

Introdução à Ciência da Computação II

Análise de Algoritmos: Parte II

Prof. Ricardo J. G. B. Campello

Este material consiste de adaptações e extensões de slides disponíveis em <http://ww3.datastructures.net> (Goodrich & Tamassia).

Sumário

◆ Análise Assintótica de Algoritmos

- Breve Revisão
- Notação Assintótica
- BIG-O
- Parentes do BIG-O
- Exemplos e Exercícios



Análise Teórica (Revisão)

- ◆ Baseada em uma descrição de alto nível do algoritmo, ao invés de uma dada implementação
- ◆ Caracteriza o tempo de execução como uma função do tamanho da entrada, n
- ◆ Leva em consideração todas as possíveis entradas
- ◆ Permite avaliar a rapidez de um algoritmo de forma independente do ambiente de hardware e/ou software
 - Faz isso estimando a **taxa de crescimento** do número de **operações primitivas** em função do **tamanho da entrada**

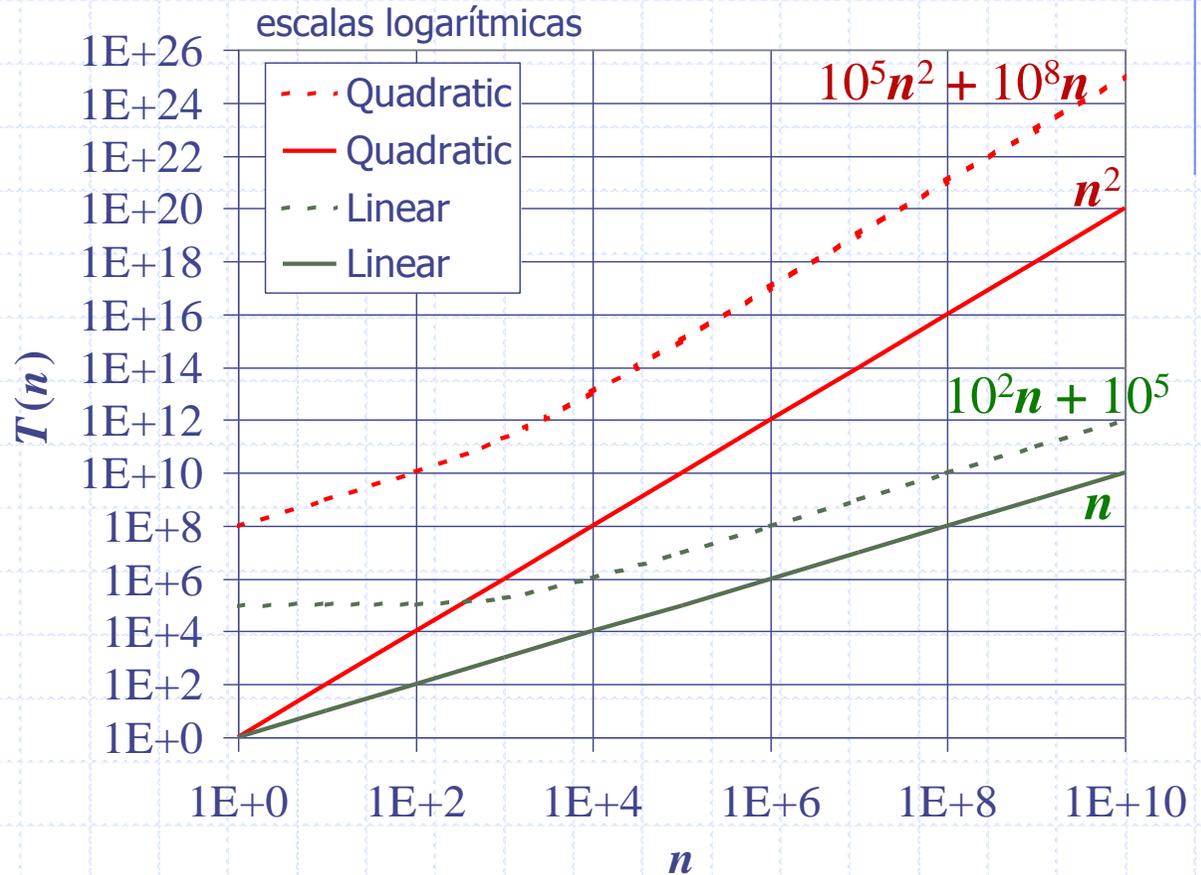
Taxa de Crescimento (Revisão)

◆ A taxa de crescimento não é afetada por:

- fatores constantes ou
- termos de menor ordem

◆ Exemplos:

- $10^2n + 10^5$ é uma função linear
- $10^5n^2 + 10^8n$ é uma função quadrática

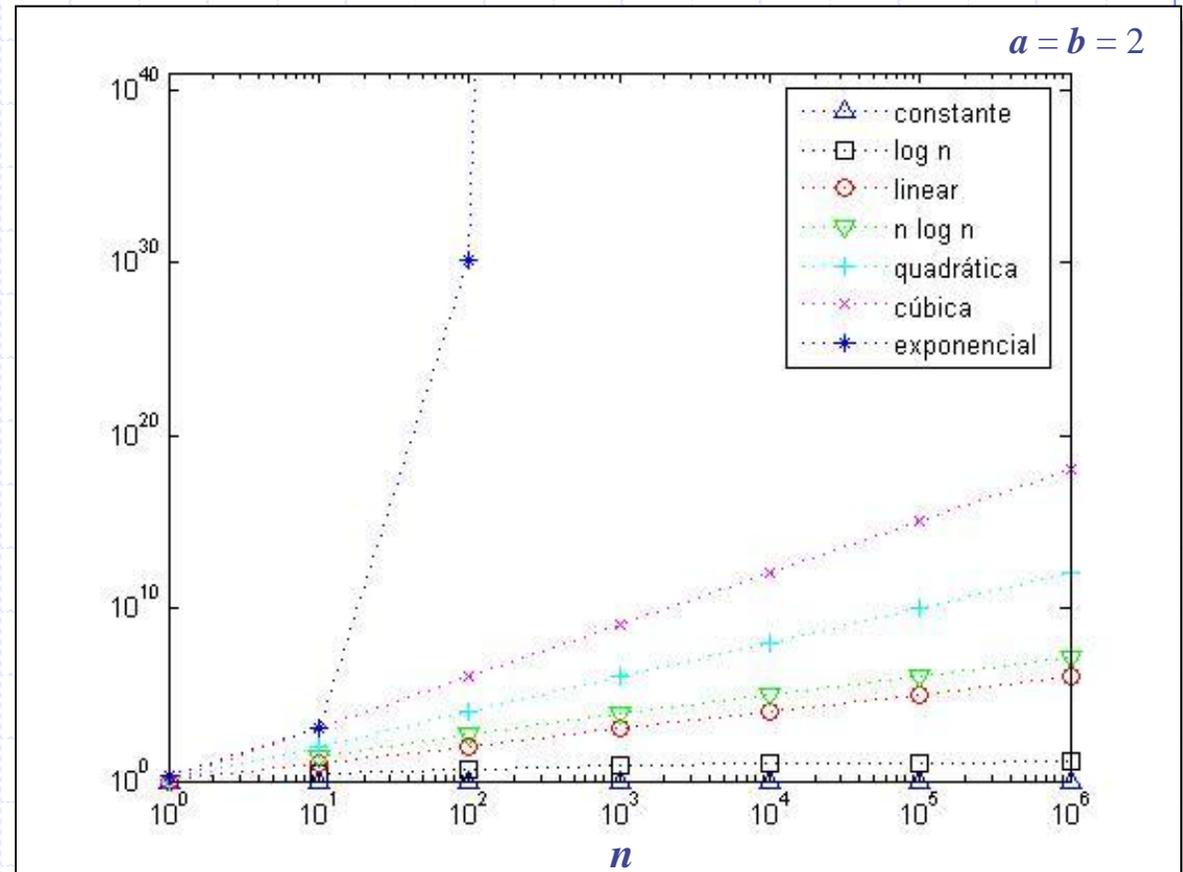


Sete Funções Importantes (Revisão)

◆ 7 funções mais comuns em análise de algoritmos:

- Constante: ~ 1
- Logarítmica: $\sim \log_b n$
- Linear: $\sim n$
- n-log-n: $\sim n \log_b n$
- Quadrática: $\sim n^2$
- Cúbica: $\sim n^3$
- Exponencial: $\sim a^n$

escalas logarítmicas



Despoluindo a Notação

- ◆ Fatores constantes não são significativos
- ◆ Termos de menor ordem também não são
- ◆ Portanto:
 - $4n^5 + 10n^3 + 100$ pode ser reescrito simplesmente como n^5
- ◆ Dizemos nesse caso que:
 - a função $f(n) = 4n^5 + 10n^3 + 100$ é $O(n^5)$

Notação O (Big-O)

◆ Dadas as funções $f(n)$ e $g(n)$, dizemos que " $f(n)$ é $O(g(n))$ " se existirem as *constantes positivas* c (real) e n_0 (inteira) tais que:

- $f(n) \leq cg(n)$ para $n \geq n_0$

◆ Exemplo: $2n + 10$ é $O(n)$

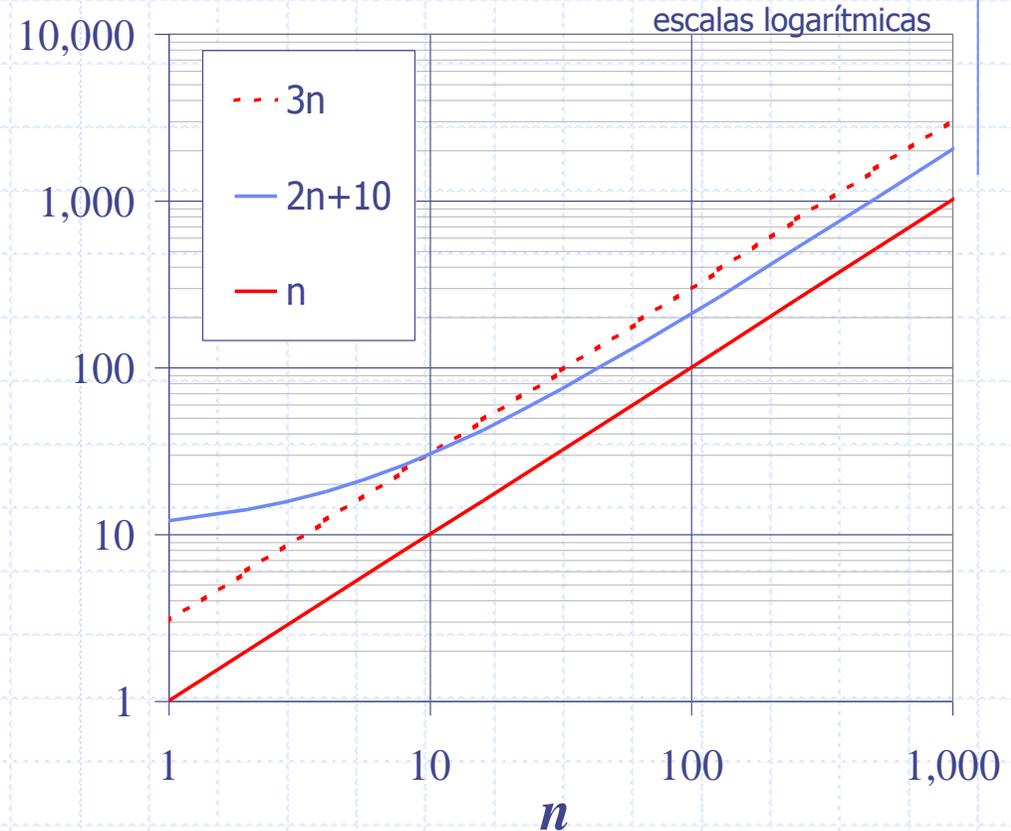
- $2n + 10 \leq cn$

- $cn - 2n \geq 10$

- $(c - 2)n \geq 10$

- $n \geq 10/(c - 2)$

- Pegue $c = 3$ e $n_0 = 10$



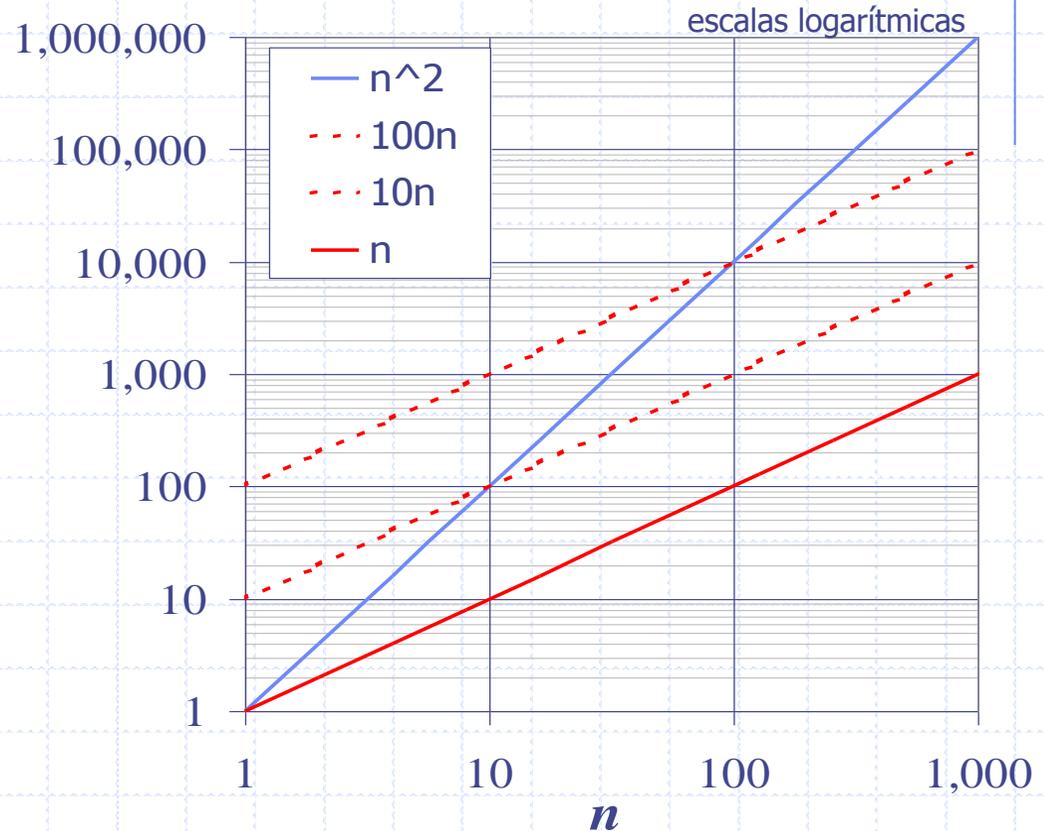
Um Exemplo

◆ Podemos dizer que a função n^2 é $O(n)$?

■ $n^2 \leq cn \Rightarrow n \leq c$

◆ Para que a desigualdade acima seja satisfeita é preciso achar uma constante c que seja sempre maior ou igual a n para todo valor de n

◆ Isso é impossível, pois c deve ser constante



Mais Exemplos Big-O

➤ $7n - 2$ é $O(n)$

- É preciso $c > 0$ e $n_0 \geq 1$ tais que $7n - 2 \leq cn$ para todo $n \geq n_0$
- Isso é verdadeiro, por exemplo, para $c = 7$ e qualquer n_0

➤ $3n^3 + 20n^2 + 5$ é $O(n^3)$

- É preciso $c > 0$ e $n_0 \geq 1$ tais que $3n^3 + 20n^2 + 5 \leq cn^3$ para todo $n \geq n_0$
- Note que $3n^3 + 20n^2 + 5 \leq (3 + 20 + 5)n^3$
- Logo, basta tomar, por exemplo, $c = 28$ e qualquer n_0

➤ $3\log n + 5$ é $O(\log n)$

- É preciso $c > 0$ e $n_0 \geq 1$ tais que $3\log n + 5 \leq c \log n$ para todo $n \geq n_0$
- Note que $3\log n + 5 \leq (3 + 5) \log n$ se $n > 1$ (pois $\log 1 = 0$)
- Logo, basta tomar, por exemplo, $c = 8$ e $n_0 = 2$

Mais Exemplos Big-O

➤ $2n^2 + 100n \log n + 5$ é $O(n^2)$

