



SCC-201 - Capítulo 1

Análise de Algoritmos - Parte 1

João Luís Garcia Rosa¹

¹Departamento de Ciências de Computação
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
<http://www.icmc.usp.br/~joaoluis>

2016

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 Análise assintótica
 - Análise assintótica
 - Conceitos de matemática
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 Análise assintótica
 - Análise assintótica
 - Conceitos de matemática
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

Algoritmo: noção geral

- **Algoritmo**¹ é um conjunto de instruções que devem ser seguidas para solucionar um determinado problema.
- Cormen *et al.* [1]:
 - Qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores de **entrada** e produz algum valor ou conjunto de valores de **saída**;
 - Ferramenta para resolver um problema computacional bem especificado;
 - Assim como o hardware de um computador, constitui uma tecnologia, pois o desempenho total do sistema depende da escolha de um algoritmo eficiente tanto quanto da escolha de um hardware rápido.

¹ A palavra “algoritmo” vem do nome de um matemático persa (825 d.C.), [Abu Ja’far Mohammed ibn Musa al Khwarizmi](#).

Algoritmo: noção geral

- Cormen *et al.* [1]:
 - Deseja-se que um algoritmo termine e seja correto.
- Perguntas:
 - Mas um algoritmo correto vai terminar, não vai?
 - A afirmação está redundante?

Eficiência e Problemas Difíceis

- Além de um algoritmo correto, busca-se também um algoritmo **eficiente** para resolver um determinado problema
- Pergunta: como 'medir' eficiência de um algoritmo?
- Obs. Existem problemas para os quais não se conhece nenhum algoritmo eficiente para obter a solução: $\mathcal{NP}$ -completos.

Recursos de um algoritmo

- Uma vez que um algoritmo está pronto/disponível, é importante determinar os **recursos necessários** para sua execução:
 - Tempo
 - Memória
- Qual o principal quesito? Por que?

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - **Análise de Algoritmos**
 - Eficiência de algoritmos
- 2 Análise assintótica
 - Análise assintótica
 - Conceitos de matemática
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

Análise de algoritmos

- Um algoritmo que soluciona um determinado problema, mas requer o processamento de **um ano**, não deve ser usado.
- O que dizer de uma afirmação como a abaixo?
“Desenvolvi um novo algoritmo chamado TripleX que leva 14,2 segundos para processar 1.000 números, enquanto o método SimpleX leva 42,1 segundos.”
- Você trocaria o *SimpleX* que roda em sua empresa pelo *TripleX*?

Análise de algoritmos

- A afirmação tem que ser examinada, pois há diversos fatores envolvidos:
 - Características da máquina em que o algoritmo foi testado:
 - Quantidade de memória.
 - Linguagem de programação:
 - Compilada vs. interpretada,
 - Alto vs. baixo nível.
 - Implementação pouco cuidadosa do algoritmo *SimpleX* vs. “super” implementação do algoritmo *TripleX*.
 - Quantidade de dados processados:
 - Se o *TripleX* é mais rápido para processar 1.000 números, ele também é mais rápido para processar quantidades maiores de números, certo?

Análise de algoritmos

- A comunidade de computação começou a pesquisar formas de comparar algoritmos de forma independente de
 - Hardware,
 - Linguagem de programação,
 - Habilidade do programador.
- Portanto, quer-se comparar **algoritmos** e não **programas**:
 - Área conhecida como “análise/complexidade de algoritmos”.

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 Análise assintótica
 - Análise assintótica
 - Conceitos de matemática
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

Eficiência de algoritmos

- Sabe-se que:
 - Processar 10.000 números leva mais tempo do que 1.000 números,
 - Cadastrar 10 pessoas em um sistema leva mais tempo do que cadastrar 5,
 - Etc.
- Então, pode ser uma boa idéia estimar a **eficiência** de um algoritmo em função do **tamanho** do problema:
 - Em geral, assume-se que n é o tamanho do problema, ou número de elementos que serão processados,
 - E calcula-se o número de operações que serão realizadas sobre os n elementos.

Eficiência de algoritmos

- O melhor algoritmo é aquele que requer menos operações sobre a entrada, pois é o mais rápido:
 - O tempo de execução do algoritmo pode variar em diferentes máquinas, mas o número de operações é uma boa medida de desempenho de um algoritmo.
- De que operações estamos falando?
- Toda operação leva o mesmo tempo?

