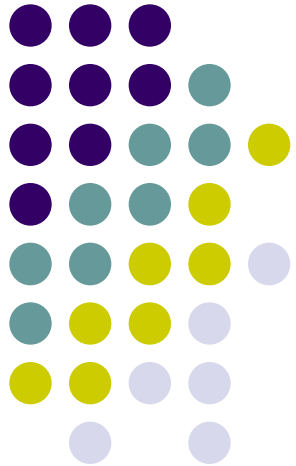
Análise de algoritmos

Introdução à Ciência da Computação 2

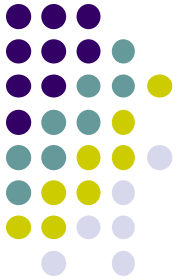
Baseado nos slides do Prof. Thiago A. S. Pardo



Exercício

- Seja f(x) = O(g(x)) e g(x) = O(h(x)).
  - Mostre que f(x) = O(h(x))

Solução



If $f(x) = O(g(x))$, then there are positive constants c_1 and n_0' such that

$$0 \leq f(n) \leq c_1\, g(n) \text{ for all } n \geq n_0',$$

and if $g(x) = O(h(x))$, then there are positive constants c_2 and n_0'' such that

$$0 \leq g(n) \leq c_2\, h(n) \text{ for all } n \geq n_0''.$$

Set $n_0 = \max(n_0', n_0'')$ and $c_3 = c_1\, c_2$. Then

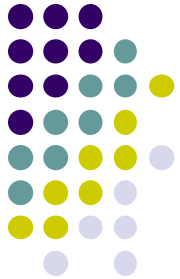
$$0 \leq f(n) \leq c_1\, g(n) \leq c_1\, c_2\, h(n) = c_3\, h(n) \text{ for all } n \geq n_0.$$

Thus $f(x) = O(h(x))$.

Exercício

- Mostrar que $x^4-23x^3+12x^2+15x-21 = \Theta(x^4)$

Solução



It is clear that when $x \geq 1$,

$$x^4 - 23x^3 + 12x^2 + 15x - 21 \leq x^4 + 12x^2 + 15x \leq x^4 + 12x^4 + 15x^4 = 28x^4.$$

Also,

$$x^4 - 23x^3 + 12x^2 + 15x - 21 \geq x^4 - 23x^3 - 21 \geq x^4 - 23x^3 - 21x^3 = x^4 - 44x^3 \geq \frac{1}{2}x^4,$$

whenever

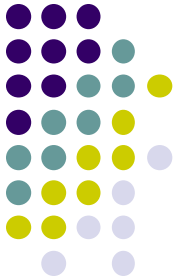
$$\frac{1}{2}x^4 \geq 44x^3 \Leftrightarrow x \geq 88.$$

Thus

$$\frac{1}{2}x^4 \leq x^4 - 23x^3 + 12x^2 + 15x - 21 \leq 28x^4, \text{ for all } x \geq 88.$$

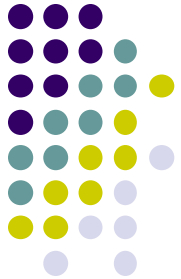
We have shown that $f(x) = x^4 - 23x^3 + 12x^2 + 15x - 21 = \Theta(x^4)$.

Taxas de crescimento



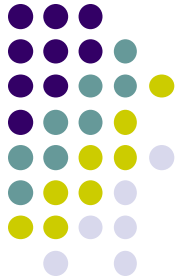
- Algumas regras
  - $\log^k n = O(n)$ para qualquer constante k, pois logaritmos crescem muito vagarosamente

Pergunta



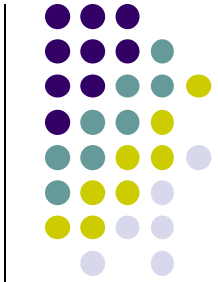
- Para qualquer algoritmo, pode-se dizer o que está abaixo?
  - $T(n) = O(\infty)$
  - $T(n) = \Omega(-\infty)$

Pergunta



- Para qualquer algoritmo, pode-se dizer o que está abaixo?
  - $T(n) = O(\infty)$
  - $T(n) = \Omega(-\infty)$
- Se sim, por que simplesmente não fazemos isso para todo algoritmo e pulamos para o próximo assunto da disciplina?

