

CTC-12

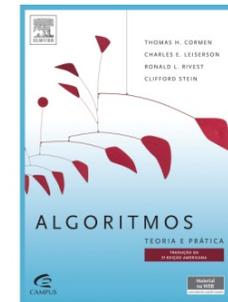
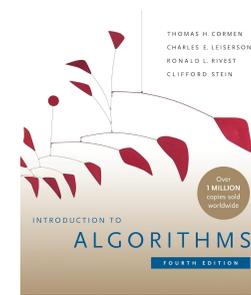


Projeto e Análise de Algoritmos

Carlos Alberto Alonso Sanches

Bibliografia

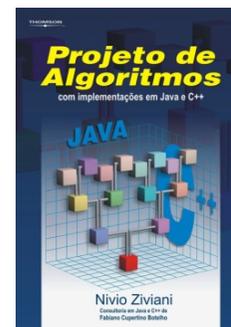
- T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein
Introduction to Algorithms



- A.V. Aho, J.E. Hopcroft and J.D. Ullman
Data Structures and Algorithms

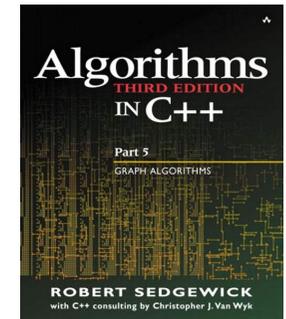
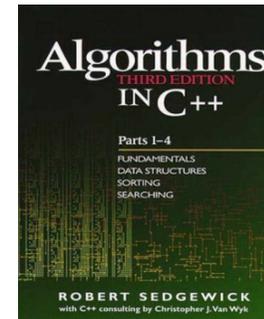


- N. Ziviani
Projeto de Algoritmos



Bibliografia complementar

- R. Sedgwick and K. Wayne
Algorithms



- A. Drozdek
Estrutura de dados e algoritmos em C++



- Aulas de P. Feofiloff

www.ime.usp.br/~pf/analise_de_algoritmos/lectures.html

- Simulação de algoritmos

www.cs.usfca.edu/~galles/visualization/Algorithms.html

www.cs.princeton.edu/courses/archive/spr10/cos226/lectures.html

www.jasonpark.me/AlgorithmVisualizer/?ref=producthunt

Primeiro bimestre



- Ordem de funções (Capítulo 1)
- Algoritmos recursivos (Capítulo 2)
- Árvores balanceadas (Capítulo 3)
- Ordenação (Capítulos 4 e 5)

- Lista de exercícios: 1 a 21

<http://www.comp.ita.br/~alonso/ensino.html>

Segundo bimestre

- Busca de padrões (Capítulo 6)
 - Algoritmos em grafos (Capítulos 7, 8 e 9)
 - Paradigmas de programação (Capítulo 10)
 - Algoritmos probabilísticos e aproximativos (Capítulo 11)
-
- Lista de exercícios: 22 a 46