Exemplo: *TripleX* vs. *SimpleX*

- *TripleX*: para uma entrada de tamanho n , o algoritmo realiza $n^2 + n$ operações:
 - Pensando em termos de função: $f(n) = n^2 + n$.
- *SimpleX*: para uma entrada de tamanho n , o algoritmo realiza $1.000n$ operações:
 - $g(n) = 1.000n$.

Exemplo: *TripleX* vs. *SimpleX*

- Faça os cálculos do desempenho de cada algoritmo para cada tamanho de entrada:

tamanho da entrada n	1	10	100	1.000	10.000
$f(n) = n^2 + n$					
$g(n) = 1.000n$					

Exemplo: *TripleX* vs. *SimpleX*

- Faça os cálculos do desempenho de cada algoritmo para cada tamanho de entrada:

tamanho da entrada n	1	10	100	1.000	10.000
$f(n) = n^2 + n$	2	110	10.100	1.001.000	100.010.000
$g(n) = 1.000n$	1.000	10.000	100.000	1.000.000	10.000.000

- A partir de $n = 1.000$, $f(n)$ mantém-se maior e cada vez mais distante de $g(n)$:
 - Diz-se que $f(n)$ cresce mais rápido do que $g(n)$.

Sumário

1

Algoritmo [2]

- Algoritmo
- Análise de Algoritmos
- Eficiência de algoritmos

2

Análise assintótica

- **Análise assintótica**
- Conceitos de matemática
- Notações

3

Taxas de crescimento

- Taxas de crescimento
- Análise de Algoritmos
- Cálculo do tempo de execução

Análise assintótica

- Devemos nos preocupar com a eficiência de algoritmos quando o tamanho de n for **grande**.
- *Definição*: a **eficiência assintótica** de um algoritmo descreve a sua eficiência relativa quando n torna-se grande.
- Portanto, para comparar 2 algoritmos, determinam-se as taxas de crescimento de cada um: o algoritmo com menor taxa de crescimento rodará mais rápido quando o tamanho do problema for grande.

Análise assintótica

- **Atenção:**

- Algumas funções podem não crescer com o valor de n :
 - Quais?
- Também se pode aplicar os conceitos de análise assintótica para a quantidade de memória usada por um algoritmo:
 - Mas não é tão útil, pois é difícil estimar os detalhes exatos do uso de memória e o impacto disso.

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 **Análise assintótica**
 - Análise assintótica
 - **Conceitos de matemática**
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

Relembrando um pouco de matemática...

- Expoentes:

- $x^a x^b = x^{a+b}$
- $x^a / x^b = x^{a-b}$
- $(x^a)^b = x^{ab}$
- $x^n + x^n = 2x^n$ (diferente de x^{2n})
- $2^n + 2^n = 2^{n+1}$

- Logaritmos (usaremos a base 2, a menos que seja dito o contrário):

- $x^a = b \Rightarrow \log_x b = a$
- $\log_a b = \log_c b / \log_c a$, se $c > 0$
- $\log ab = \log a + \log b$
- $\log a/b = \log a - \log b$
- $\log(a^b) = b \log a$
- E o mais importante:
 - $\log x < x$ para todo $x > 0$.

Função exponencial vs. logarítmica

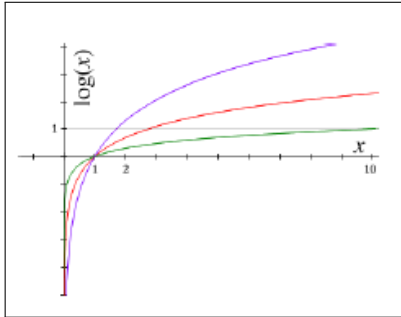


Figure : Exemplos de logaritmos para várias bases.



Figure : Na palma da mão direita.

Relembrando um pouco de matemática...

