ESTRUTURAS DE DADOS II

MSC. DANIELE CARVALHO OLIVEIRA

DOUTORANDA EM CIÊNCIA DA COMPUTAÇÃO - USP

MESTRE EM CIÊNCIA DA COMPUTAÇÃO – UFU

BACHAREL EM CIÊNCIA DA COMPUTAÇÃO - UFJF

2

PRINCÍPIO DE ANÁLISE DE ALGORITMOS

3 INTRODUÇÃO

"Uma base sólida de conhecimento e técnica de algoritmos é uma das características que separa o programador experiente do aprendiz. Com a moderna tecnologia de computação, você pode realizar algumas tarefas sem saber muito sobre algoritmos, mas com um boa base em algoritmos você pode fazer muito, muito mais."

CLRS: Cormen, Leiserson, Rivest, Stein

PROBLEMA X ALGORITMO

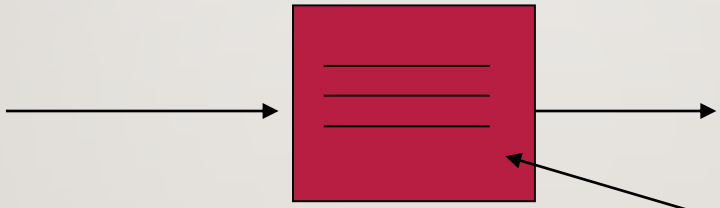
5 "PROBLEMAS" E "ALGORITMOS"

- O que é um problema ?
  - Função P: Input → Output
  - Instância do Problema = I ∈ Input

- Exemplos de Problemas
  - Problema dos Primos
    - Primos: N → {Sim, Não}
      - Instância = número natural
      - Primos (n) = Sim, se n é primo; Não, caso contrário
  - Problema da Decomposição em primos
    - Decomposição:
    - N → {Seq | Seq = <(p1,n1),...,(pk,nk)>, pi primo}
      - Decomposição (n) = <(p1,n1), ..., (pk,nk)> se n = p1n1p2n2...pknk
      - Exemplo: Decomposição (10) = <(2,1), (5,1)>, pois 10 = 2x5

6 "PROBLEMAS" E "ALGORITMOS"

- Solução de um Problema
  - Conjunto finito de instruções cuja execução sobre o input termina depois de um tempo finito, produzindo no final o output.

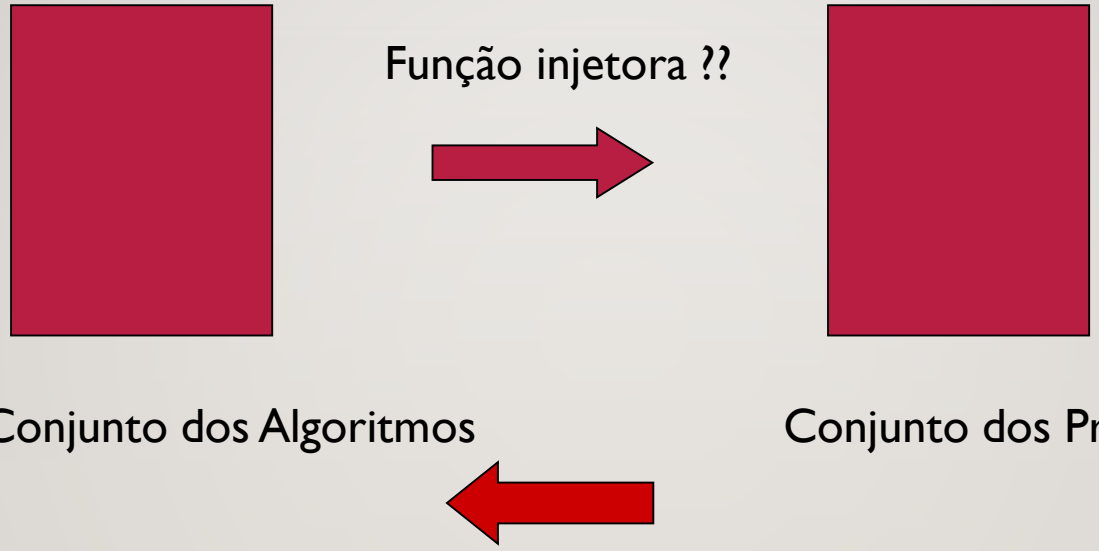


Algoritmo que resolve o problema

  - Solução de um problema = algoritmo

- Algoritmo : conjunto finito de instruções que transformam uma entrada em uma saída depois de um tempo finito.
- Todo Algoritmo está associado a um Problema