- É preciso $c > 0$ e $n_0 \geq 1$ tais que $2n^2 + 100n \log n + 5 \leq cn$ para todo $n \geq n_0$
- $2n^2 + 100n \log n + 5 \leq (2 + 100 + 5)n^2$ se $n \geq 1$ (pois $\log n < n$ para $n \geq 1$)
- Isso é verdadeiro, por exemplo, para $c = 107$ e qualquer n_0

➤ $n - 1000 \log n$ é $O(n)$

- É preciso $c > 0$ e $n_0 \geq 1$ tais que $n - 1000 \log n \leq cn$ para todo $n \geq n_0$
- Basta tomar, por exemplo, $c = 1$ e qualquer n_0

➤ 2^{n+2} é $O(2^n)$

- É preciso $c > 0$ e $n_0 \geq 1$ tais que $2^{n+2} \leq c 2^n$ para todo $n \geq n_0$
- Note que $2^{n+2} = 2^n 2^2 = 4 \cdot 2^n$
- Logo, basta tomar, por exemplo, $c = 4$ e qualquer n_0

Exercício

Mostre que $f(n) = \log_y n$ é $O(\log_x n)$ para $x, y > 1$, ou seja, a base logarítmica não afeta a complexidade assintótica da função / algoritmo

Big-O e Taxa de Crescimento

- ◆ Afinal de contas, o que quer dizer " $f(n)$ é $O(g(n))$ "?
 - Formalmente, isso significa que "a função $g(n)$ é um **limitante superior assintótico para $f(n)$** "
 - Interpreta-se como " $f(n)$ cresce a uma taxa menor ou igual a $g(n)$ "
 - Notas:
 - ◆ Embora seja comum, é controverso dizer " $f(n) = O(g(n))$ "
 - ◆ Não é incorreto (embora não seja usual) dizer " $f(n) \in O(g(n))$ ", já que o operador Big-O representa todo um conjunto de funções