alonso@ita.br, sala 112 (Ele/Comp), ramal 5985

CTC-12

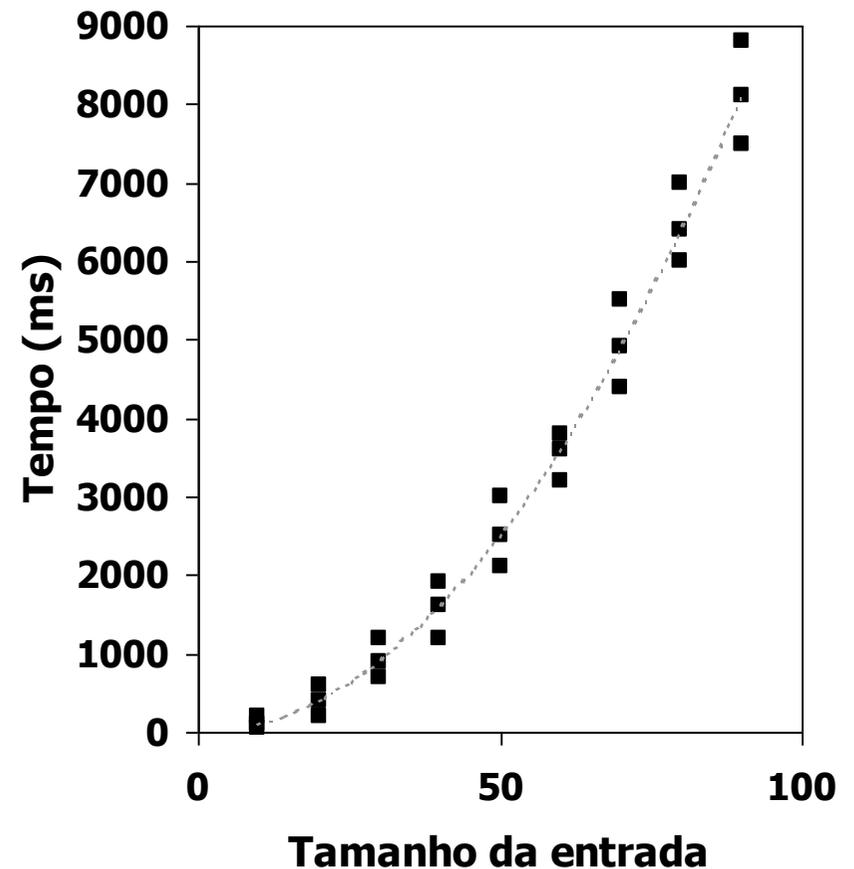


1) Ordem de funções

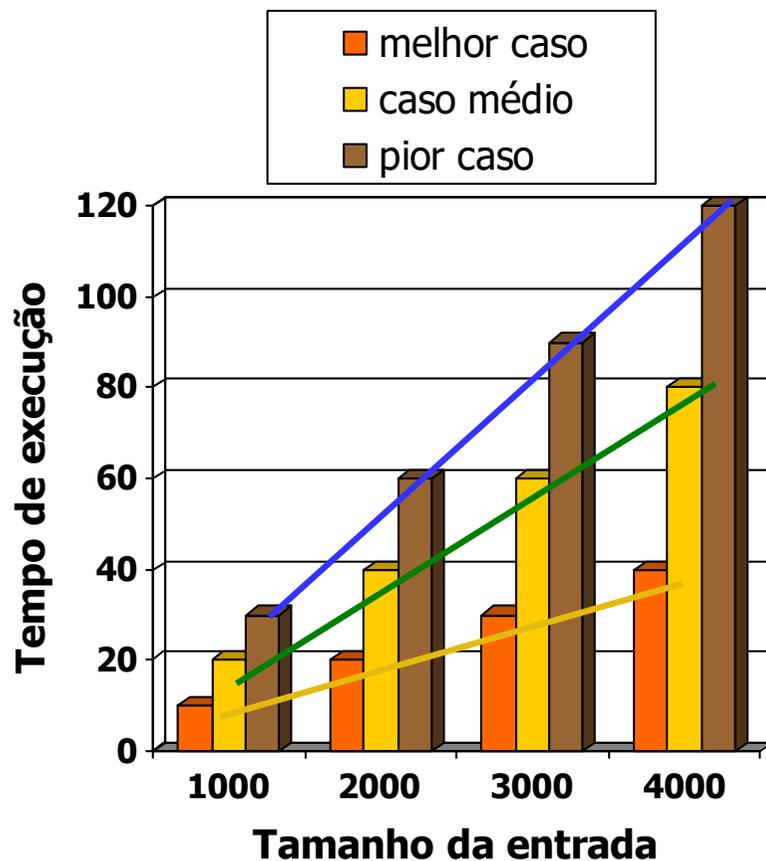
Notação O (*Big-Oh*) e similares

Resultados experimentais

- Para resolver um problema, implemente um determinado algoritmo.
- Execute esse programa com diversas instâncias do problema (entradas com *valores e tamanhos* variados).
- Meça o tempo real dessas execuções.
- Desenhe um gráfico com os resultados obtidos.



Tempo de execução



- O tempo de execução de um algoritmo varia (e normalmente cresce) com o tamanho da entrada do problema.
- Além disso, para instâncias de mesmo tamanho, também há variação no tempo de execução.
- Geralmente, o tempo médio é difícil de determinar.
- Costuma-se estudar os tempos máximos (pior caso):
 - é mais fácil de analisar;
 - é crucial para a qualidade das aplicações.
- Um possível inconveniente: quando o pior caso for uma exceção...

Análise teórica de complexidade

- Leva em consideração todas as possíveis entradas.
- Permite a avaliação do desempenho de um algoritmo, independentemente das características do *hardware* e do *software* utilizados.

Operações primitivas

- De modo geral, são as computações básicas realizadas por um algoritmo:
 - Atribuição de valor para uma variável
 - Comparação entre valores
 - Cálculo aritmético
 - Chamada de função
 - etc.
- Sua definição exata não é importante.
- Apesar de serem obviamente diferentes, são contabilizadas como *tempo unitário*.

Variáveis indexadas

- Exemplo: um vetor de inteiros
- `int A[1:10]` (Obs: índice *low* costuma ser 0)
- $@A[i] = @A + (i - low) \cdot \text{sizeof}(A[1])$

Endereço base

Quando *low* é 0, acesso torna-se mais rápido (economiza-se uma subtração)

Quase sempre é uma potência de 2, conhecida em tempo de compilação (nesse caso, usam-se *shifts* para acelerar o cálculo)

Cálculo em tempo constante

Exemplo de contagem

- O programa abaixo encontra o maior valor em um vetor de tamanho n:

```
int arrayMax (int A[], int n)
{
    currentMax = A[0];
    for (i=1; i<n; i++) {
        if (A[i] > currentMax)
            currentMax = A[i];
    }
    return currentMax;
}
```

1 atribuição e 1 indexação,
1 atribuição
repete n-1 vezes [1 teste,
1 indexação e 1 teste,
1 atribuição e 1 indexação
(no máximo),
1 incremento]
1 teste,
1 return

$$\text{Total: } 5 + (n-1).6 = 6n-1$$

- Inspeccionando o código, podemos calcular, em função de n, o *número máximo de operações primitivas executadas*.

