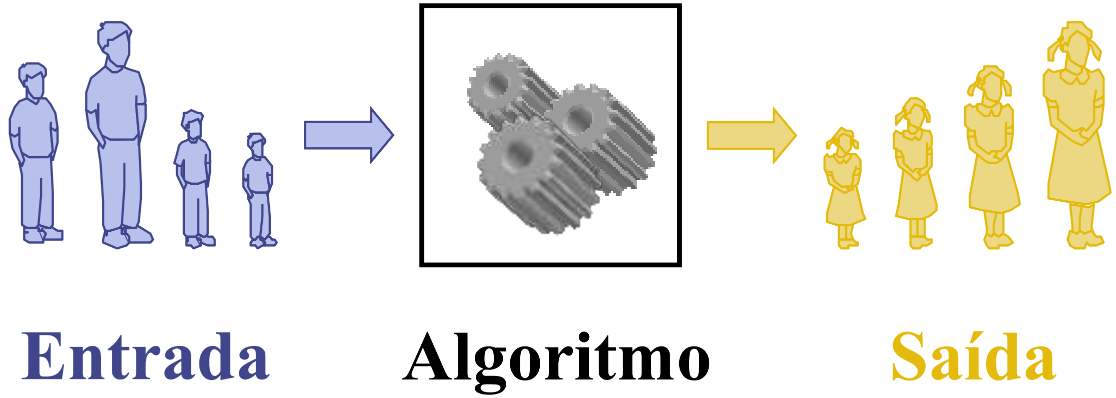
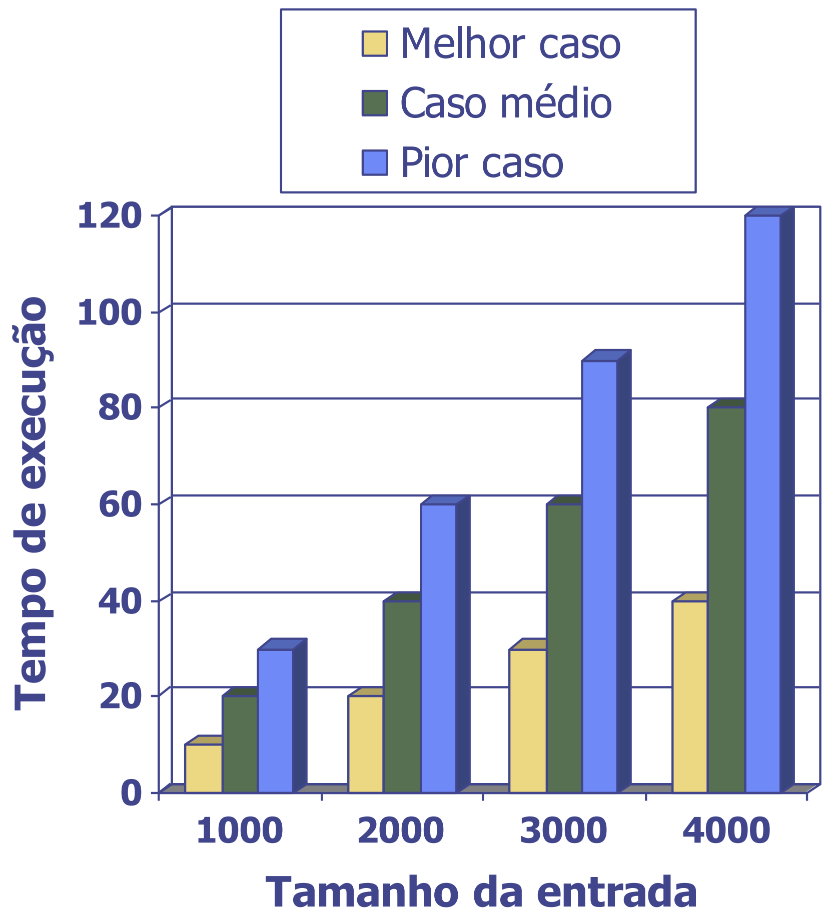
Revisão Análise de algoritmos



Tempo de execução

- O tempo de execução de um algoritmo varia de acordo com a entrada e tipicamente cresce com o tamanho da entrada
- Difícil determinar o caso médio