Regras Usuais em Big-O

- ◆ Se $f(n)$ for um polinômio de grau d , então $f(n)$ é $O(n^d)$:
 - Despreze os termos de menor ordem
 - Despreze os fatores constantes
- ◆ Use a menor classe de funções que for possível:
 - Diga " $2n$ é $O(n)$ " em vez de " $2n$ é $O(n^2)$ "
- ◆ Use a expressão mais simples:
 - Diga " $3n + 5$ é $O(n)$ " ao invés de " $3n + 5$ é $O(3n)$ "

Regras Usuais em Big-O

◆ Se $f_1(n)$ é $O(g_1(n))$ e $f_2(n)$ é $O(g_2(n))$ então:

■ $f(n) = f_1(n) + f_2(n)$ é $O(\max(g_1(n), g_2(n)))$

◆ Exemplo 1: rotinas consecutivas (execução sequencial)

◆ Exemplo 2: polinômio (vide regra no slide anterior)

■ $f(n) = f_1(n) * f_2(n)$ é $O(g_1(n) * g_2(n))$

◆ Exemplo: $O(g_1(n))$ chamadas a rotina que executa $O(g_2(n))$ operações

■ $(\log n)^a$ é $O(n^b)$ para qualquer a e para b positivo

◆ Logaritmos e suas potências crescem mais devagar que polinomiais

◆ Em particular, para $b = 1$, tem-se $(\log n)^a$ é $O(n)$

Análise Assintótica de Algoritmos

- ◆ A análise assintótica de um algoritmo determina o seu tempo de execução em notação Big-O
- ◆ Para fazer a análise assintótica:
 - Encontra-se o no. de operações primitivas executado pelo algoritmo no pior caso (em função do tamanho da entrada)
 - Escreve-se essa função com notação Big-O
- ◆ Exemplo:
 - Nós determinamos anteriormente que o algoritmo *arrayMax* executa no máximo $8n - 2$ operações primitivas
 - Então, dizemos que o algoritmo *arrayMax* "executa em tempo $O(n)$ " ou "possui **complexidade assintótica linear**"

Algumas Palavras de Precaução

- ◆ A análise assintótica é uma ferramenta fundamental ao projeto, análise ou escolha de um algoritmo específico para uma dada aplicação
- ◆ No entanto, deve-se ter sempre em mente que essa análise “esconde” fatores assintoticamente irrelevantes mas que em alguns casos podem ser relevantes na prática
 - particularmente se o problema de interesse se limitar a entradas (relativamente) pequenas

Algumas Palavras de Precaução

- ◆ Por exemplo, um algoritmo com tempo de execução da ordem de $10^{100}n$ é $O(n)$, assintoticamente melhor do que outro com tempo $10 n \log n$, ou seja, $O(n \log n)$
- ◆ Em princípio, isso nos faria preferir o primeiro...
- ◆ No entanto, 10^{100} é o no. estimado por alguns astrônomos como um limite superior para a quantidade de átomos existente no universo observável!
 - $10 n \log n > 10^{100}n$ apenas para $n > (2 \text{ elevado a } 10^{99}) !$

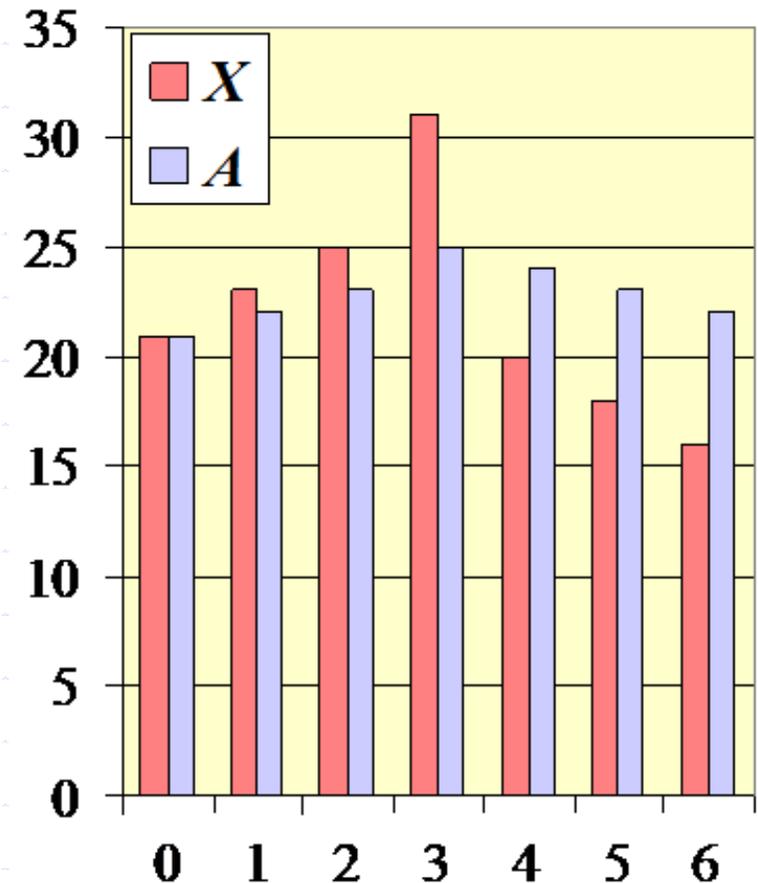
Exemplo (Média de Prefixo)