Estimativa do tempo de execução

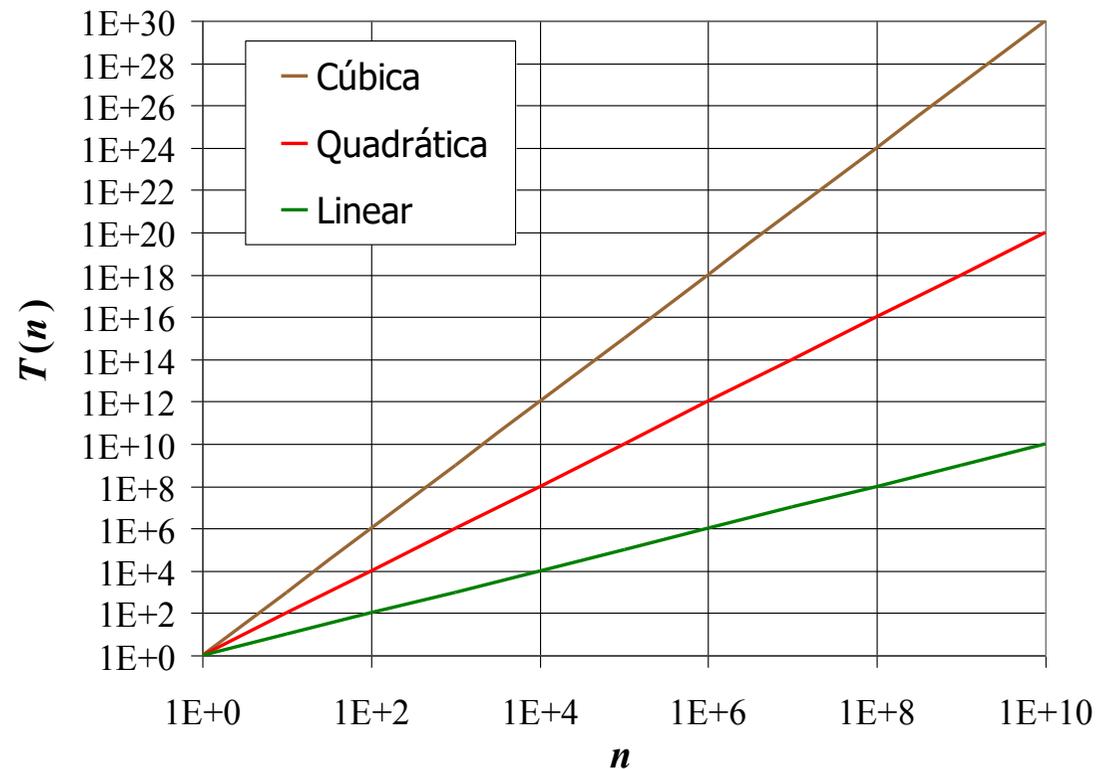
- No pior caso, o algoritmo `arrayMax` executa $6n-1$ operações primitivas.
- Definições:
 - a : tempo gasto na execução da operação primitiva mais rápida
 - b : tempo gasto na execução da operação primitiva mais lenta
- Seja $T(n)$ o tempo real de execução de pior caso de `arrayMax`.
- Portanto, $a.(6n-1) \leq T(n) \leq b.(6n-1)$
- $T(n)$ é limitado por duas funções lineares.

Taxa de crescimento do tempo de execução

- Alterações nos ambientes de *hardware* ou *software* :
 - afetam $T(n)$ apenas por um fator constante;
 - não alteram a taxa de crescimento de $T(n)$: continua linear!
- Portanto, a linearidade de $T(n)$ é uma propriedade intrínseca do algoritmo `arrayMax`.
- Cada algoritmo tem uma taxa de crescimento do pior caso que lhe é intrínseca.
- O que varia, de ambiente para ambiente, é somente o tempo absoluto de cada execução, que depende de fatores relacionados com o *hardware* e o *software* utilizados.

Taxas de crescimento

- Exemplos de taxas de crescimento:
 - Linear $\approx n$
 - Quadrática $\approx n^2$
 - Cúbica $\approx n^3$
- No gráfico *log-log* ao lado, a inclinação da reta corresponde à taxa de crescimento da função.