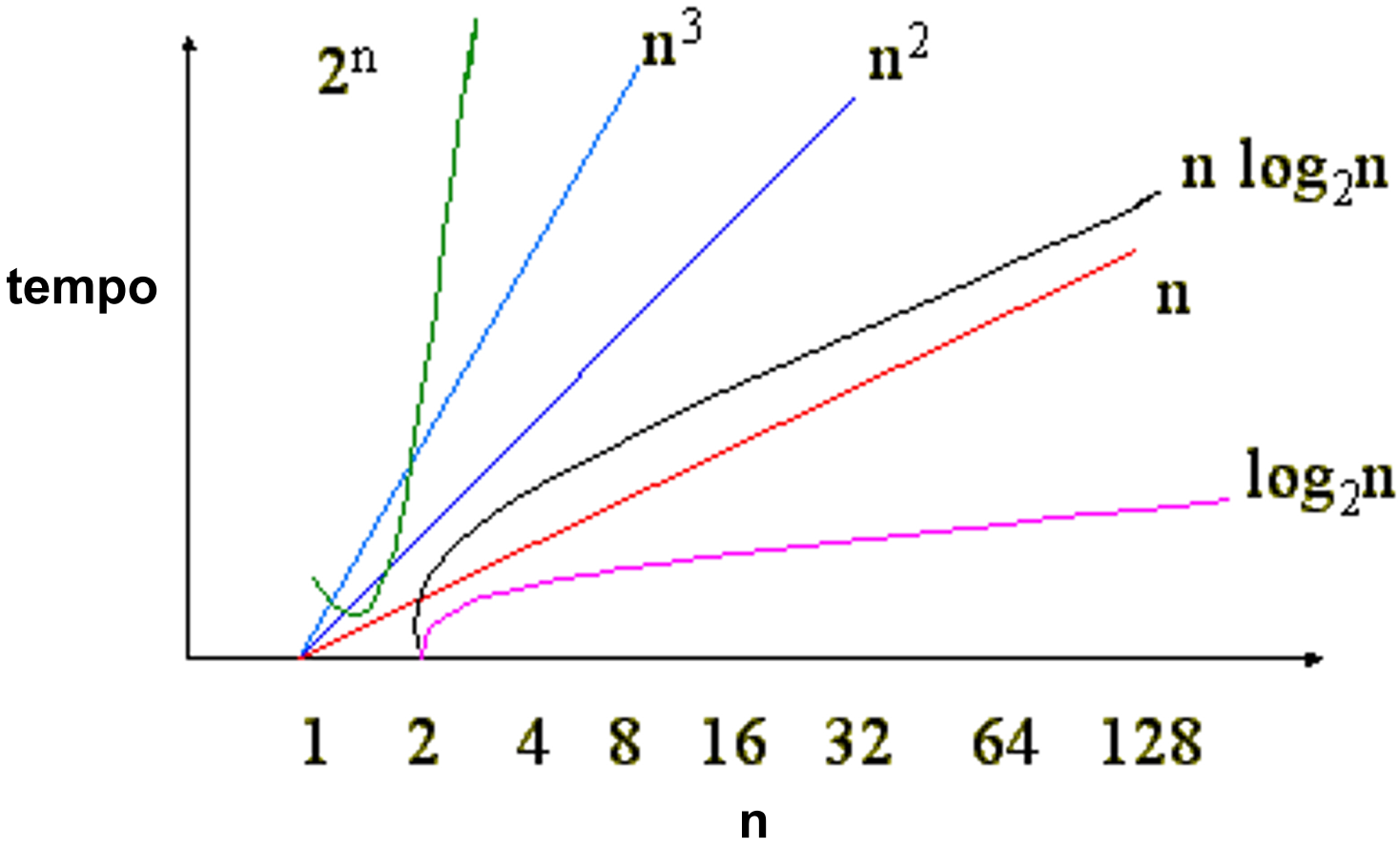
Funções e taxas de crescimento



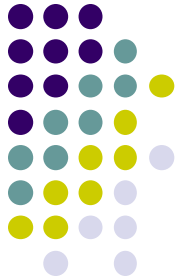
- As mais comuns

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

Funções e taxas de crescimento



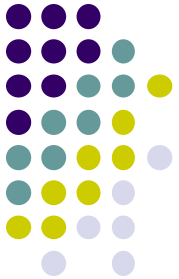
Taxas de crescimento



- Apesar de às vezes ser importante, não se costuma 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

- Não se diz que $T(n) = O(2n^2)$ ou que $T(n) = O(n^2+n)$
  - Diz-se apenas $T(n) = O(n^2)$

Exercício



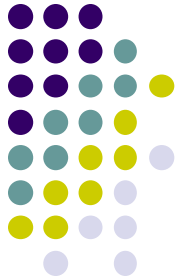
- Um algoritmo tradicional e muito utilizado é da ordem de $n^{1,5}$, enquanto um algoritmo novo proposto recentemente é da ordem de $n \log n$
  - $f(n)=n^{1,5}$
  - $g(n)=n \log n$