7 PERGUNTAS



Não é função injetora: podem existir diferentes algoritmos para resolver um mesmo problema

Não é função sobrejetora: Existem problemas que não têm solução

O QUE É UM BOM ALGORITMO ?

- Correto ?
- Eficiente em tempo ?
- Eficiente em espaço ?

9 TIPOS DE ALGORITMOS

- Problema P
  - P é decidível (tem solução ?)
  - Qual a complexidade de P ?
  - Se P for NP-completo: existem algoritmos aproximados ?
  - Como projetar um bom algoritmo para P, dentro das limitações da complexidade inerente ao problema P ?
- Tipos de Algoritmos:
  - Exatos versus Aproximativos
  - Iterativos versus Recursivos
  - Probabilísticos (ou Randômicos)

10 O QUE É UM ALGORITMO PROBABILÍSTICO ?

Problema: Encontrar um elemento *a* em um array A de *n* elementos

Input: A, *a* , *n*

Output: posição m onde se encontra *a* ou *'Não'*

Algoritmo Exato

```
Begin
    Para  i = 1, ..., n faça
        Se A[i] = a
            Retorna i
            Pára
    Retorna 'Não'
End
```

Se n é muito grande, algoritmo pode levar muito tempo para dar a resposta .

O QUE É UM ALGORITMO PROBABILÍSTICO ?

Problema: Encontrar um elemento *a* em um array A de *n* elementos

Input: A, *a* , *n*

Output: posição m onde se encontra *a* ou *'Não'*

Algoritmo Monte Carlo (não exato)

```
begin
  i=1
  repeat
        Selecione aleatoriamente um número inteiro m em [1,n]
              Se A[m] = a
              Retorna m e pára
        i = i + 1
   until i=k
   Retorna 'Não'
end
```

Monte Carlo encontra 'a' com Probabilidade $(1 - (1/2)^k)$

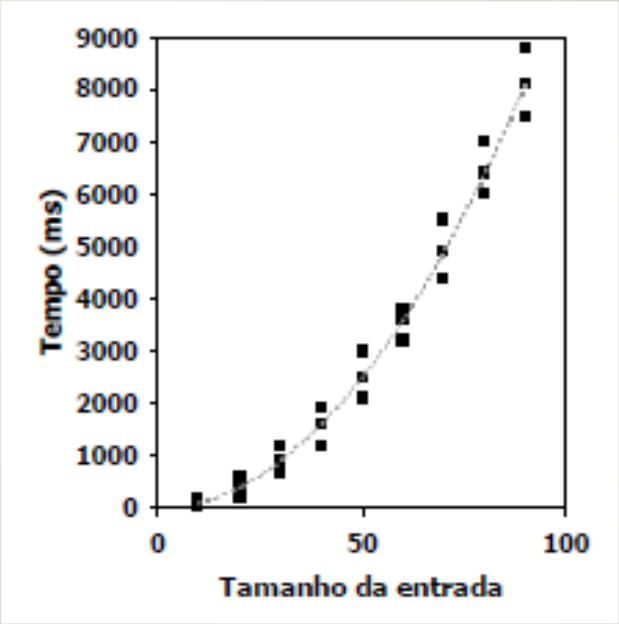
Tempo de Execução de Monte Carlo é fixo

12 ANÁLISE DE UM ALGORITMO

- O que é analisar um algoritmo ?
- Determinar sua eficiência
  - quanto ao tempo de execução
  - quanto à quantidade de memória utilizada para executar o algoritmo