Taxas de crescimento

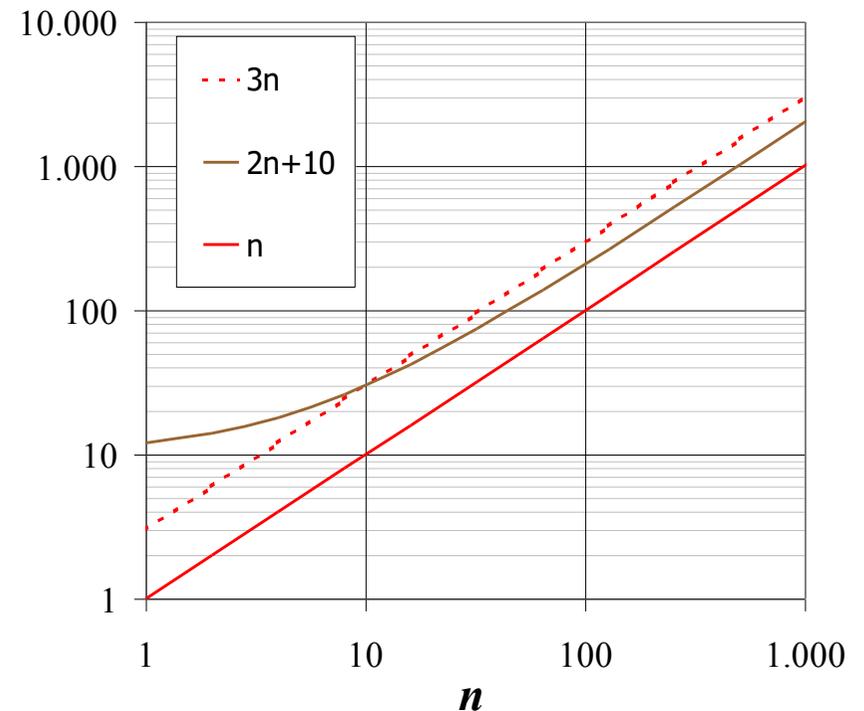
- A taxa de crescimento não é afetada por:
 - fatores constantes;
 - fatores de ordem mais baixa.
- Exemplos:
 - $10^2n + 10^5$: é uma função linear
 - $10^5n^2 + 10^8n$: é uma função quadrática
 - $10^{-9}n^3 + 10^{20}n^2$: é uma função cúbica

Notação O (*Big-Oh*)

- Dadas as funções $f(n)$ e $g(n)$, dizemos que $f(n)$ é $O(g(n))$ se existem duas constantes positivas c e n_0 tais que $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$

- Exemplo: $2n + 10$ é $O(n)$

- $2n + 10 \leq cn$
- $(c - 2)n \geq 10$
- $n \geq 10/(c - 2)$
- Uma possível escolha: $c = 3$ e $n_0 = 10$



Basta que essas constantes existam!

Lembretes sobre a notação O



Uma função $f(n)$ é $O(g(n))$ se, para todo n suficientemente grande, $f(n)$ não é maior que $c \cdot g(n)$, onde $c > 0$

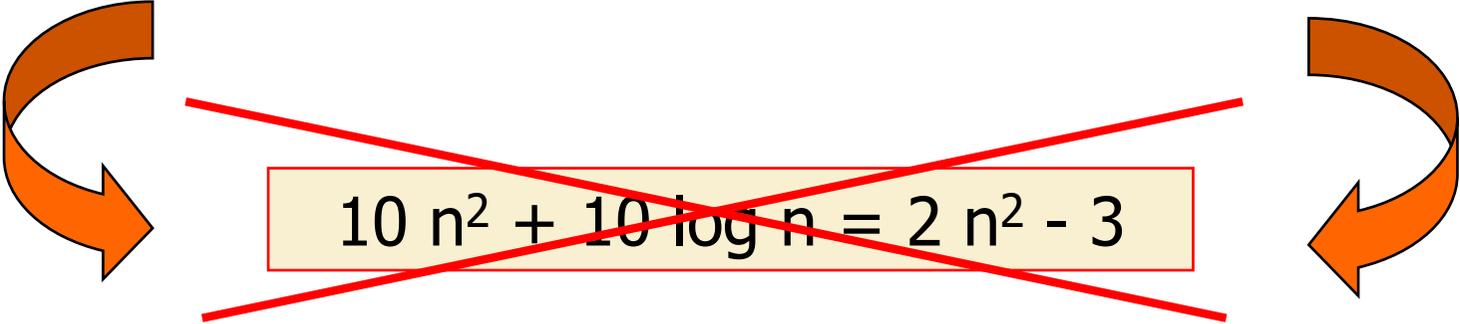
" $f(n)$ é $O(g(n))$ " ou " $f(n) = O(g(n))$ "
na realidade significam " $f(n) \in O(g(n))$ "

Abuso de linguagem

- No uso da notação O , costuma ocorrer um “abuso de linguagem”: o sinal de igualdade não tem o seu significado habitual.