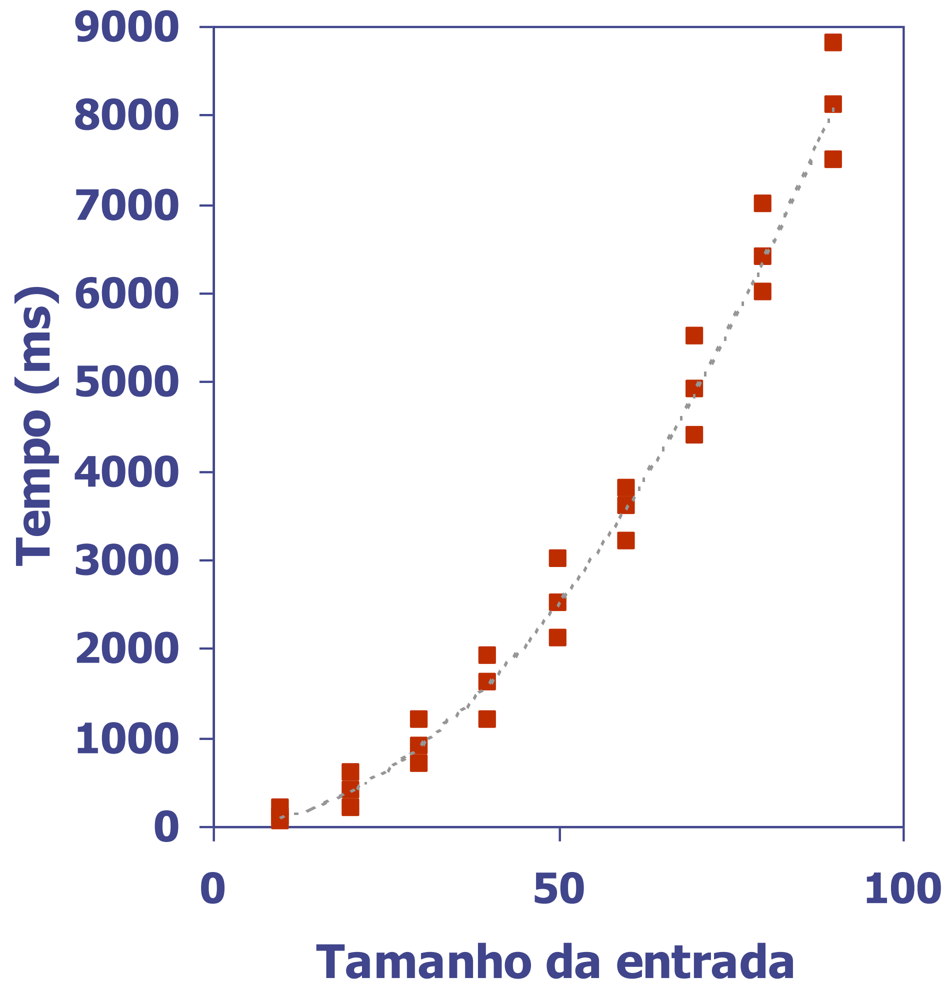


Tempo de execução

- Geralmente o foco é na análise do pior caso
  - Mais fácil de analisar
  - O tempo de execução no pior caso é um limite superior sobre o tempo de execução para qualquer entrada. Existe a garantia de que a execução não irá demorar mais tempo
  - Para alguns algoritmos o pior caso acontece com muita freqüência
  - Muitas vezes, o caso médio é quase tão ruim quanto o pior caso

Estudos experimentais

- Escreva um programa que implemente um algoritmo
- Execute o programa com vários casos de teste
- Use um método como System.currentTimeMillis() para medir o tempo de execução
- Faça um gráfico com os resultados



Estudos experimentais

- **Experimento 1**
  - Foram feitas dez amostragens para n igual a 10000, 100000 e 1000000

Número Experimento	Tempo em ms/n=1000	Tempo em ms/ n-10000	Tempo em ms/ n-100000	Tempo em ms/ n-1000000
1	**187**	266	**5641**	**32563**
2	94	312	4000	
3	94	297	3844	
4	94	312	3266	
5	78	**859**	3250	
6	78	265	3766	
7	93	296	3718	
8	78	281	2719	
9	78	297	3078	
10	94	313	3640	

```
public class teste {
 public static void main(String[] x) {
    long t1=System.currentTimeMillis();
    System.out.println("T1"+t1);
    for(int f=0;f<10000;f++){
       System.out.println("testando..."+f);
    }
    long t2=System.currentTimeMillis();
    System.out.println("Tempo Total:");
    System.out.print(t2-t1);
 }
}
```

Estudos experimentais

- Experimento 2
  - Entrada de mesmo tamanho, mas com instâncias diferentes do mesmo problema

Número Experimento	Primeiro elemento Tempo em ms/ n=1000000	Último elemento Tempo em ms/ n=1000000
1	0	15
2	0	16
3	0	15
4	0	15
5	0	16
6	0	15
7	0	15
8	0	16
9	0	15
10	0	15

```
public int acha(int a[],int ch){
    for (int f=0;f<a.length;f++){
        if (ch==a[f])
            return a[f];
    }
    return -1;
}
```

Limitação dos experimentos

- É necessário implementar o algoritmo para poder estudar o seu comportamento, o que pode ser difícil
- Resultados podem não ser indicativos do tempo de execução sobre entradas não incluídas nos casos de teste
- Para comparar dois algoritmos, as mesmas configurações de *software* e *hardware* devem ser usadas

Análise teórica

- Usa uma descrição de alto nível do algoritmo
- Leva em consideração todas as entradas possíveis
- Nos permite avaliar a velocidade de um algoritmo independente das configurações de *hardware* e *software*

Análise teórica

- Associa a cada algoritmo uma função $f(n)$ que representa o tempo de execução do algoritmo como uma função do tamanho n da entrada. Ex.: n, n^2, $\log n$

Pseudo-código

- Descrição de alto nível do algoritmo
- Mais estruturado que português
- Menos detalhado que um programa
- Melhor notação para descrever algoritmos
- Esconde detalhes inerentes da programação

Exemplo: Encontrar o maior elemento de um arranjo

```
Algoritmo maiorArray(A, n)
  Entrada array A de n inteiros
  Saída maior elemento de A

  maior ← A[0]
  para i ← 1 até n – 1 faça
    se (A[i] > maior)
      maior ← A[i]
  retorne maior
```

Operações primitivas

- Computações básicas realizadas por um algoritmo
- Identificável no pseudo-código
- Muito independente da linguagem de programação
- Definição exata não é tão importante (veremos mais a frente)

- Exemplos:
  - Avaliação de expressões
  - Atribuição de um valor a uma variável
  - Indexando um elemento de um arranjo (*array*)
  - Chamando um método
  - Retornando de um método

Contando operações primitivas

- Inspecionando o pseudo-código, é possível determinar o número máximo de operações primitivas executadas por um algoritmo, em função do tamanho da entrada

Algoritmo *maiorArray*(A, n)	# operações
$maior \leftarrow A[0]$	2
para $i \leftarrow 1$ **até** $n - 1$ **faça**	$1 + n$
se $(A[i] > maior)$	$2(n - 1)$
$maior \leftarrow A[i]$	$2(n - 1)$
{ incrementar i }	$2(n - 1)$
retorne *maior*	1
Total	$7n - 2$