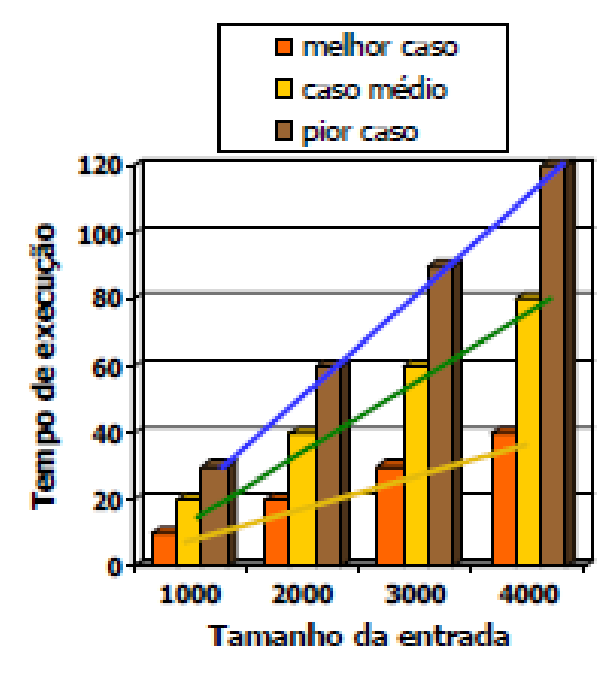
13 ANÁLISE EXPERIMENTAL

- Escreva uma determinada implementação para um algoritmo
- Execute esse programa com diversas entradas de vários tamanhos
- Para cada um desses casos, obtenha o tempo real de execução
- Desenhe um gráfico com os resultados obtidos



14 TEMPO DE EXECUÇÃO

- O tempo de execução de um algoritmo varia com o tamanho da entrada do problema
- 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



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.

16 ANÁLISE DE UM ALGORITMO

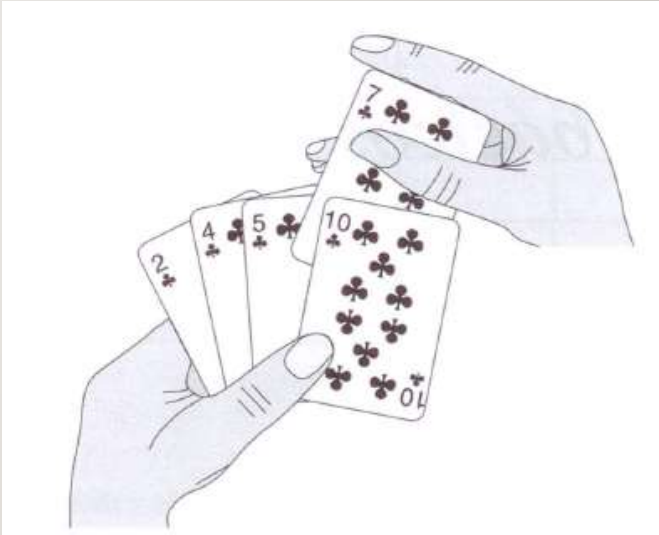
- Que operações atômicas considerar no cálculo de custo ?
- Operações atômicas = custo constante