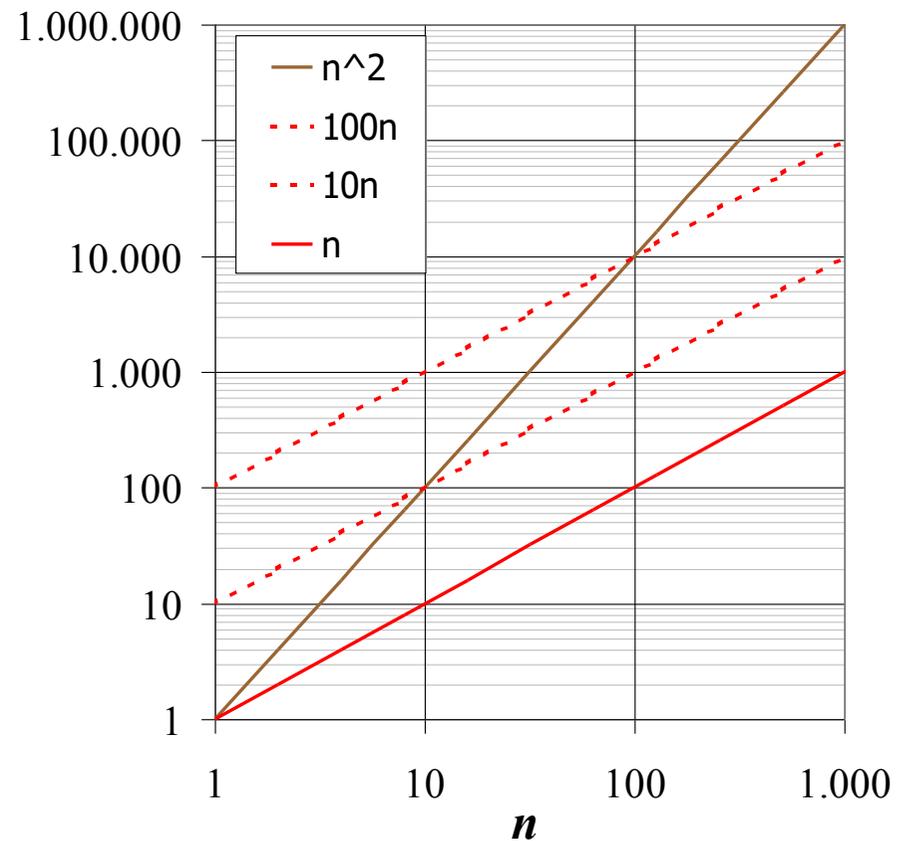
$$10 n^2 + 10 \log n = O(n^2)$$

$$2 n^2 - 3 = O(n^2)$$


$$10 n^2 + 10 \log n = 2 n^2 - 3$$

Outro exemplo

- n^2 não é $O(n)$
 - $n^2 \leq cn$
 - $n \leq c$
- A inequação acima não pode ser sempre satisfeita, pois c é uma constante e n não...
- Qualquer c escolhido poderia ser superado por n : basta escolher $n_0 = c+1$



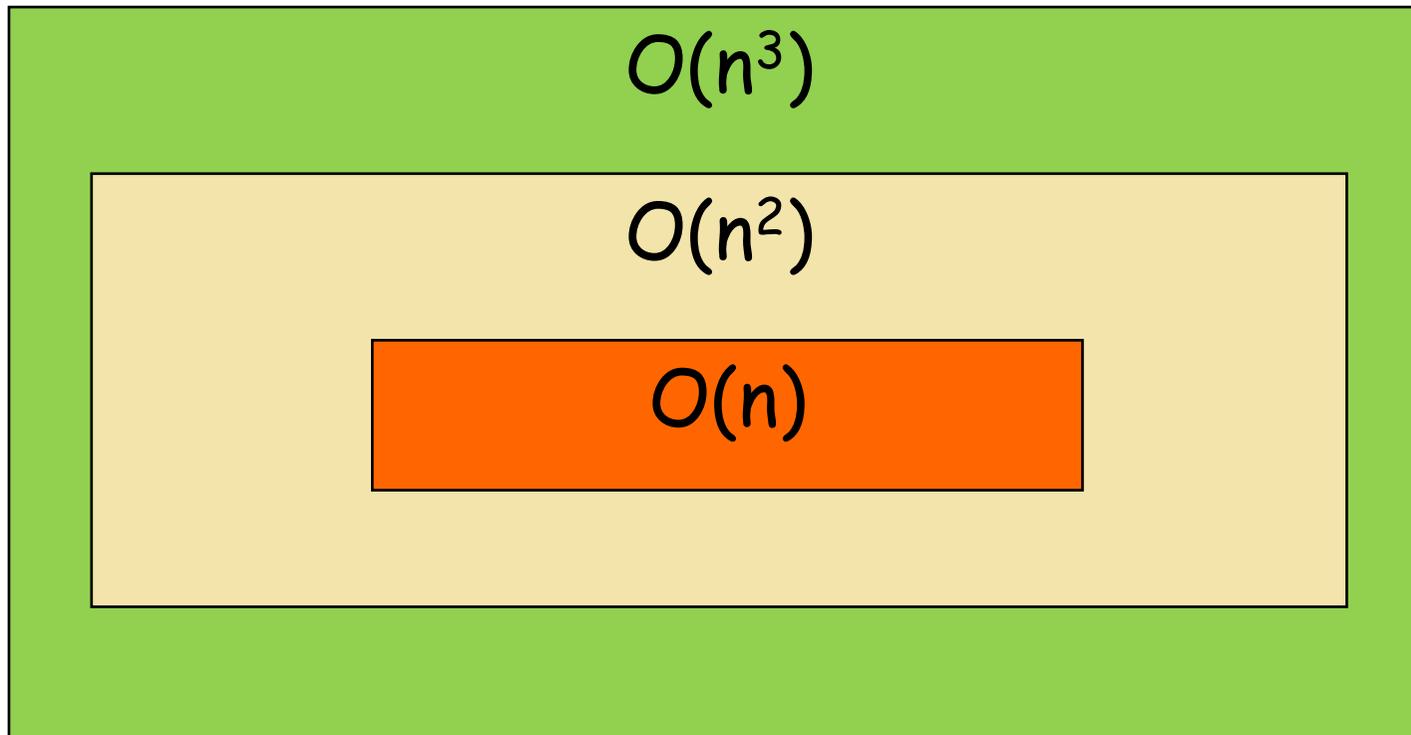
A notação O e a taxa de crescimento

- A notação O fornece um *limite superior* para a taxa de crescimento de uma determinada função.
- A afirmação " $f(n)$ é $O(g(n))$ " significa que a taxa de crescimento de $f(n)$ não é maior que a de $g(n)$.
- A notação O permite ordenar as funções de acordo com as suas correspondentes taxas de crescimento.