- Séries:

$$\sum_{i=0}^n 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$$

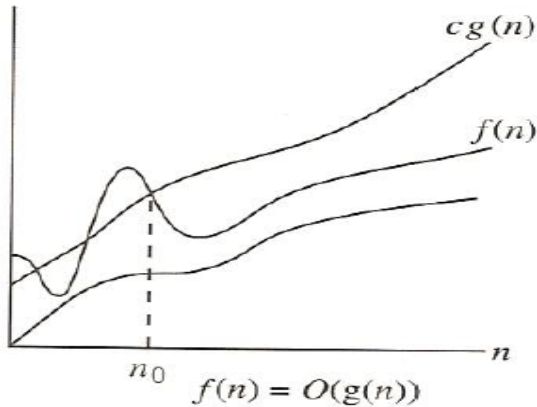
Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 **Análise assintótica**
 - Análise assintótica
 - Conceitos de matemática
 - **Notações**
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

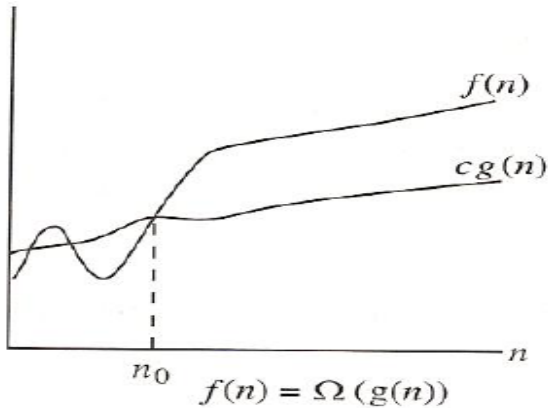
Algumas notações

- Dadas duas funções, $f(n)$ e $g(n)$,
 - diz-se que $f(n)$ é **da ordem de (big-oh)** $g(n)$ ou que $f(n)$ é $\mathcal{O}(g(n))$, se existirem constantes c e n_0 tais que $f(n) \leq c * g(n)$ para todo $n \geq n_0$.
 - A taxa de crescimento de $f(n)$ é menor ou igual à taxa de $g(n)$.
 - diz-se que $f(n)$ é **ômega** $g(n)$ ou que $f(n) = \Omega(g(n))$, se existirem constantes c e n_0 tais que $f(n) \geq c * g(n)$ para todo $n \geq n_0$.
 - A taxa de crescimento de $f(n)$ é maior ou igual à taxa de $g(n)$.
 - diz-se que $f(n)$ é **theta** $g(n)$ ou que $f(n) = \Theta(g(n))$, se e somente se $f(n) = \mathcal{O}(g(n))$ e $f(n) = \Omega(g(n))$.
 - A taxa de crescimento de $f(n)$ é igual à taxa de $g(n)$.
 - diz-se que $f(n)$ é **little-oh** $g(n)$ ou que $f(n) = o(g(n))$, se e somente se $f(n) = \mathcal{O}(g(n))$ e $f(n) \neq \Theta(g(n))$.
 - A taxa de crescimento de $f(n)$ é menor do que a taxa de $g(n)$.

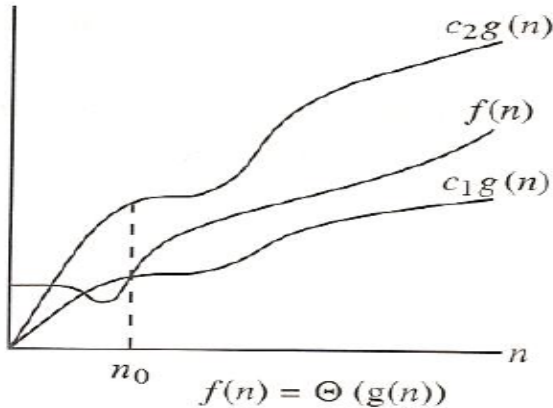
Algumas notações



Algumas notações



Algumas notações



Algumas considerações

- O uso das notações permite **comparar a taxa de crescimento** das funções correspondentes aos algoritmos:
 - **Não faz sentido comparar pontos isolados das funções**, já que podem não corresponder ao comportamento assintótico.
- Ao dizer que $g(n) = \mathcal{O}(f(n))$, garante-se que $g(n)$ cresce numa taxa não maior do que $f(n)$, ou seja, $f(n)$ é seu limite superior.
- Ao dizer que $f(n) = \Omega(g(n))$, tem-se que $g(n)$ é o limite inferior de $f(n)$.