Estimando tempo de execução

- Algoritmo ***maiorArray*** executa $7n - 2$ operações primitivas no pior caso
- Defina
  - a Tempo da operação primitiva mais rápida
  - b Tempo da operação primitiva mais lenta
- Seja $T(n)$ O tempo de execução de ***maiorArray*** no pior caso. Temos

$$a\,(7n - 2) \le T(n) \le b(7n - 2)$$

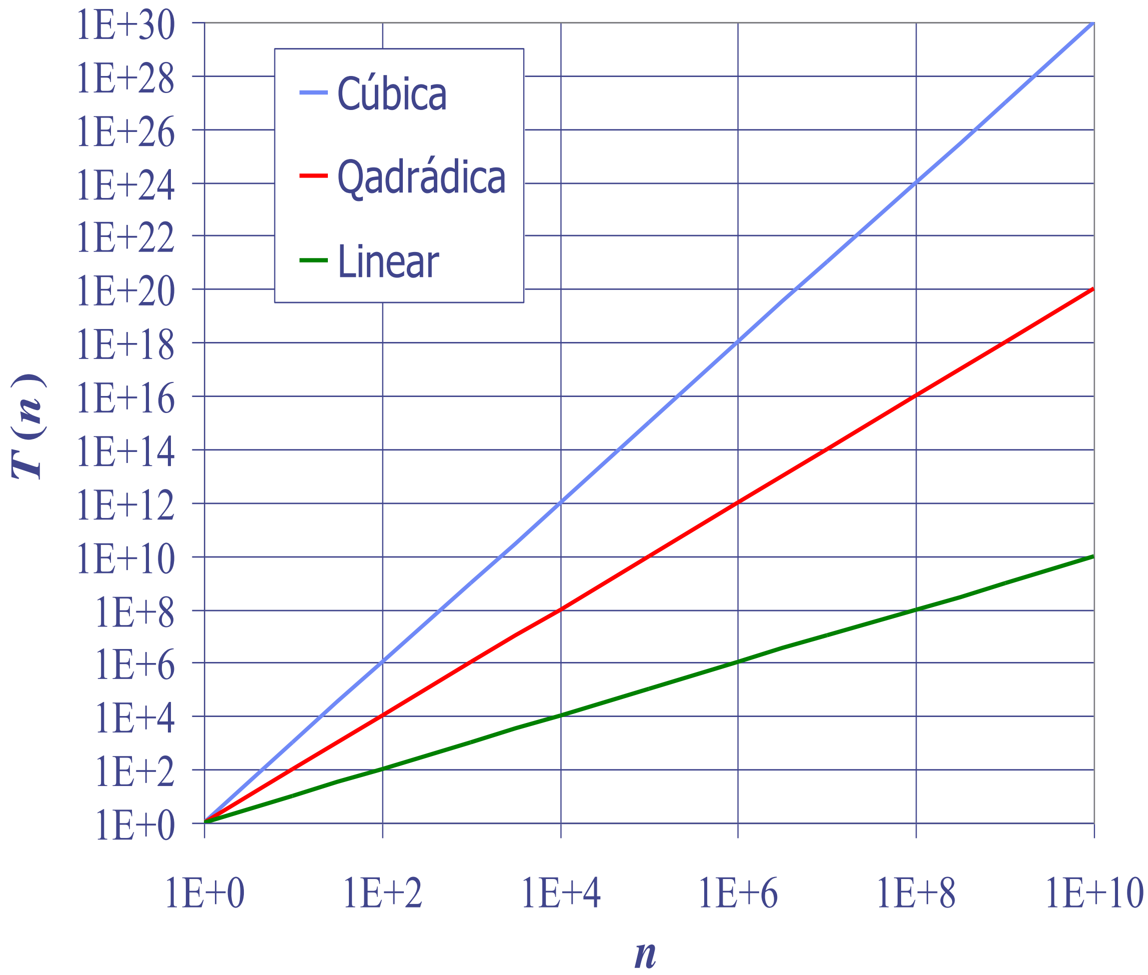
- Dessa forma, o tempo de execução $T(n)$ é limitado por duas funções lineares

Taxa de crescimento

- Mudando a configuração de *hardware/ software*
  - Afeta $T(n)$ de forma constante, mas
  - Não altera a taxa de crescimento de $T(n)$
- A taxa de crescimento linear de $T(n)$ é uma propriedade intrínseca do algoritmo ***maiorArray***