- ◆ Vamos ilustrar a análise assintótica com dois algoritmos que fazem o cálculo da **média de prefixos**

- ◆ A **i -ésima média de prefixo** de um arranjo X é a média dos $(i + 1)$ primeiros elementos de X :

$$A[i] = (X[0] + X[1] + \dots + X[i]) / (i+1)$$

- ◆ O cálculo do vetor A de médias de prefixo de um outro vetor X possui aplicações em análise financeira



Exemplo (Média de Prefixo)

- ◆ O algoritmo abaixo calcula médias de prefixos em **tempo quadrático**, aplicando a definição:

Algoritmo *Prefix1*(X, n)

Entrada: vetor X de n inteiros

Saída: vetor A de médias de prefixos de X #operações

$A \leftarrow$ novo vetor de n inteiros

para $i \leftarrow 0$ até $n - 1$ **faça**

$\sim n$

$s \leftarrow 0$

$\sim n$

para $j \leftarrow 0$ até i **faça**

$\sim 1 + 2 + \dots + n$

$s \leftarrow s + X[j]$

$\sim 1 + 2 + \dots + n$

$A[i] \leftarrow s / (i + 1)$

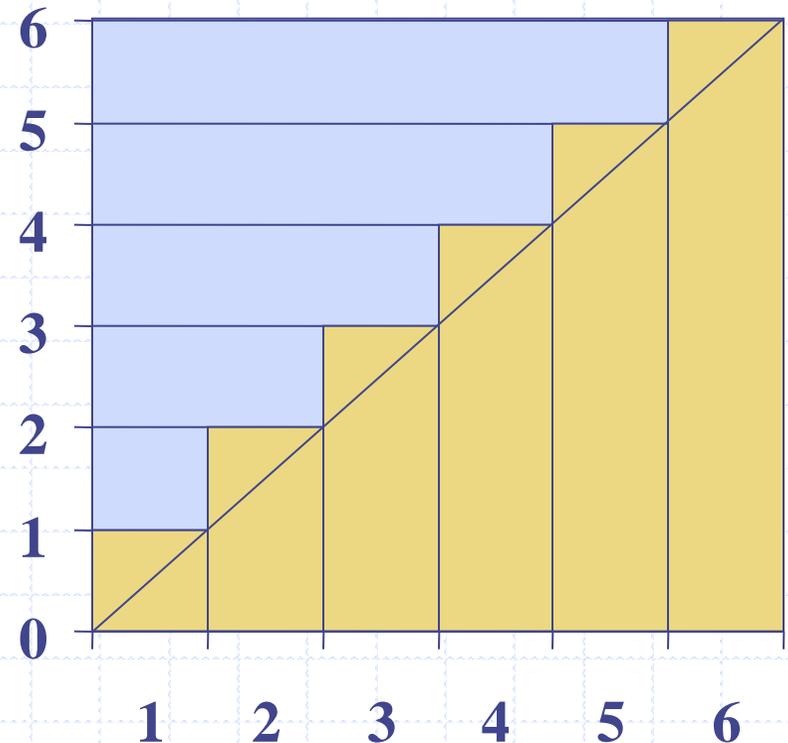
$\sim n$

retorne A

1

Exemplo (Média de Prefixo)

- ◆ O tempo de execução de *Prefix1* é $O(1 + 2 + \dots + n)$
- ◆ A soma dos primeiros n inteiros é $n(n + 1) / 2$ (prog. aritmética):
 - Pode-se provar graficamente
 - Área em laranja ao lado é dada por $n^2/2 + n \cdot 1/2$
- ◆ Portanto, *Prefix1* executa em tempo $O(n^2)$



Exemplo (Média de Prefixo)

- ◆ O algoritmo abaixo calcula a média de prefixos em **tempo linear**, varrendo e somando uma só vez:

Algoritmo *Prefix2*(X, n)

Entrada: vetor X de n inteiros

Saída: vetor A de médias de prefixos de X

#operações

$A \leftarrow$ novo vetor de n inteiros

$s \leftarrow 0$

1

para $i \leftarrow 0$ até $n - 1$ **faça**

$\sim n$

$s \leftarrow s + X[i]$

$\sim n$

$A[i] \leftarrow s / (i + 1)$

$\sim n$

retorne A

1

Exercício

Faça uma análise **detalhada** do no. de operações primitivas computadas por cada um dos algoritmos anteriores para cálculo da média de prefixo e mostre o número exato de operações executadas por cada um deles. Em seguida, prove usando a **definição** que *Prefix1* é $O(n^2)$ e *Prefix2* é $O(n)$. Discuta se existem diferenças entre pior caso, melhor caso e caso médio desses algoritmos. Justifique