Exemplo

- Para 2 algoritmos quaisquer, considere as funções de eficiência correspondentes $1.000n$ e n^2 :
 - A primeira é maior do que a segunda para valores pequenos de n ,
 - A segunda cresce mais rapidamente e finalmente será uma função maior, sendo que o ponto de mudança é $n = 1.000$,
 - Segundo as notações anteriores, se existe um ponto n_0 a partir do qual $c * f(n)$ é sempre pelo menos tão grande quanto $g(n)$, então, ignorados os fatores constantes $f(n)$ é pelo menos tão grande quanto $g(n)$:
 - No nosso caso, $g(n) = 1.000n$, $f(n) = n^2$, $n_0 = 1.000$ e $c = 1$ (ou, ainda, $n_0 = 10$ e $c = 100$): Dizemos que $1.000n = \mathcal{O}(n^2)$.

Outros exemplos

- 1 A função n^3 cresce mais rapidamente que n^2 :
 - $n^2 = \mathcal{O}(n^3)$
 - $n^3 = \Omega(n^2)$
- 2 Se $f(n) = n^2$ e $g(n) = 2n^2$, então essas duas funções têm taxas de crescimento iguais:
 - Portanto, $f(n) = \mathcal{O}(g(n))$ e $f(n) = \Omega(g(n))$.

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 Análise assintótica
 - Análise assintótica
 - Conceitos de matemática
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - Análise de Algoritmos
 - Cálculo do tempo de execução

Taxas de crescimento

- Algumas regras:

- Se $T_1(n) = \mathcal{O}(f(n))$ e $T_2(n) = \mathcal{O}(g(n))$, então:

- $T_1(n) + T_2(n) = \max(\mathcal{O}(f(n)), \mathcal{O}(g(n)))$.
- $T_1(n) * T_2(n) = \mathcal{O}(f(n) * g(n))$.

- Se $T(x)$ é um polinômio de grau n , então:

- $T(x) = \Theta(x^n)$.

- Relembrando: um polinômio de grau n é uma função que possui a forma abaixo:

$$f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + a_1 \cdot x + a_0$$

seguindo a seguinte classificação em função do grau:

- Grau 0: polinômio constante
- Grau 1: função afim (polinômio linear, caso $a_0 = 0$)
- Grau 2: polinômio quadrático
- Grau 3: polinômio cúbico
- $\log^k n = \mathcal{O}(n)$ para qualquer constante k , pois logaritmos crescem muito vagarosamente.

Funções e taxas de crescimento

- As mais comuns:

c	constante
$\log n$	logarítmica
$\log^2 n$	logarítmica ao quadrado
n	linear
$n \log n$	quadrática
n^2	
n^3	
2^n	exponencial
a^n	

Funções e taxas de crescimento

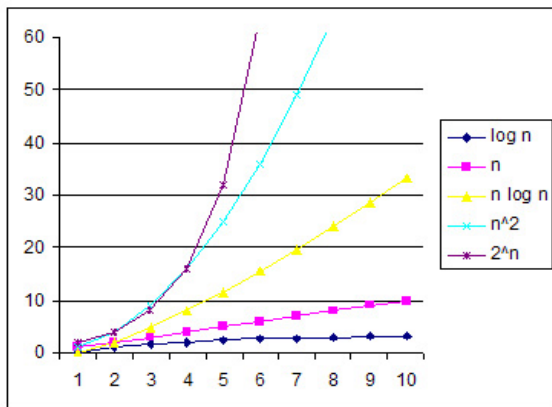


Figure : Crescimentos de algumas funções.

Taxas de crescimento

- Apesar de às vezes ser importante, não é comum 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 $T(n) = \mathcal{O}(2n^2)$ ou que $T(n) = \mathcal{O}(n^2 + n)$:
 - Diz-se apenas $T(n) = \mathcal{O}(n^2)$.

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:

$$f(n) = n^{1,5} \quad \Rightarrow \quad \frac{n^{1,5}}{n} = n^{0,5} \quad \Rightarrow \quad (n^{0,5})^2 = n$$

$$g(n) = n \log n \quad \Rightarrow \quad \frac{(n \log n)}{n} = \log n \quad \Rightarrow \quad (\log n)^2 = \log^2 n$$

- 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.