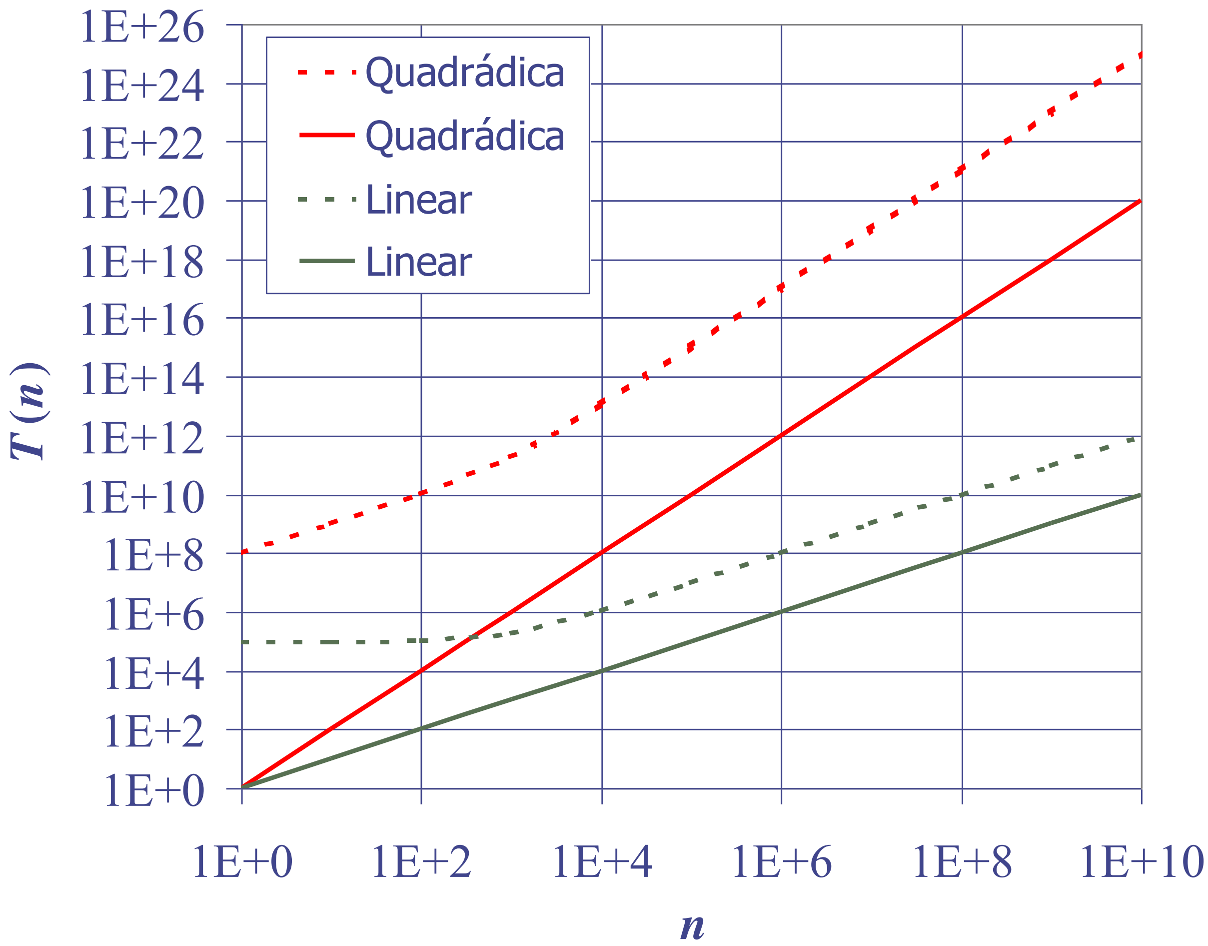
Taxas de crescimento

- Funções de taxa de crescimento:
  - Linear $\approx n$
  - Quadradica $\approx n^2$
  - Cúbica $\approx n^3$
- Em um gráfico de escala logarítmica, a inclinação das linhas corresponde a ***taxa*** de crescimento das funções



Fatores constantes

- A taxa de crescimento não é afetada por:
  - Fatores constantes
  - Termos de baixa ordem
- Exemplos
  - $10^2 n + 10^5$ é uma função linear
  - $10^5 n^2 + 10^8 n$ é uma função quadrádica

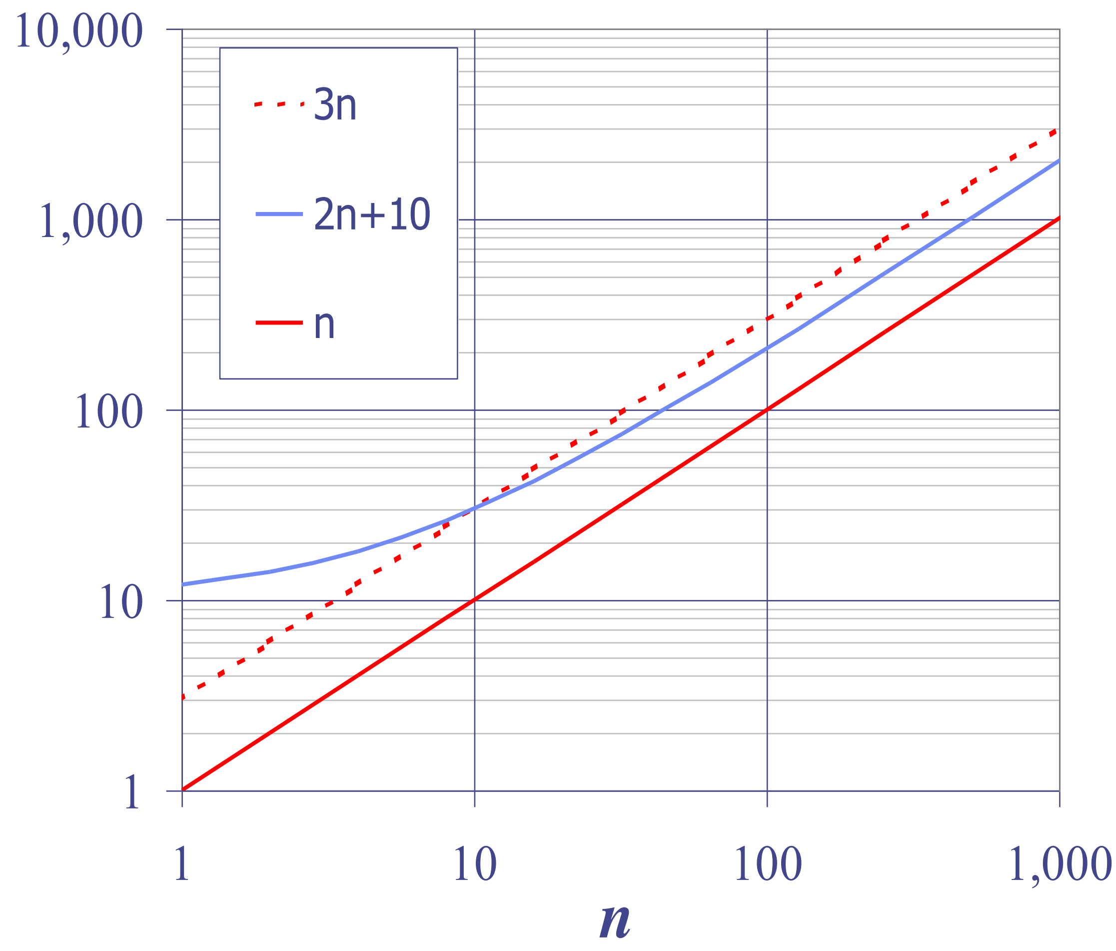


Notação *Big-Oh*

- Dada funções $f(n)$ e $g(n)$, podemos dizer que $f(n)$ é $O(g(n))$ se existem constantes positivas c e n_0 tal que