- Qual algoritmo você adotaria na empresa que está fundando?
  - Lembre-se que a eficiência desse algoritmo pode determinar o sucesso ou o fracasso de sua empresa

Exercício

- Uma possível solução
  - $f(n) = n^{1,5}$ □ $n^{1,5}/n = n^{0,5}$ □ $(n^{0,5})^2 = n$
  - $g(n) = n \log n$ □ $(n \log n)/n = \log n$ □ $(\log n)^2 = \log^2 n$
    - Como n cresce mais rapidamente do que qualquer potência de log, temos que o algoritmo novo é mais eficiente e, portanto, deve ser o adotado pela empresa no momento

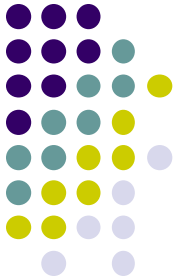
Análise de algoritmos



- Para proceder a uma análise de algoritmos e determinar as taxas de crescimento, necessitamos de um modelo de computador e das operações que executa

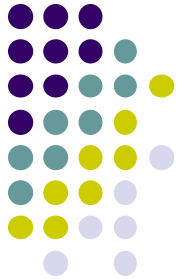
- Assume-se o uso de um computador tradicional, em que as instruções de um programa são executadas sequencialmente
  - Com memória infinita, por simplicidade

Análise de algoritmos



- Repertório de instruções simples: soma, multiplicação, comparação, atribuição, etc.
  - Por simplicidade e viabilidade da análise, assume-se que cada instrução demora exatamente uma unidade de tempo para ser executada
    - Obviamente, em situações reais, isso pode não ser verdade: a leitura de um dado em disco pode demorar mais do que uma soma
  - Operações complexas, como inversão de matrizes e ordenação de valores, não são realizadas em uma única unidade de tempo, obviamente: devem ser analisadas em partes

Análise de algoritmos

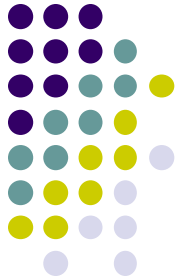


- Considera-se somente o algoritmo e suas entradas (de tamanho n)

- Para uma entrada de tamanho n, pode-se calcular $T_{melhor}(n)$, $T_{média}(n)$ e $T_{pior}(n)$, ou seja, o melhor tempo de execução, o tempo médio e o pior, respectivamente
  - Obviamente, $T_{melhor}(n) \leq T_{média}(n) \leq T_{pior}(n)$

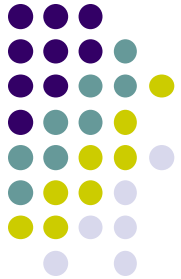
- Atenção: para mais de uma entrada, essas funções teriam mais de um argumento

Análise de algoritmos



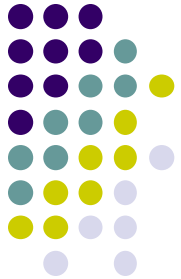
- Geralmente, utiliza-se somente a análise do pior caso $T_{pior}(n)$, pois ela fornece os limites para todas as entradas, incluindo particularmente as entradas ruins
  - Logicamente, muitas vezes, o tempo médio pode ser útil, principalmente em sistemas executados rotineiramente
    - Por exemplo: em um sistema de cadastro de alunos como usuários de uma biblioteca, o trabalho difícil de cadastrar uma quantidade enorme de pessoas é feito somente uma vez; depois, cadastros são feitos de vez em quando apenas
  - Dá mais trabalho calcular o tempo médio
  - O melhor tempo não tem muita utilidade

Pergunta



- Idealmente, para um algoritmo qualquer de ordenação de vetores com n elementos
  - Qual a configuração do vetor que você imagina que provavelmente geraria o melhor tempo de execução?
  - E qual geraria o pior tempo?

Exemplo



- Soma da subseqüência máxima
  - Dada uma seqüência de inteiros (possivelmente negativos) a_1, a_2, ..., a_n, encontre o valor da máxima soma de quaisquer números de elementos consecutivos; se todos os inteiros forem negativos, o algoritmo deve retornar 0 como resultado da maior soma
  - Por exemplo, para a entrada -2, 11, -4, 13, -5 e -2, a resposta é 20 (soma de a_1 a a_3)

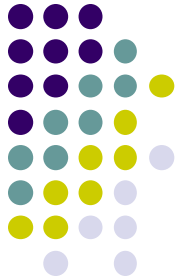
Soma da subseqüência máxima

- Há muitos algoritmos propostos para resolver esse problema
- Alguns são mostrados abaixo juntamente com seus tempos de execução