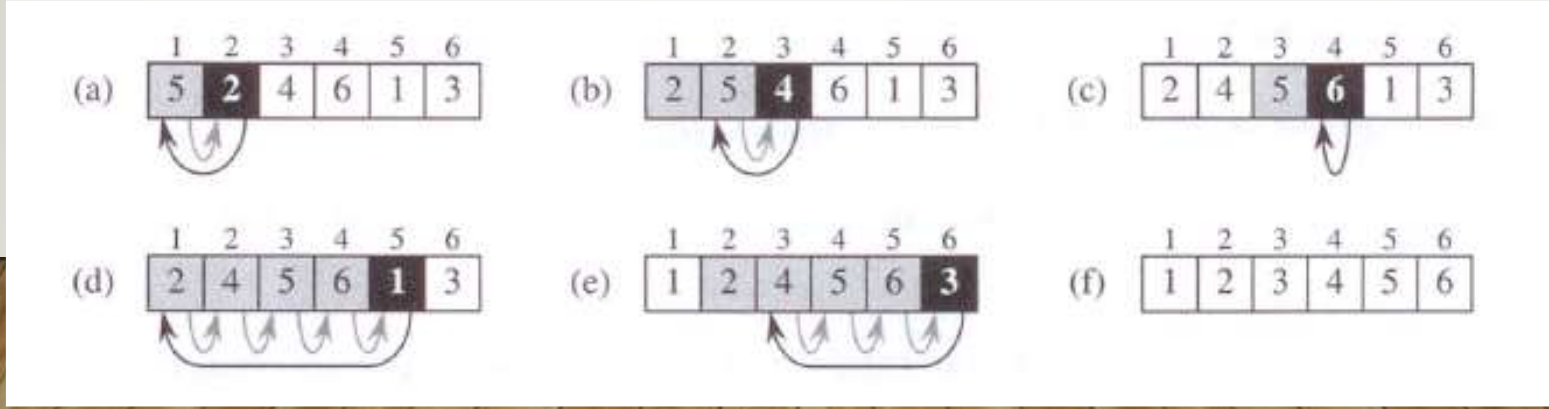
  - Operações aritméticas
    - Soma, subtração, multiplicação, divisão, resto, piso, teto
  - Movimentação de dados:
    - Carregar, armazenar, copiar
  - Controle
    - Desvio condicional e incondicional
    - Chamada e retorno de subrotinas

17 EXEMPLO: PROBLEMA, ALGORITMO, ANÁLISE DO ALGORITMO

- Problema: (Ordenação de uma sequência)
- Input: sequência de n números A = <a1,...,an>
- Output: B = <b1,...,bn>, onde B é uma permutação de A e b1≤ b2 ≤ ... ≤ bn

Projeto de um Algoritmo





18 ALGORITMO INSERTION-SORT

Insertion-Sort (A) **Entrada :** A = array [a1,...,an]

```
1.   For j ← 2 to n
2.       do chave ← A[j]
3.          i ← j – 1          % Procura lugar anterior onde inserir a chave
4.          While i > 0 e A[i] > chave
5.              do A[i+1] ← A[i]
6.                 i ← i - 1
7.          A[i+1] ← chave
```

Algoritmo é executado *in place* :

Espaço necessário = espaço da entrada + espaço das variáveis Chave, j, i

Complexidade em Espaço = constante (=3)

(não se conta o espaço ocupado pela entrada)

19 ALGORITMO INSERTION-SORT

Insertion-Sort (A)

		Custo	Vezes
1.	**For** j ← 2 to n	c1	**n**
2.	**do** chave ← A[j]	c2	**n-1**
3.	i ← j – 1	c3	**n-1**
4.	**While** i > 0 e A[i] > chave	c4	$\Sigma^{n}_{j=2}$ **tj**
5.	**do** A[i+1] ← A[i]	c5	$\Sigma^{n}_{j=2}$ **(tj-1)**
6.	i ← i - 1	c6	$\Sigma^{n}_{j=2}$ **(tj-1)**
7.	A[i+1] ← chave	c7	**n-1**

tj = número de vezes que o teste do While em (4) é executado para cada valor j do loop for

20 ALGORITMO INSERTION-SORT

- T(n) = custo temporal do algoritmo em função do tamanho da entrada (=n)

$$T(n) = c1.n + c2(n-1) + c3(n-1) + c4(\Sigma^{n}_{j=2}\ tj) + c5(\Sigma^{n}_{j=2}\ (tj-1)) + c6(\Sigma^{n}_{j=2}\ (tj-1)) + c7(n-1)$$

- T(n) depende de tj
- O valor de tj depende do "grau de desordenação" da entrada.

21 ALGORITMO INSERTION-SORT

Melhor caso: a entrada está corretamente em ordem crescente.