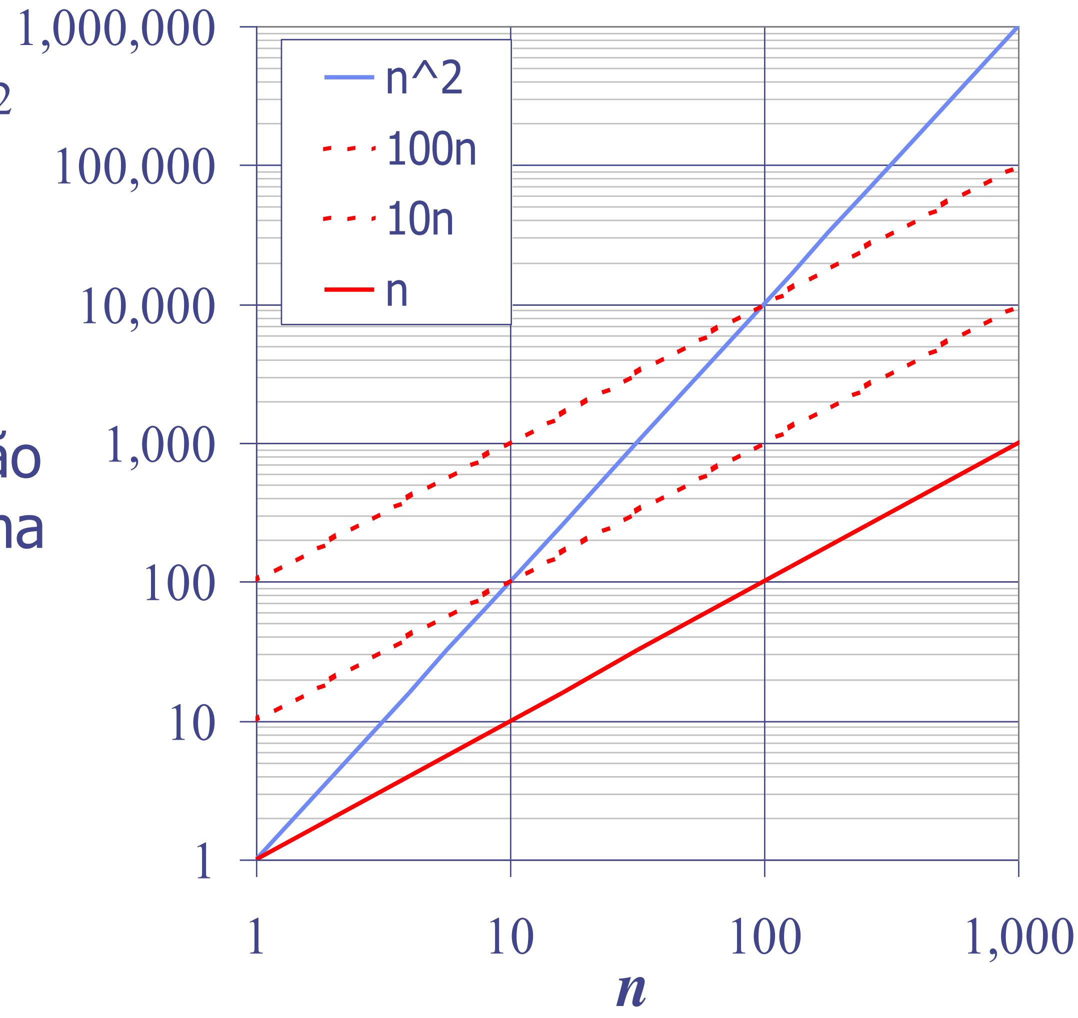
  $f(n) \leq cg(n)$ para $n \geq n_0$

- Exemplo: $2n + 10$ é $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pegue $c = 3$ e $n_0 = 10$



Notação *Big-Oh* (cont.)

- Exemplo: a função n^2 não é $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - A inequação acima não pode ser satisfeita uma vez que c deve ser uma constante



