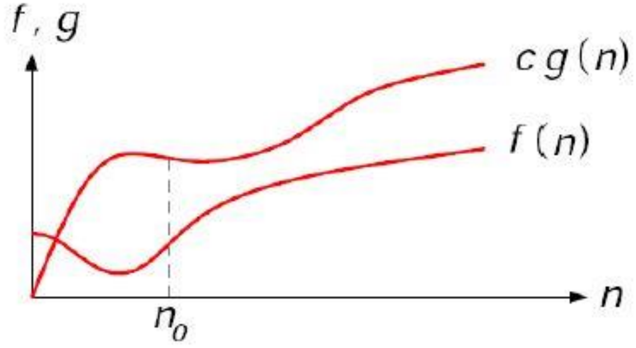
Introdução

- É difícil determinar-se - de forma exata - o tempo de execução de um algoritmo!

- Em **geral**, cada passo em um pseudocódigo e cada **statement** em uma implementação **HLL** corresponde a um pequeno número de **operações primitivas** que não depende do tamanho da entrada;

- Assim, pode-se executar uma **análise simplificada** que estima o número de operações primitivas, por meio da contagem dessas operações;

- Felizmente, há uma notação que permite caracterizar os principais fatores que afetam o desempenho de um algoritmo, sem levar em conta os detalhes dessas operações primitivas.



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

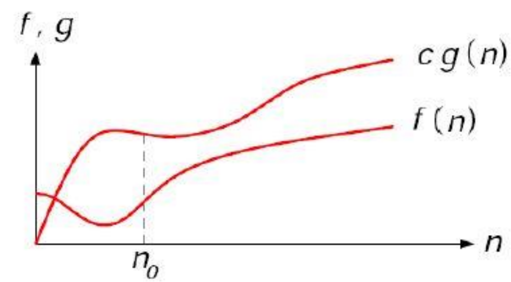
Análise Assintótica

- Ao se deparar com uma Função de Complexidade definida por **F(n) = n+10** ou **F(n) n^2 + 1**, geralmente pensa-se em valores não muito grandes de **n**, ou ainda valores próximos de zero;

- Na Análise de Algoritmos, **por outro lado**, atua-se de forma exatamente ao contrário;

- **Ignora-se valores pequenos** de **n** e **foca-se** em valores **grandes** (suficientemente grandes) de **n**;

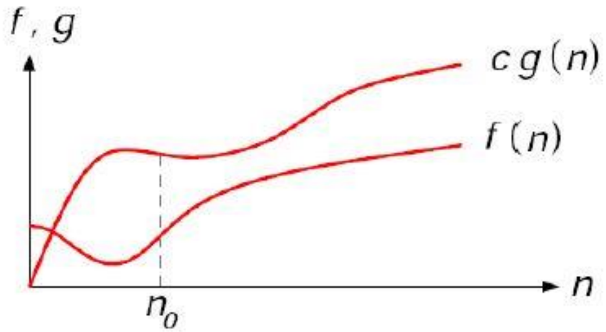
- Esse tipo de Análise é denominada **Análise Assintótica**.



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

Análise Assintótica - Exemplo

- Consideremos o número de operações de dois Algoritmos que resolvem um mesmo problema, em função de **n**, onde **n** corresponde ao tamanho da entrada.
- **Algoritmo A**: **F1(n) = $2n^2$ + 50** operações
- **Algoritmo B**: **F2(n) = 500 n + 4000** operações
- Dependendo do valor de n, o **Algoritmo A** pode requerer mais ou menos operações que o **Algoritmo B**.

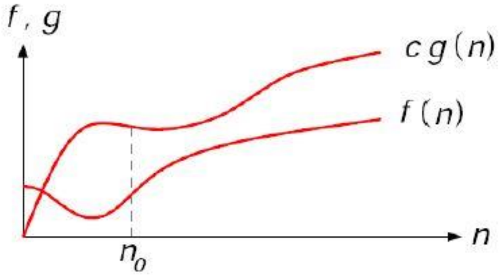


$$f(n) = O(g(n))$$

Análise Assintótica - Exemplo

- **Algoritmo A**: **$F_1(n) = 2n^2 + 50$** operações
  - ✓ Para **n=10** => Serão necessárias $10.10^2 + 50$ = **1.050 operações**
  - ✓ Para **n= 100** => Serão necessárias 10.100^2 = 100.000 + 500 = **100.500 operações**