- **tj = 1**, para j = 2,...,n

$T(n) = c1.n + c2(n-1) + c3(n-1) + c4(n-1) + c7(n-1)$

$= (c1+c2+c3+c4+c7)n - (c2+c3+c4+c7)$

Pior caso : a entrada está ordenada de forma reversa (descrescente)

- **tj = j**, para j = 2,...,n

$\Sigma^{n}_{j=2} j = [n(n+1)/2] - 1$

$T(n) = c1.n + c2(n-1) + c3(n-1) + c4([n(n+1)/2] - 1) + c5([n(n-1)/2]) +$

$+ c6([n(n-1)/2]) + c7(n-1) =$

$= (c4/2 + c5/2 + c6/2)n^2 + (c1+c2+c3 - c4/2 - c5/2 - c6/2 + c7)n - (c2 + c3 + c4 + c7)$

22 ALGORITMO INSERTION-SORT

Caso médio:

tj = j/2, para j = 2,...,n

Exercício: Determinar o valor de T(n) para o caso médio.

23 OUTRO EXEMPLO

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

```
int arrayMax (int A[], int n)
{
   currentMax = A[0];                          1
   for (i=1; i<n; i++) {          1 + (repete n-1 vezes) [ 1 teste,
         if (A[i] > currentMax)      1 incremento, 1 teste,
           currentMax = A[i];      1 atribuição (no máximo) ]
   }
   return currentMax;                          1
}                                  Total: 3 + (n-1).4 = 4n-1
```

- Inspecionando 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 4n-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, $a.(4n-1) \leq T(n) \leq b.(4n-1)$
- O tempo de execução T(n) é limitado por duas funções lineares.

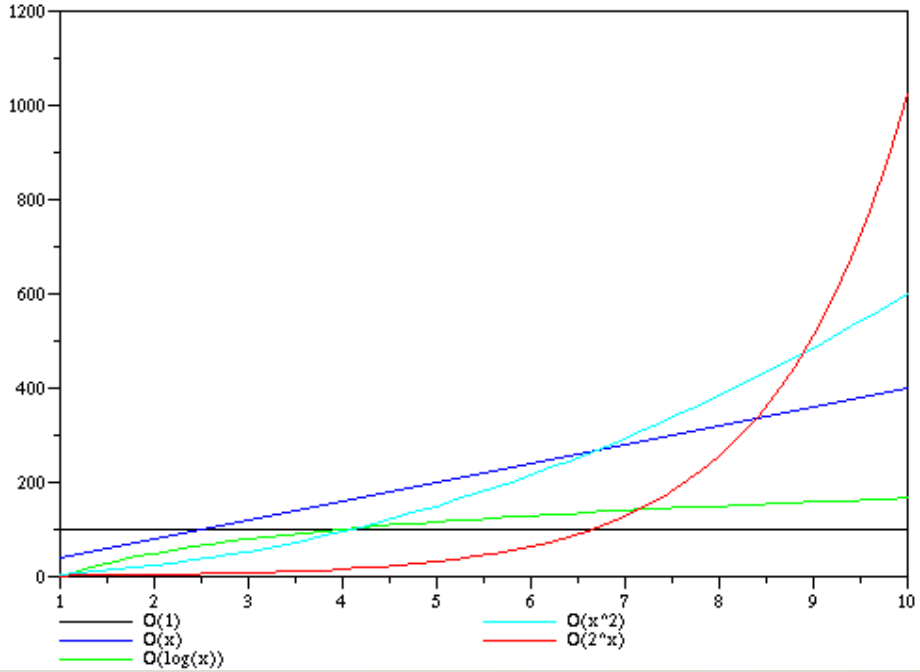
TAXA DE CRESCIMENTO DO TEMPO E EXECUÇÃO

- 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 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.