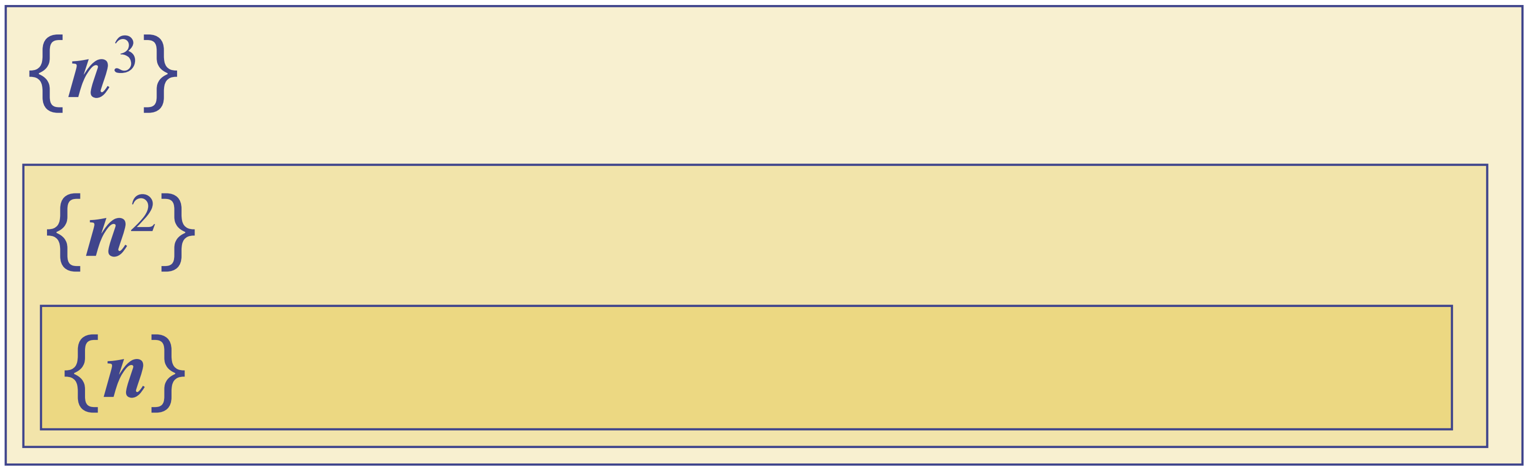
Big-Oh e taxa de crescimento

- A notação *big-Oh* dá o limite superior da taxa de crescimento de uma função
- A afirmação "$f(n)$ é $O(g(n))$" quer dizer que a taxa de crescimento de $f(n)$ não é maior que a taxa de crescimento de $g(n)$
- Podemos usar a notação *big-Oh* para ranquear funções de acordo com suas taxas de crescimento

	$f(n)$ é $O(g(n))$	$g(n)$ é $O(f(n))$
$g(n)$ cresce mais	Sim	Não
$f(n)$ cresce mais	Não	Sim
Mesmo cresc.	Sim	Sim

Classes de funções

- $\{g(n)\}$ denota a classe (conjunto) de funções que são $O(g(n))$
- Temos que
  $\{n\} \subset \{n^2\} \subset \{n^3\} \subset \{n^4\} \subset \{n^5\} \subset \ldots$
  onde a continência é estrita



Regras *Big-Oh*

- Se $f(n)$ é polinomial de grau d, então $f(n)$ é $O(n^d)$, i.e.,
  1. Elimine termos de baixa ordem
  2. Elimine fatores constantes
- Use a menor classe possível de funções
  - Diga "$2n$ é $O(n)$" ao invés de "$2n$ é $O(n^2)$"
- Use a expressão mais simples da classe
  - Diga "$3n + 5$ é $O(n)$" ao invés de "$3n + 5$ é $O(3n)$"

Exemplo Big-Oh

- Analisando o algoritmo ao lado, percebemos que para o pior caso ele é $O(n^3)$. Ou seja, quando x for diferente de 0 o produto das matrizes A e B será efetuado

Algoritmo A(*A*,*B*, x)
  Entrada array ***A*,*B* e o int *x***
  Saída Soma / Produto de A,B
  if (x=0){ // soma de A e B
    para i $\leftarrow$ 1 até n faça
      para j $\leftarrow$ 1 até n faça
        $c_{ij} \leftarrow a_{ij} + b_{ij}$
  }else{
    para i $\leftarrow$ 1 até n faça
      para j $\leftarrow$ 1 até n faça
        $c_{ij} \leftarrow 0$
          para k $\leftarrow$ 1 até n faça
      $c_{ij} \leftarrow c_{ij} + a_{ik} * b_{kj}$

Notação Ômega

- Dada funções $f(n)$ e $g(n)$, podemos dizer que $f(n)$ é $\Omega(g(n))$ se existem constantes positivas c e n_0 tal que

  $cg(n) \leq f(n)$ para $n \geq n_0$
- Define uma cota assintótica inferior
- Pode-se dizer que a notação omega é a do melhor caso e é geralemente limitada pelos limite inferiores triviais (determinados pelo tamanho da entrada)
- Exemplo: a função $g(n)=7n^3+5$, cresce menos rapidamente do que uma função exponencial $f(n)=2^n$. Diz-se que $f(n)= \Omega(g(n))$

Exemplo Notação Ômega

- Analisando o algoritmo ao lado, percebemos que para o melhor caso ele é $\Omega(n^2)$. Ou seja, quando x for 0 a soma das matrizes A e B será efetuada

Algoritmo A(*A*,*B*, x)
  Entrada array ***A*,*B* e o int x**
  Saída Soma / Produto de A,B
  if (x=0){ // soma de A e B
    para i **← 1 até n faça**
      para j ← 1 até n faça
        $c_{ij} \leftarrow a_{ij} + b_{ij}$
  }else{
    para i ← 1 até n faça
      para j ← 1 até n faça
        $c_{ij} \leftarrow 0$
          para k ← 1 até n faça
        $c_{ij} \leftarrow c_{ij} + a_{ik} * b_{kj}$

Notação Teta

- Dada funções $f(n)$ e $g(n)$, podemos dizer que $f(n)$ é $\Theta(g(n))$ se existem constantes positivas c_1, c_2 e n_0 tal que

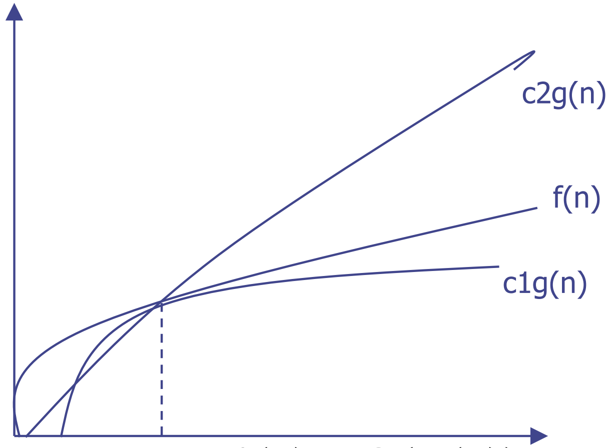
  $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ para $n \geq n_0$
- Define uma cota assintótica exata
- Pode-se dizer que a notação teta é a do caso médio ou esperado
- É mais complicada para determinar, pois utiliza conceitos de probabilidades