Sumário

- 1 Algoritmo [2]
 - Algoritmo
 - Análise de Algoritmos
 - Eficiência de algoritmos
- 2 Análise assintótica
 - Análise assintótica
 - Conceitos de matemática
 - Notações
- 3 Taxas de crescimento
 - Taxas de crescimento
 - **Análise de Algoritmos**
 - Cálculo do tempo de execução

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 $T_{\text{melhor}}(n)$, $T_{\text{media}}(n)$ e $T_{\text{pior}}(n)$, ou seja, o **melhor** tempo de execução, o tempo **médio** e o **pior**, respectivamente:
 - Obviamente, $T_{\text{melhor}}(n) \leq T_{\text{media}}(n) \leq T_{\text{pior}}(n)$.
- 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.

Análise de algoritmos

- Idealmente, para um algoritmo qualquer de ordenação de vetores com n elementos:
 - Qual a configuração do vetor que você imagina que provavelmente resultaria no melhor tempo de execução?
 - E qual resultaria no pior tempo?
- Exemplo:
 - Soma da subsequência máxima:
 - Dada uma sequência de inteiros (possivelmente negativos) $a_1, a_2, \dots, a_n$, 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_2 a a_4).

Soma da subsequência máxima

- Há muitos algoritmos propostos para resolver esse problema:
 - Alguns são mostrados abaixo juntamente com seus tempos de execução (n é o tamanho da entrada):

algoritmo	1	2	3	4
tempo	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
$n = 10$	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 ²	111,13	0,68631	0,03042
$n = 100.000$	ND	ND	8,0113	0,29832

²Não Disponível.

Soma da subsequência 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.

The graph illustrates the relative efficiency of four algorithms as the input size n increases. The x-axis represents n (ranging from 10 to 100), and the y-axis represents the time complexity (ranging from 1 to 5). The curves show that Alg. 1 ($O(n^3)$) grows much faster than Alg. 2 ($O(n^2)$), which in turn grows faster than Alg. 3 ($O(n \log n)$). Alg. 4 ($O(n)$) is the most efficient, showing a linear growth rate that is significantly lower than the others for large n .

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ | ≡ ↺ 🔍 ↻

Taxas de crescimento

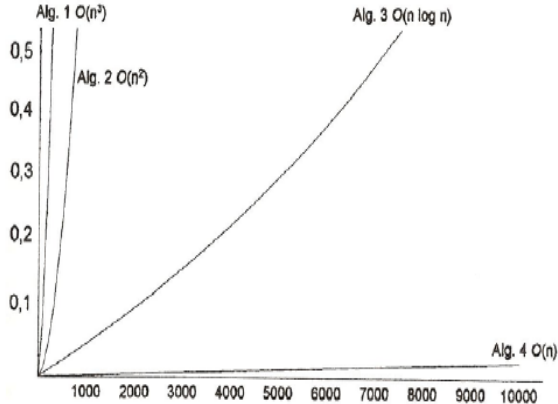


Figure : Gráfico (n vs. segundos) das taxas de crescimentos dos quatro algoritmos para entradas maiores.

Sumário

1

Algoritmo [2]

- Algoritmo
- Análise de Algoritmos
- Eficiência de algoritmos

2

Análise assintótica

- Análise assintótica
- Conceitos de matemática
- Notações

3

Taxas de crescimento

- Taxas de crescimento
- Análise de Algoritmos
- Cálculo do tempo de execução

Análise de algoritmos

- Existem basicamente 2 formas de estimar o tempo de execução de programas e decidir quais são os melhores:
 - empiricamente,
 - teoricamente.
- É desejável e possível estimar qual o melhor algoritmo sem ter que executá-los:
 - Função da análise de algoritmos.

Calculando o tempo de execução

- Supondo que as operações simples demoram uma unidade de tempo para executar, considere o programa abaixo para calcular o resultado de

$$\sum_{i=1}^n i^3$$

```
1 Início
2 declare soma_parcial numérico;
3 soma_parcial ← 0;
4 para  $i \leftarrow 1$  até  $n$  faça
5   soma_parcial ← soma_parcial +  $i * i * i$ ;
6 escreva(soma_parcial);
7 Fim
```

Calculando o tempo de execução

$$\sum_{i=1}^n i^3$$

- 3 1 unidade de tempo
- 4 1 unidade para iniciação de i , $n + 1$ unidades para testar se $i = n$ e n unidades para incrementar $i = 2n + 2$
- 5 4 unidades (1 da soma, 2 das multiplicações e 1 da atribuição) executada n vezes (pelo comando “para”) = $4n$ unidades
- 6 1 unidade para escrita
- **Custo total:** somando tudo, tem-se $6n + 4$ unidades de tempo, ou seja, a função é $\mathcal{O}(n)$!