Algoritmo	1	2	3	4
Tempo	$O(n^3)$	$O(n^2)$	$O(n \log n)$	$O(n)$
Tamanho da entrada				
n =10	0.00103	0.00045	0.00066	0.00034
n =100	0.47015	0.01112	0.00486	0.00063
n =1.000	448.77	1.1233	0.05843	0.00333
n =10.000	ND*	111.13	0.68631	0.03042
n =100.000	ND	ND	8.0113	0.29832

*ND = Não Disponível

Soma da subseqüência máxima



- Deve-se notar que
  - Para entradas pequenas, todas as implementações rodam num piscar de olhos
    - Portanto, se somente entradas pequenas são esperadas, não devemos gastar nosso tempo para projetar melhores algoritmos
  - Para entradas grandes, o melhor algoritmo é o 4
  - Os tempos não incluem o tempo requerido para leitura dos dados de entrada
    - Para o algoritmo 4, o tempo de leitura é provavelmente maior do que o tempo para resolver o problema: característica típica de algoritmos eficientes

Taxas de crescimento

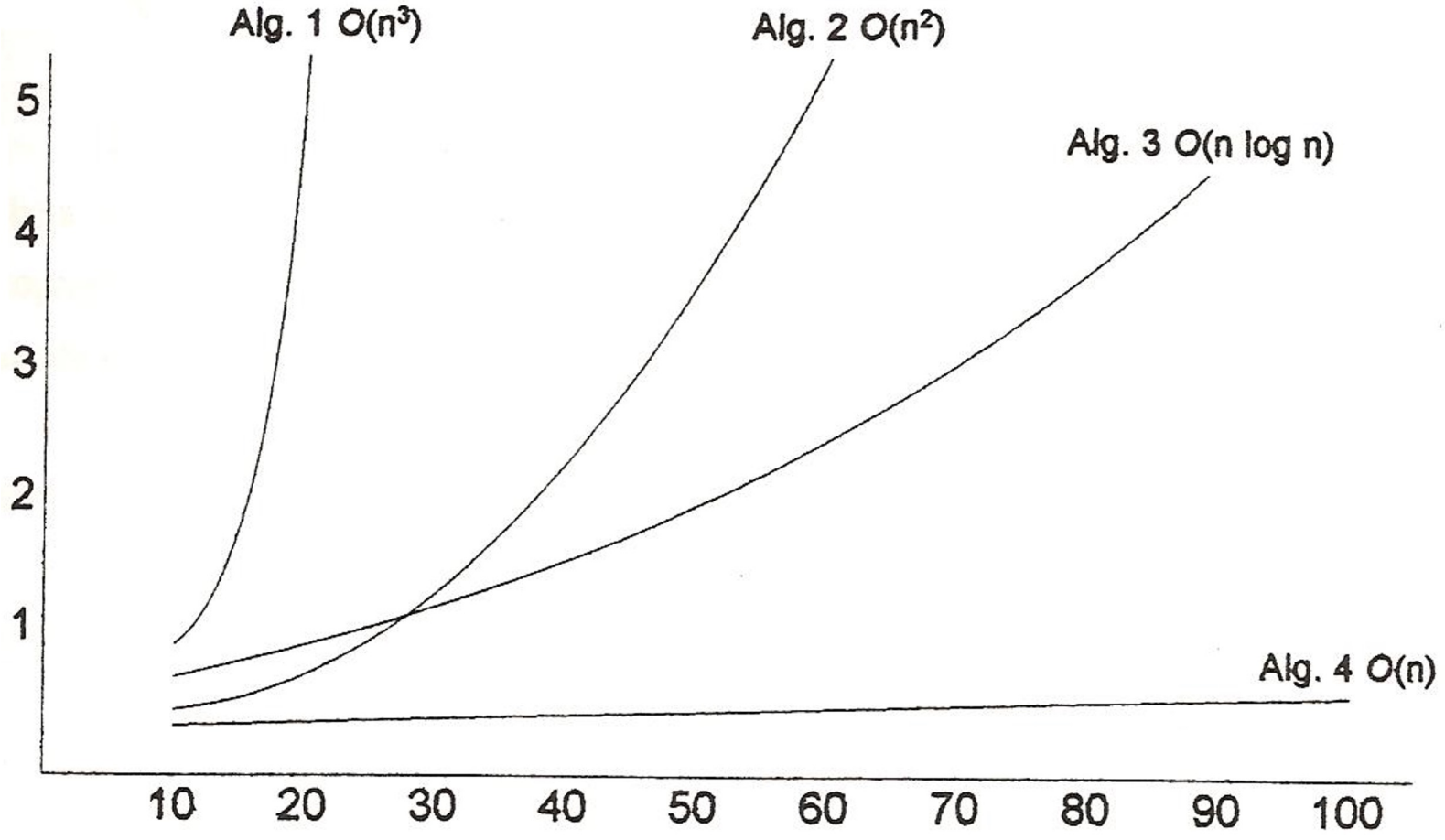


Gráfico (*n* x milisegundos) das taxas de crescimento dos 4 algoritmos com entradas entre 10 e 100.