Algumas Habilidades Necessárias



◆ Somatórios:

- PAs, PGs, etc

◆ Logaritmos e Potências

◆ Técnicas de prova:

- Exemplos e Contra-Exemplos
- Contradição
- Indução

◆ Teoria de Probabilidade

◆ **Propriedades de Logaritmos:**

$$\log_b(xy) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b x^a = a \cdot \log_b x$$

$$\log_b a = \log_x a / \log_x b$$

◆ **Propriedades de Potências:**

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b / a^c = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c \cdot \log_a b}$$



Parentes do Big-O

◆ Big-Omega:

- $f(n)$ é $\Omega(g(n))$ se existir uma constante real positiva c e uma constante inteira positiva n_0 tais que $f(n) \geq c g(n)$ para $n \geq n_0$
- Exemplo: $f(n) = 5n^2$ é $\Omega(n^2)$
 - As constantes $c = 5$ e $n_0 = 1$ satisfazem a condição
- Ao contrário do Big-O, toma-se a **maior classe** de funções possível
 - ◆ Diz-se $5n^2$ é $\Omega(n^2)$, embora $g(n) = n \log n$, n , $\log n$ e 1 sejam válidas
 - ◆ Já $g(n) = n^3$, $g(n) = n^4$, $g(n) = 2^n$, etc, não satisfazem a definição



Parentes do Big-O

◆ Exemplo:

- Seja uma função definida como $f(n) = 10n^2 + 5n$ para valores pares de n e como $f(n) = 3n + 3$ para valores ímpares de n
- É possível mostrar que $f(n)$ é $\Omega(n)$ e $O(n^2)$

◆ Exemplo:

- Seja um algoritmo que execute $T(n) = 8n + 3$ operações primárias no **pior caso** (correspondente a uma dada composição da sua entrada, de tamanho n) e $T(n) = 15$ operações no **melhor caso**
- A complexidade assintótica desse algoritmo é $\Omega(1)$ e $O(n)$
 - **constante** no *melhor caso* e **linear** no *pior caso*



Parentes do Big-O

◆ Big-Teta:

- $f(n)$ é $\Theta(g(n))$ se existirem duas constantes reais positivas c' e c'' e uma constante inteira positiva n_0 tais que $c'g(n) \leq f(n) \leq c''g(n)$ para $n \geq n_0$
- Em outras palavras, $f(n)$ é $\Theta(g(n))$ se for ao mesmo tempo $\Omega(g(n))$ e $O(g(n))$
- Exemplo: $f(n) = 5n^2$ é $\Theta(n^2)$
 - $c' = c'' = 5$ satisfazem a condição para qualquer n_0
- De fato, qualquer polinômio de ordem d é $\Theta(n^d)$



Parentes do Big-O

◆ little-O:

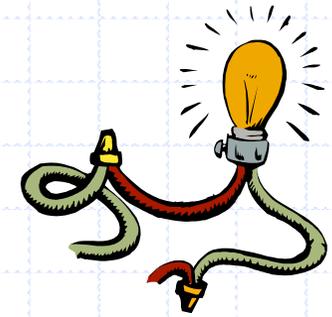
- $f(n)$ é $o(g(n))$ se para **qualquer** constante real positiva c existe uma constante inteira positiva n_0 tal que $f(n) < c g(n)$ para $n \geq n_0$
- Exemplo: $f(n) = 5n$ é $o(n^2)$
 - $5n < c n^2$ para qualquer $c > 0$ e $n > 5/c$
 - Exercício: $f(n) = 5n$ é $o(n)$? Justifique



Parentes do Big-O

◆ little-Omega:

- $f(n)$ é $\omega(g(n))$ se para **qualquer** constante real positiva c existe uma constante inteira positiva n_0 tal que $f(n) > c g(n)$ para $n \geq n_0$
- Exemplo: $f(n) = 5n^2$ é $\omega(n)$
 - $5n^2 > c n$ para qualquer $c > 0$ e $n > c/5$
 - Exercício: $f(n) = 5n^2$ é $\omega(n^2)$? Justifique



Trocando em Miúdos ...