26 TAXAS DE CRESCIMENTO

- Exemplos de taxas de crescimento:
  - Linear $\approx n$
  - Quadrática $\approx n^2$
  - Cúbica $\approx n^3$

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



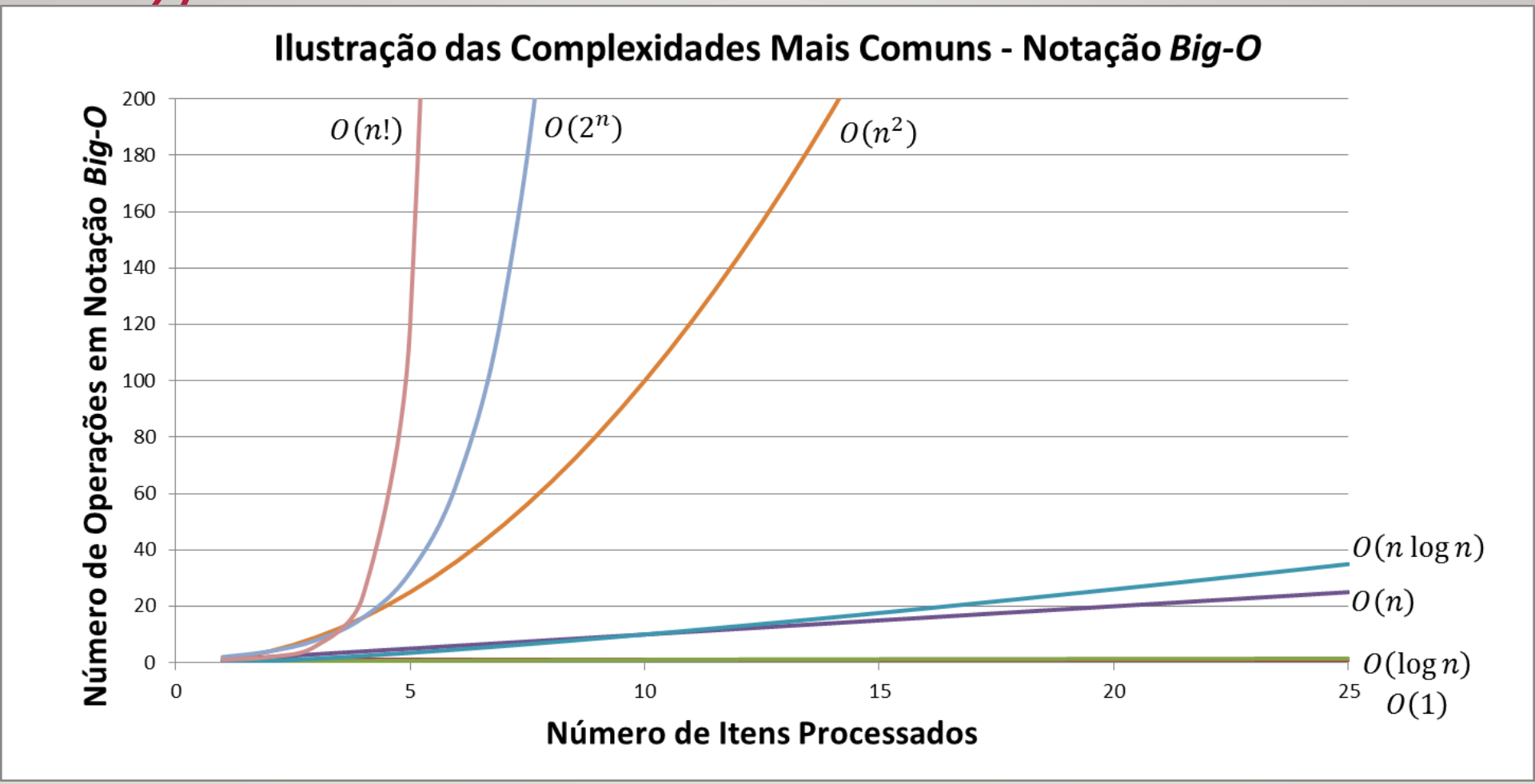


Ilustração das Complexidades Mais Comuns - Notação Big-O
Número de Operações em Notação Big-O
200
180
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
Número de Itens Processados
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)

EXERCÍCIO

- Elabore o Algoritmo do bubble sort e calcule o seu tempo T(n) de execução no pior caso.

FIM DA AULA 4

Próxima aula:

Busca em Texto