Taxas de crescimento

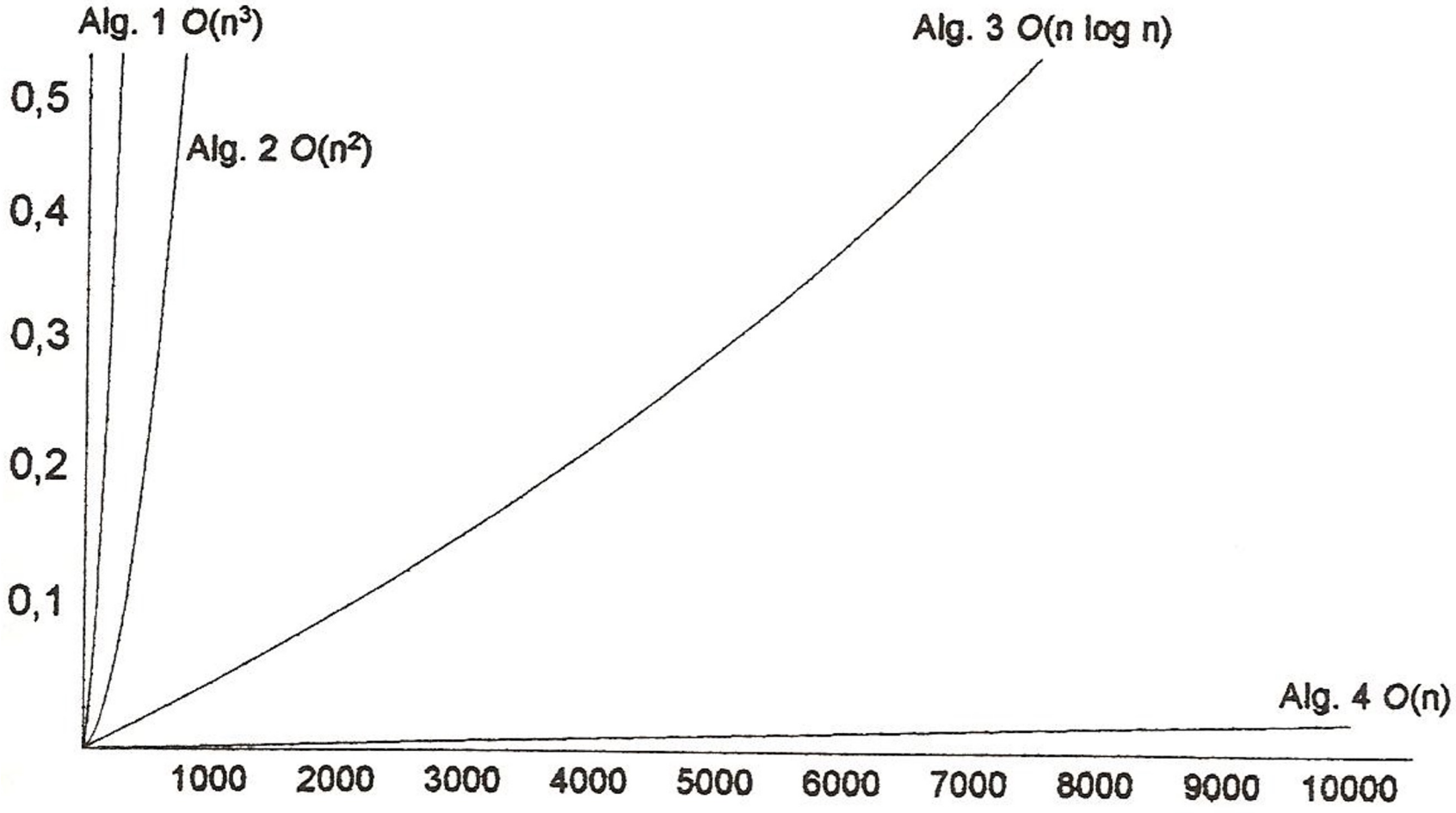


Gráfico (*n* x segundos) dos 4 algoritmos para entradas maiores