	$f(n)$ é $O(g(n))$?	$g(n)$ é $O(f(n))$?
Se $g(n)$ cresce mais que $f(n)$:	Sim	Não
Se $f(n)$ cresce mais que $g(n)$:	Não	Sim
Se $f(n)$ e $g(n)$ têm a mesma taxa:	Sim	Sim

Funções polinomiais

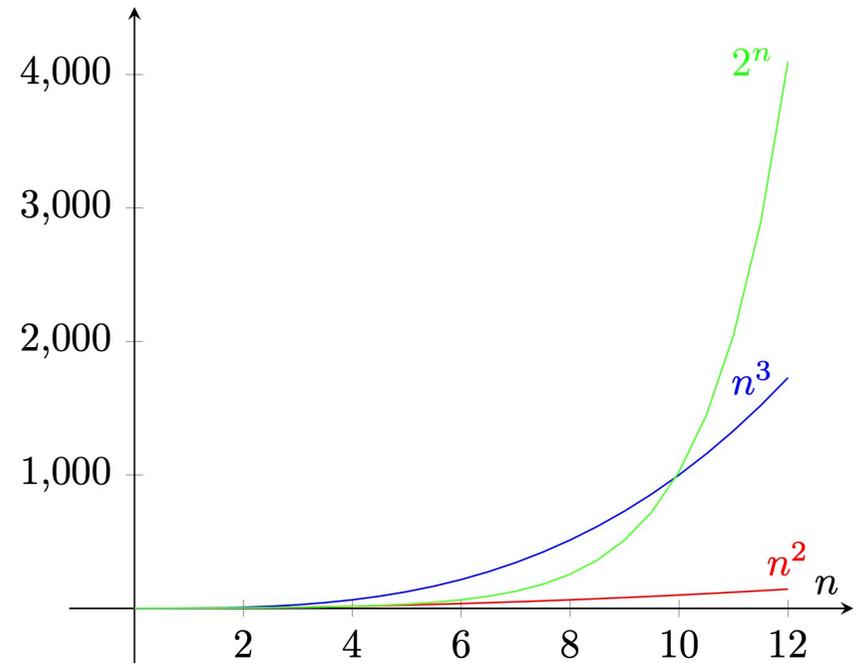
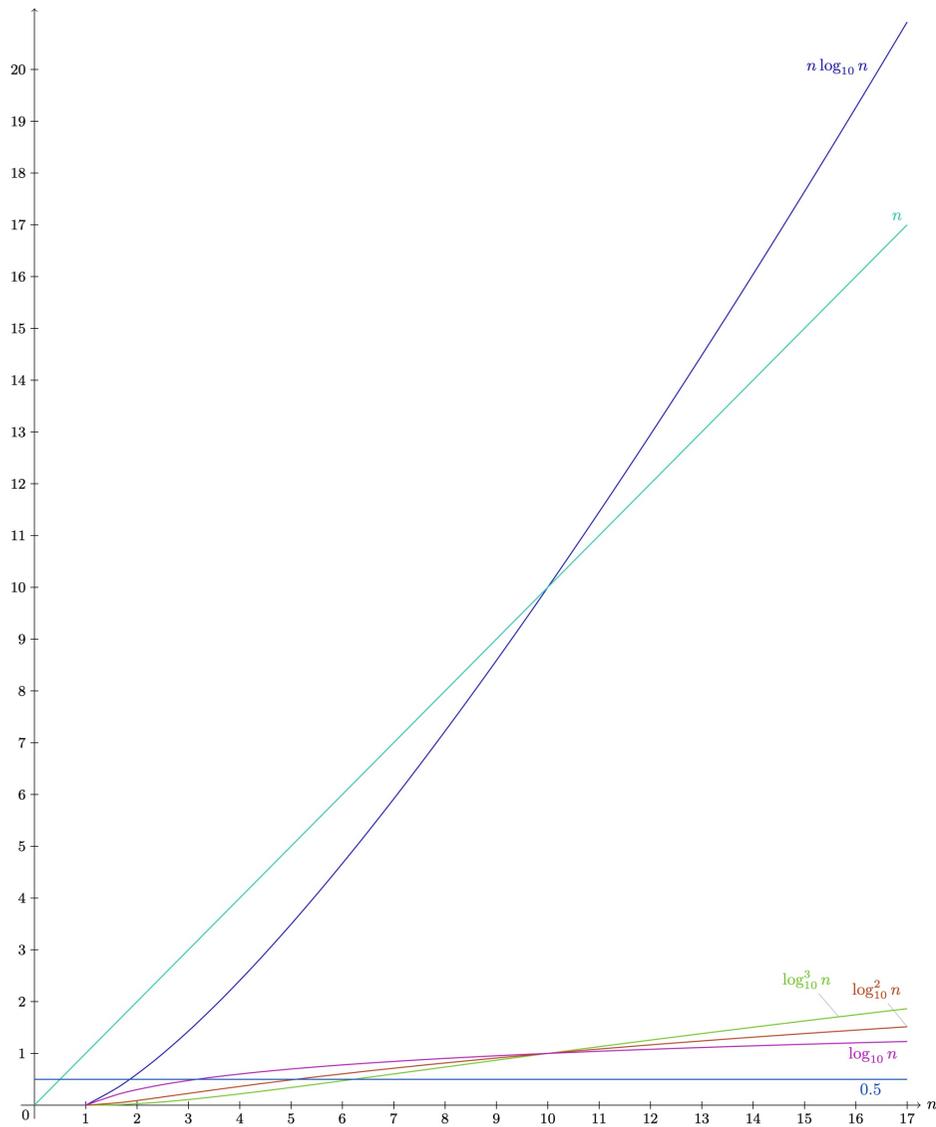
Com relação às funções polinomiais, é possível concluir que $O(n) \subset O(n^2) \subset O(n^3) \subset O(n^4) \subset O(n^5) \subset \dots$