Big-O

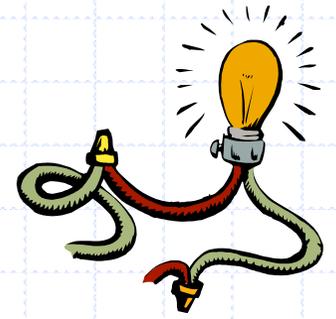
- $f(n)$ é $O(g(n))$ se $f(n)$ cresce a uma **taxa menor ou igual** a $g(n)$

Big-Omega

- $f(n)$ é $\Omega(g(n))$ se $f(n)$ cresce a uma **taxa maior ou igual** a $g(n)$
 - Nota: $f(n)$ é $\Omega(g(n))$ se e somente se $g(n)$ é $O(f(n))$

Big-Teta

- $f(n)$ é $\Theta(g(n))$ se for **Big-O e Big-Omega** ao mesmo tempo
- $f(n)$ e $g(n)$ **crecem a uma mesma taxa** assintótica
 - Nota: $f(n)$ é $\Theta(g(n))$ se e somente se $g(n)$ é $\Theta(f(n))$



Trocando em Miúdos ...

Little-O

- $f(n)$ é $o(g(n))$ se $f(n)$ cresce a uma **taxa menor** que $g(n)$

Little-Omega

- $f(n)$ é $\omega(g(n))$ se $f(n)$ cresce a uma **taxa maior** que $g(n)$
 - Nota: $f(n)$ é $\omega(g(n))$ se e somente se $g(n)$ é $o(f(n))$

Forma Assintótica usando Limites

Extraído do material auxiliar da Profa. M. Carolina Monard

Forma Assintótica	Relação	Definição
$f(n) \in \Theta(g(n))$	$f(n) \equiv g(n)$	$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$
$f(n) \in O(g(n))$	$f(n) \preceq g(n)$	$0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$
$f(n) \in \Omega(g(n))$	$f(n) \succeq g(n)$	$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$
$f(n) \in o(g(n))$	$f(n) \prec g(n)$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \omega(g(n))$	$f(n) \succ g(n)$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$

Exemplo 1: $f(n) = 2n^2+n$ e $g(n) = n^2$

- $\lim_{n \rightarrow \infty} (2n^2+n / n^2) = \lim_{n \rightarrow \infty} (2 + 1/n) = 2$

Exemplo 2: $f(n) = n^2$ e $g(n) = 2n^2+n$

- $\lim_{n \rightarrow \infty} (n^2 / 2n^2+n) = [L'Hôpital 2x] \lim_{n \rightarrow \infty} (2 / 4) = 1/2$

Exercícios Adicionais

- ◆ Seja $f(n)$ que assume o valor $3n^2$ se n for par ou $7n$ se n for ímpar. Mostre que essa função é $O(n^2)$ e $\Omega(n)$
- ◆ Qual é o maior tamanho de entrada n de um problema P tal que P pode ser resolvido em tempo t se o algoritmo para resolver P leva $f(n)$ micro-segundos? Responda em uma tabela com todas as combinações de $t = 1s, 1hora, 1mês, 1 século$ e $f(n) = \log(n), n, n \cdot \log(n), n^2, n^3, 2^n$. Relacione os resultados obtidos com as discussões em aula
- ◆ Qual a complexidade assintótica de **pior caso** para o tempo de execução do algoritmo de busca binária iterativo visto em aula? Justifique formalmente usando a definição do operador BIG-O

Exercícios Adicionais

- ◆ Diz-se que um algoritmo que executa um no. constante de operações primitivas (que independe do tamanho n da entrada) “executa em tempo $O(1)$ ” ou simplesmente “é $O(1)$ ”. Mostre, usando a definição formal do operador BIG-O, que qualquer função $f(n) = d$, onde d é uma constante positiva qualquer, é $O(1)$
- ◆ Explique porque, no **melhor caso** (com relação às possíveis composições da entrada) o tempo de execução do algoritmo de busca binária iterativo visto em aula é $O(1)$
 - ◆ Pode-se então concluir que esse algoritmo é $\Omega(1)$?
- ◆ Ordene as seguintes funções de acordo com as suas taxas de crescimento assintótico (justifique!): $4*n*\log(n) + 2n$, 2^{10} , $2^{\log(n)}$, $3n + 100\log(n)$, $4n$, 2^n , 3^{3n} , $n^2 + 10n$, n^3 , $n*\log(n)$

Leitura e Exercícios

- ◆ Slides do Prof. João Luis Rosa
- ◆ Notas da Profa. Maria Carolina Monard
- ◆ Lista de Exercícios Extras

Vide Webpage da Disciplina!

Bibliografia

- ◆ N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004
- ◆ T. H. Cormen et al., *Introduction to Algorithms*, MIT Press, 2nd Edition, 2001
- ◆ M. T. Goodrich & R. Tamassia, *Data Structures and Algorithms in Java*, John Wiley & Sons, 3rd Edition, 2004