Calculando o tempo de execução

- Ter que realizar todos esses passos para cada algoritmo (principalmente algoritmos grandes) pode se tornar uma tarefa cansativa
- Em geral, como se dá a resposta em termos do *big-oh*, costuma-se desconsiderar as constantes e elementos menores dos cálculos:
 - No exemplo anterior:
 - A linha 3 $\text{soma_parcial} \leftarrow 0$ é insignificante em termos de tempo,
 - É desnecessário ficar contando 2, 3 ou 4 unidades de tempo na linha 5 $\text{soma_parcial} \leftarrow \text{soma_parcial} + i * i$,
 - O que realmente dá a grandeza de tempo desejada é a repetição na linha 4 “para $i \leftarrow 1$ até n faça”.

Regras para o cálculo

- Repetições:
 - O tempo de execução de uma repetição é pelo menos o tempo dos comandos dentro da repetição (incluindo testes) vezes o número de vezes que é executada.
- Repetições aninhadas:
 - A análise é feita de dentro para fora,
 - O tempo total de comandos dentro de um grupo de repetições aninhadas é o tempo de execução dos comandos multiplicado pelo produto do tamanho de todas as repetições.
 - O exemplo abaixo é $\mathcal{O}(n^2)$:
para $i \leftarrow 0$ até n faça
 para $j \leftarrow 0$ até n faça
 faça $k \leftarrow k + 1$;

Regras para o cálculo

- Comandos consecutivos:

- É a soma dos tempos de cada um, o que pode significar o máximo entre eles,
- O exemplo abaixo é $\mathcal{O}(n^2)$, apesar da primeira repetição ser $\mathcal{O}(n)$:

```
para  $i \leftarrow 0$  até  $n$  faça  
   $k \leftarrow 0$ ;  
para  $i \leftarrow 0$  até  $n$  faça  
  para  $j \leftarrow 0$  até  $n$  faça  
    faça  $k \leftarrow k + 1$ ;
```

Regras para o cálculo

- Se... então... senão:
 - Para uma cláusula condicional, o tempo de execução nunca é maior do que o tempo do teste mais o tempo do maior entre os comandos relativos ao então e os comandos relativos ao senão,
 - O exemplo abaixo é $\mathcal{O}(n)$:
se $i < j$
então $i \leftarrow i + 1$
senão para $k \leftarrow 1$ até n faça
 $i \leftarrow i * k$;
- Chamadas a sub-rotinas:
 - Uma sub-rotina deve ser analisada primeiro e depois ter suas unidades de tempo incorporadas ao programa/sub-rotina que a chamou.

Regras para o cálculo

- **Exercício:** Estime quantas unidades de tempo são necessárias para rodar o algoritmo abaixo:

```
1  Início
2  declare  $i$  e  $j$  numéricos;
3  declare  $A$  vetor numérico de  $n$  posições;
4   $i \leftarrow 1$ ;
5  enquanto  $i \leq n$  faça
6     $A[i] \leftarrow 0$ ;
7     $i \leftarrow i + 1$ ;
8  para  $i \leftarrow 1$  até  $n$  faça
9    para  $j \leftarrow 1$  até  $n$  faça
10      $A[i] \leftarrow A[i] + i + j$ ;
11  Fim
```

Referências I

- [1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Algoritmos - Teoria e Prática.
Ed. Campus, Rio de Janeiro, Segunda Edição, 2002.
- [2] Pardo, T. A. S.
Análise de Algoritmos. SCE-181 Introdução à Ciência da Computação II.
Slides. Ciência de Computação. ICMC/USP, 2008.
- [3] Rosa, J. L. G.
Análise de Algoritmos. SCC-201 Introdução à Ciência da Computação II (capítulo 1).
Slides. Ciência de Computação. ICMC/USP, 2009.