Algumas dicas sobre a notação O

- No uso da notação O , consideramos apenas valores suficientemente grandes de n , ou seja, quando $n \rightarrow \infty$
 - Outro possível inconveniente: esses casos podem não ser viáveis na prática...
- Se $p(n)$ é um polinômio de grau k , então $p(n)$ é $O(n^k)$
 - Pode-se descartar seus termos de menor ordem, inclusive as constantes.
- Convém utilizar termos simples de menor ordem:
 - " $2n$ é $O(n)$ " é preferível a " $2n$ é $O(n^2)$ "
 - " $3n + 5$ é $O(n)$ " é preferível a " $3n + 5$ é $O(3n)$ "

Comparações entre algumas funções



Exemplos

$$6n^4 + 12n^3 + 12$$

$$\in O(n^4)$$

$$\in O(n^5)$$

$$\notin O(n^3)$$

$$3n^2 + 12n \cdot \log n$$

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

$$\in O(n^4)$$

$$\notin O(n \cdot \log n)$$

$$5n^2 + n(\log n)^2 + 12$$

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

$$\in O(n^3)$$

$$\notin O(n \cdot \log^9 n)$$

$$\log n + 4$$

$$\in O(\log n)$$

$$\in O(n)$$

$$\log^k n, k > 1$$

$$\in O(n)$$

$$\notin O(\log n)$$

Hierarquia entre funções

- A partir da notação O , é possível estabelecer uma hierarquia entre as funções:

Constante	$O(1)$
Logarítmica	$O(\log n)$
Linear	$O(n)$
$n \cdot \log n$	$O(n \cdot \log n)$
Quadrática	$O(n^2)$
Cúbica	$O(n^3)$
Polinomial	$O(n^k)$, com $k \geq 4$
Exponencial	$O(k^n)$, com $k > 1$

Maior
ordem



Evidentemente, as funções lineares, quadráticas e cúbicas também são polinomiais ...

Análise assintótica

- A análise assintótica descreve o tempo de pior caso dos algoritmos em notação O .
- Para realizar a análise assintótica de um algoritmo:
 - Calcula-se o número de operações primitivas executadas como função do tamanho da entrada.
 - Expressa-se esta função na notação O .
- Exemplo:
 - O algoritmo `arrayMax` executa no máximo $6n-1$ operações primitivas.
 - Dizemos que o tempo de pior caso do algoritmo `arrayMax` é $O(n)$, ou seja, sua complexidade de tempo é linear.

Notação O na análise de algoritmos

- Em comandos condicionais, o tempo total corresponde à execução do teste mais o tempo do bloco mais lento.

```
if (x == y)
    doSomething();
else
    doSomethingElse();
```

- Chamadas de funções: corresponde ao *tempo de execução da função chamada* (não ao tempo da chamada em si, que é descartado).

Notação O na análise de algoritmos

- Em comandos consecutivos, somam-se os tempos:

```
for (int x=1; x <= n; x++)  
    <operação primitiva qualquer>;
```

} $O(n)$

```
for (int x=1; x <= n; x++)  
    for (int y=1; y <= n; y++)  
        <operação primitiva qualquer>;
```

} $O(n^2)$

- Tempo total: $O(n) + O(n^2) = O(n^2)$
- E se o laço `for` mais interno começasse com $y=x$?
- Qual o tempo total gasto pelo laço abaixo?

```
for (int x=1, int y=1; y <= n; x++) {  
    <operação primitiva qualquer>;  
    if (x==n) { y++; x=1; }  
}
```

} $O(n^2)$

Exercícios



- Elabore um algoritmo paralelo para encontrar o maior valor presente em um vetor de n posições.
 - Dica: utilize n processadores.
- Qual a complexidade de tempo desse algoritmo?
- Haveria outros algoritmos mais eficientes ou que utilizassem menos processadores?
 - Dica: divida o vetor em blocos de tamanho $O(\log n)$ e utilize um processador em cada bloco.