Exemplo Notação Teta

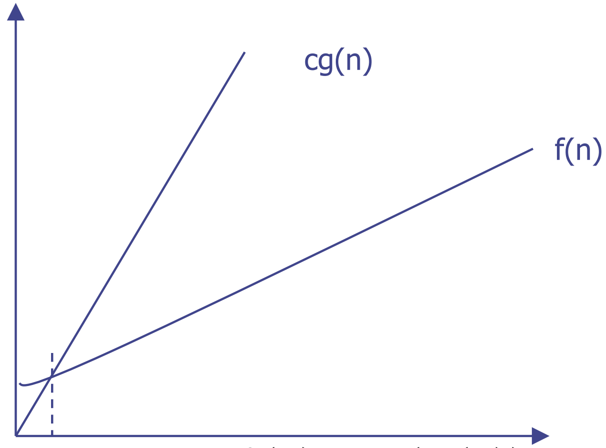
- Sendo p a probabilidade de x=0 (melhor caso) e q a probabilidade de x!=0 (pior caso) e que p+q=1
- A complexidade no caso médio é dado por $p*n^2 + (1-p)*n^3$

Algoritmo A(*A*,*B*, x)
  Entrada array ***A*,*B* e o int x**
  Saída Soma / Produto de A,B
  if (x=0){ // soma de A e B
    para i ← 1 até n faça
      para j ← 1 até n faça
        $c_{ij} \leftarrow a_{ij} + b_{ij}$
  }else{
    para i ← 1 até n faça
      para j ← 1 até n faça
        $c_{ij} \leftarrow 0$
          para k ← 1 até n faça
      $c_{ij} \leftarrow c_{ij} + a_{ik} * b_{kj}$

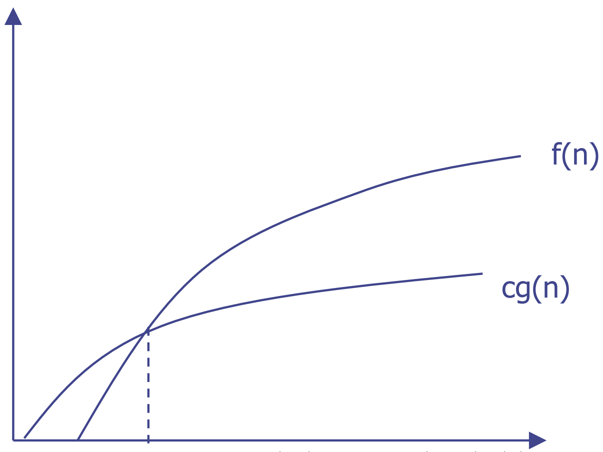
Gráficos da Notação O,Teta e Ômega



$$f(x) = \Theta(g(x))$$



$$f(x) = O(g(x))$$



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

Algoritmos Ótimos

- Um algoritmo é definido como sendo ótimo quando sua cota assintótica inferior conhecida é igual a sua cota assintótica superior
- Exemplo: Algoritmo para inversão de seqüências

$$\mathrm{O}(n) = \Omega(n)$$

  - Limite assintótico inferior trivial (baseado no tamanho da entrada) *n* elementos de entrada é igual ao limite no pior caso

Análise assintótica de algoritmos

- A análise assintótica de algoritmos determina o tempo de execução na notação big-Oh
- Para fazer a análise assintótica
  - Encontramos o número de operações primitivas executadas no pior caso em função da entrada
  - Expressamos esta função usando a notação big-Oh
- Exemplo:
  - Determinamos que o algoritmos ***maiorArray*** executa pelo menos $7n - 2$ operações primitivas
  - Podemos dizer que o algoritmo ***maiorArray*** "executa em tempo $O(n)$"
- Como fatores constantes e termos de baixa ordem serão retirados, podemos ignorá-los quando contando as operações primitivas

Exemplos de Análises Assintóticas

```
O(1)
S
------------------------------
O(n)
para i ← 1 até n faça
   S
------------------------------
O(n²)
para i ← 1 até n faça
     para j ← 1 até n faça
   S
para i ← 1 até n faça
     para j ← i até n faça
   S
------------------------------
```

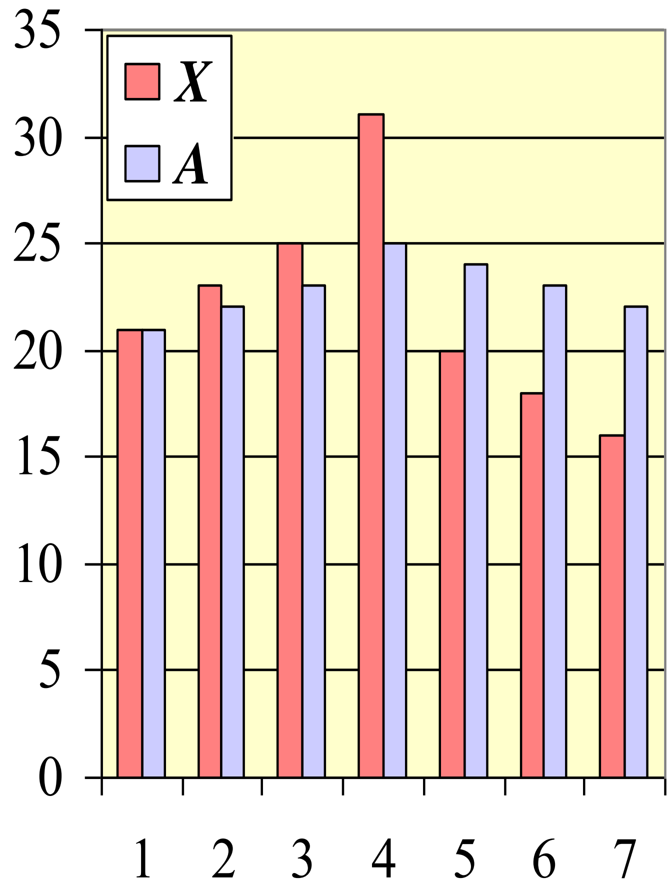
```
O(log n)
Enquanto n>1 faça
     n ← n div 2;
     S
------------------------------

O(n log n)
para i ← 1 até n faça
     m ← n
     Enquanto m>1 faça
        m ← m div 2;
        S
------------------------------
```