- **Algoritmo B**: **$F_2(n) = 500\ n + 4000$** operações
  - ✓ Para **n=10** => Serão necessárias 500.10 + 4.000 = **9.000 operações**
  - ✓ Para **n= 100** => Serão necessárias 500.100 + 4000 = 50.000 + 500 = **50.500 operações**



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

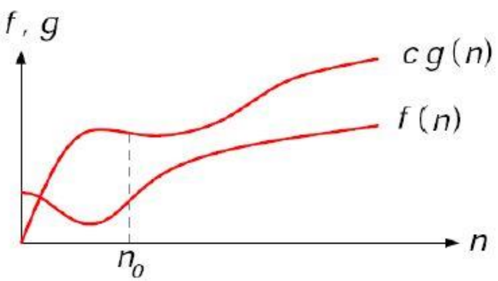
Análise Assintótica

- Assim, o importante é observar-se que **$F_1(n)$** tem crescimento proporcional a **n^2** (**<u>Quadrático</u>**);

- Ao passo que **$F_2(n)$** tem crescimento proporcional a **n** (**<u>Linear</u>**);

- Um crescimento **quadrático** é **<u>PIOR</u>** que um crescimento **linear**;

- Portanto, na comparação das Funções **F_1** e **F_2**, deve-se preferir o **Algoritmo B**.

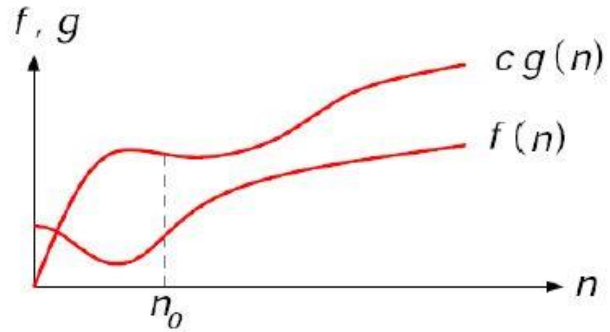


$$f(n) = O(g(n))$$

Análise Assintótica

- Considerando que é **muito difícil** levantar-se a quantidade **exata** de operações executadas por um algoritmo, em **Análise de Algoritmos** concentra-se, portanto, no **comportamento assintótico** das funções de complexidade, ou seja, deve-se observar a **taxa de crescimento** da função quando **n** é **suficientemente grande**;

- Em geral, os **termos inferiores** e as **constantes multiplicativas** pouco contribuem na análise e podem, dessa forma, serem **descartadas**.



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

Análise Assintótica

- ✓ Para valores suficientemente grandes de n, as funções:

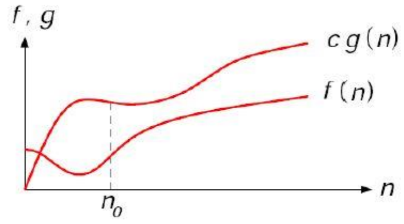
  n^2 , $7/2\ n^2$, $555555n^2$, $n^2/8888$, $7n^2 + 300n + 4$

- ✓ Crescem todas com a mesma velocidade e, portanto, do ponto de vista **assintótico**, são "**equivalentes**";

- ✓ Na Área de **Análise de Algoritmos**, as funções de Complexidade são classificadas em "ordens";

- ✓ Todas as funções de uma mesma ordem são "**equivalentes**";

- ✓ As cinco funções acima pertencem, portanto, à mesma ordem (**quadráticas**);

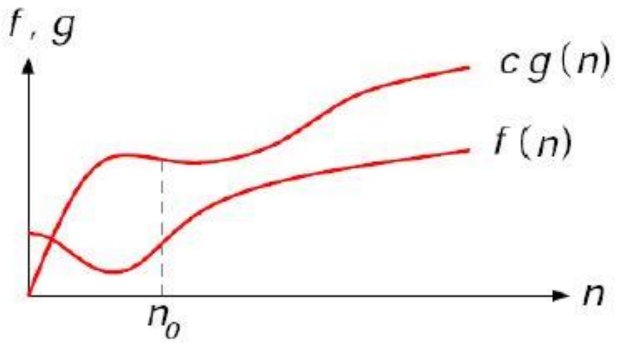


$$f(n) = O(g(n))$$

Funções Assintoticamente Não Negativas

- Na **Análise de Algoritmos**, restringem-se o estudo para funções assintoticamente não-negativas, ou seja, uma função **f** tal que **f(n) >= 0**, para todo **n** suficientemente grande;

- Mais explicitamente, **f** é assintoticamente não-negativa se existe um n_0 tal que **f(n) >= 0**, para todo $n > n_0$.