Limites inferiores

- Enquanto a notação O fornece um limite superior (ou teto assintótico) para o crescimento das funções, também há outras notações que oferecem mais informações interessantes.
- Seja $\Omega(g(n))$ o conjunto de funções $f(n)$ para as quais existem constantes positivas c e n_0 tais que $f(n) \geq c \cdot g(n), \forall n \geq n_0$.
- A notação Ω fornece um limite inferior (ou piso assintótico) para o crescimento das funções.

Exemplos

- $f(n) = 12n^2 - 10 \in \Omega(1)$
- $f(n) = 12n^2 - 10 \in \Omega(n)$
- $f(n) = 12n^2 - 10 \in \Omega(n^2)$
- Entretanto, $f(n) = 12n^2 - 10 \notin \Omega(n^3)$

Notação Ω na prática

- Na notação Ω , convém utilizar a maior função possível:

É correto dizer que $f(n) = 3n^2 + 10$ é $\Omega(1)$,
mas representa pouca coisa sobre $f(n)$...

- Analogamente, $f(n) = \Omega(g(n))$ significa que $f(n) \in \Omega(g(n))$.

Limites inferiores e superiores

- Quando uma função $f(n)$ pertence simultaneamente a $O(g(n))$ e a $\Omega(g(n))$, dizemos que $f(n) \in \Theta(g(n))$.

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in (O(g(n)) \cap \Omega(g(n)))$$

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n)) \text{ e } g(n) \in O(f(n))$$

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \Omega(g(n)) \text{ e } g(n) \in \Omega(f(n))$$

Mais precisamente, $\Theta(g(n))$ é o conjunto de todas as funções $f(n)$ para as quais existem constantes positivas c_1 , c_2 e n_0 tais que:

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

Análise assintótica de funções

- Basicamente, podemos dizer que $f(n)$ é $\Theta(g(n))$ se e somente se:

$$\lim_{n \rightarrow \infty} (f(n)/g(n)) = c, \text{ onde } c > 0$$

- Por outro lado:
 - Se $\lim_{n \rightarrow \infty} (f(n)/g(n)) = 1$, dizemos que $f(n) \sim g(n)$
 - Se $\lim_{n \rightarrow \infty} (f(n)/g(n)) = 0$, dizemos que $f(n)$ é $o(g(n))$
 - Se $\lim_{n \rightarrow \infty} (f(n)/g(n)) = \infty$, dizemos que $f(n)$ é $w(g(n))$

Análise assintótica de funções

- É possível fazer uma *analogia* entre a comparação assintótica de duas funções f e g e a comparação de dois números reais a e b .

$$f(n) = O(g(n)) \quad \approx \quad a \leq b$$

$$f(n) = \Omega(g(n)) \quad \approx \quad a \geq b$$

$$f(n) = \Theta(g(n)) \quad \approx \quad a = b$$

$$f(n) = o(g(n)) \quad \approx \quad a < b$$

$$f(n) = \omega(g(n)) \quad \approx \quad a > b$$

Alguns exemplos

$$6n^4 + 12n^3 + 12$$

$$\in \Theta(n^4)$$

$$\in o(n^5)$$

$$\notin \omega(n^5)$$

$$3n^2 + 12n \cdot \log n$$

$$\in \Theta(n^2)$$

$$\in \Omega(n)$$

$$\notin O(n \cdot \log n)$$

$$5n^2 + n(\log n)^2 + 12$$

$$\in \Theta(n^2)$$

$$\in \omega(n \cdot \log^2 n)$$

$$\notin \Omega(n^3)$$

$$\log n + 4$$

$$\in \Theta(\log n)$$

$$\notin o(\log n)$$

$$\log^k n, k > 1$$

$$\notin \Theta(n)$$

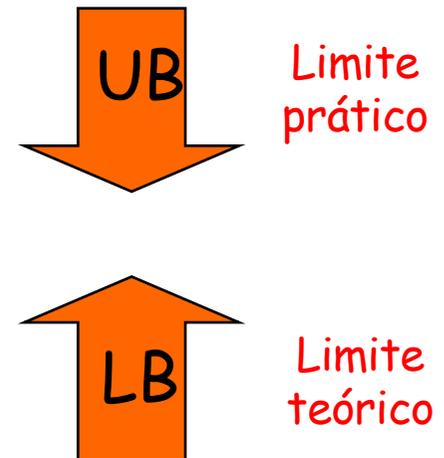
$$\in o(n)$$

Lower e upper bounds

- Dado um determinado problema P , chamamos de:

- $UB(P)$ ou *upper bound* de P : a complexidade do melhor algoritmo conhecido que o resolve.
- $LB(P)$ ou *lower bound* de P : a complexidade mínima necessária em qualquer de suas resoluções.

Deseja-se:



- Um problema P pode ser considerado computacionalmente resolvido se $UB(P) \in \Theta(LB(P))$

Conclusões finais

- Na análise de complexidade de um algoritmo, estamos mais interessados no seu **comportamento geral** que em outros detalhes (que dependem da máquina, do sistema operacional, da linguagem ou dos compiladores, etc.).
- Procura-se medir a complexidade de um algoritmo em função de um parâmetro do problema, que **geralmente é o tamanho da sua entrada**.
- O que se costuma considerar é o **comportamento assintótica do tempo de pior caso** dos algoritmos executados em máquinas que operam no tradicional **modelo de acesso aleatório**.