Computação de médias pré-fixadas

- Ilustraremos a análise assintótica com dois algoritmos de médias pré-fixadas.
- A i-ésima média pré-fixada de um *array* X é a média dos primeiros $(i + 1)$ elementos de X

$$A[i] = X[0] + X[1] + \ldots + X[i]$$

- Médias pré-fixadas são bastante usadas em aplicações de análise financeira



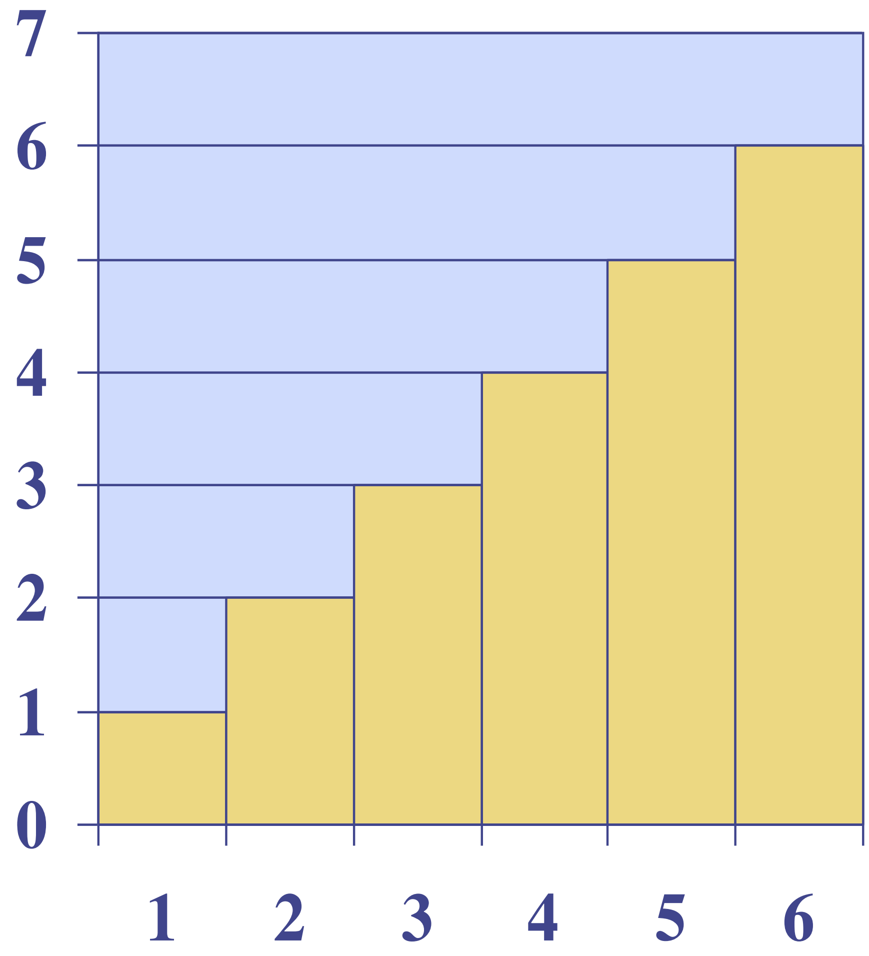
Média pré-fixada (quadrádica)

- O algoritmo a seguir computa as médias pré-fixadas em tempo quadrádico aplicando a definição

	operações
Algoritmo *mediasPrefixada1*(X, n)	
Entrada array X de n inteiros	
Saída array A das médias pré-fixadas	
$A \leftarrow$ novo *array* de n inteiros	n
para $i \leftarrow 0$ **até** $n - 1$ **faça**	n
$s \leftarrow X[0]$	n
para $j \leftarrow 1$ **até** i **faça**	$1 + 2 + \ldots + (n - 1)$
$s \leftarrow s + X[j]$	$1 + 2 + \ldots + (n - 1)$
$A[i] \leftarrow s / (i + 1)$	n
retorne A	1

Tempo de execução

- O tempo de execução de ***mediasPrefixadas1*** é $O(1 + 2 + \ldots + n)$
- A soma dos n primeiros inteiros é $n(n + 1) / 2$
- Então, o algoritmo ***mediasPrefixadas1*** roda em tempo $O(n^2)$



Média pré-fixada (linear)

- O seguinte algoritmo computa as médias pré-fixadas em tempo linear mantendo a soma parcial

Algoritmo *mediasPrefixada2*(X, n)	
Entrada array X de n inteiros	
Saída array A das médias pré-ficadas de	operações
$A \leftarrow$ novo *array* de n inteiros	n
$s \leftarrow 0$	1
para $i \leftarrow 0$ **até** $n - 1$ **faça**	n
$s \leftarrow s + X[i]$	n
$A[i] \leftarrow s / (i + 1)$	n
retorne A	1

- Algoritmo ***mediasPrefixadas2*** roda em tempo $O(n)$