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

Ordem de grandeza de execução

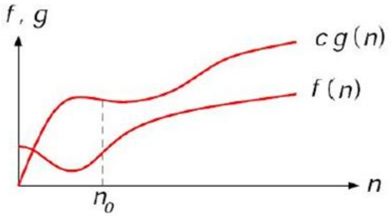
- Por exemplo, o tempo exato de execução de um algoritmo pode ser dado pela função polinomial **$f(n) = 3n^2 + 2n + 3$.**

- Neste caso, o tempo aproximado de execução será uma função de **n^2**, ou seja **$f(n^2)$.** (mais alta potência de **n**)

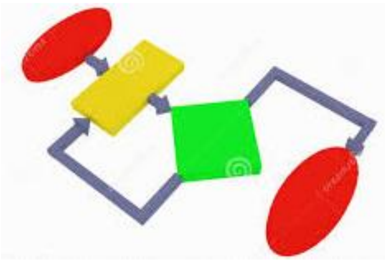
- Dessa forma, pode-se desprezar o coeficiente de **n^2**, bem como os outros termos da **função polinomial** que define a **complexidade** do algoritmo;

- Assim, para efeito de análise de algoritmos, utiliza-se uma notação que seja capaz de exprimir a ordem de grandeza do tempo de execução.

- Essa notação é **assintótica**, ou seja, representa uma linha que se aproxima da função de complexidade do algoritmo.



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

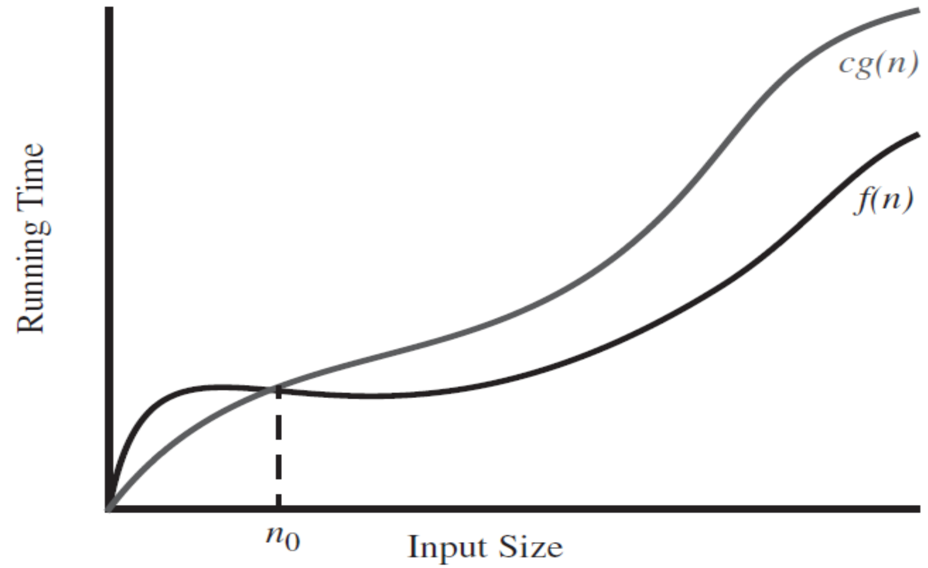


A notação Big-Oh

- Seja f(n) e g(n) funções que mapeiam inteiros não negativos para números reais;

- Diz-se que f(n) é O(g(n)) se existir uma constante real $c > 0$ e uma constante inteira $n_o \geq 1$ tal que $f(n) \leq cg(n)$ para todo inteiro $n \geq n_0$;

- Essa definição é frequentemente dita **“f(n) é big-Oh de g(n)”** ou **“f(n) é ordem g(n)”**.



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.