

ED62A-COM2A

ESTRUTURAS DE DADOS

Aula 02 - Noções Básicas de
Complexidade de Algoritmos

Prof. Rafael G. Mantovani

20/08/2019

Roteiro



- 1 Introdução / Motivação**
- 2 Análise de Complexidade / Assintótica**
- 3 Notações Assintóticas**
- 4 Classes de Complexidade**
- 5 Síntese / Revisão**
- 6 Referências**

Roteiro

- 1 Introdução / Motivação**
- 2 Análise de Complexidade / Assintótica**
- 3 Notações Assintóticas**
- 4 Classes de Complexidade**
- 5 Síntese / Revisão**
- 6 Referências**

Introdução



Problema
(tarefa)

Introdução



Problema
(tarefa)



Introdução



Problema
(tarefa)

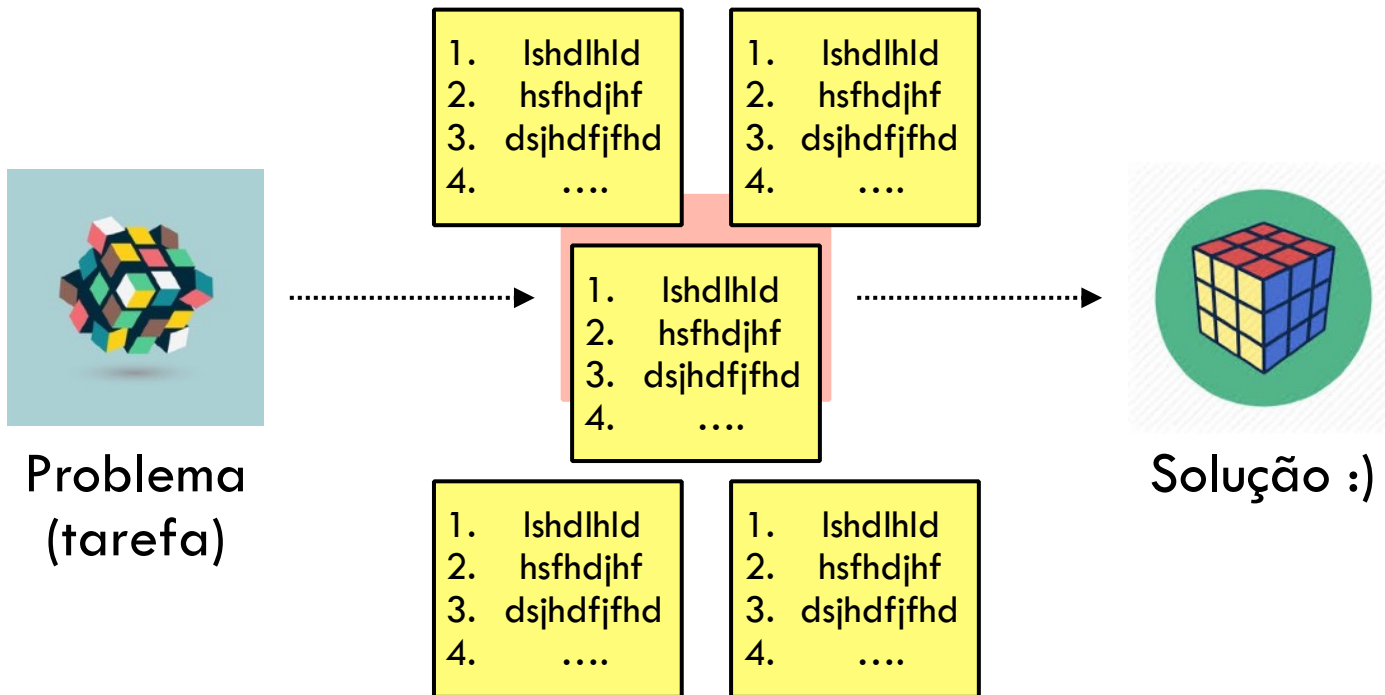


Algoritmo

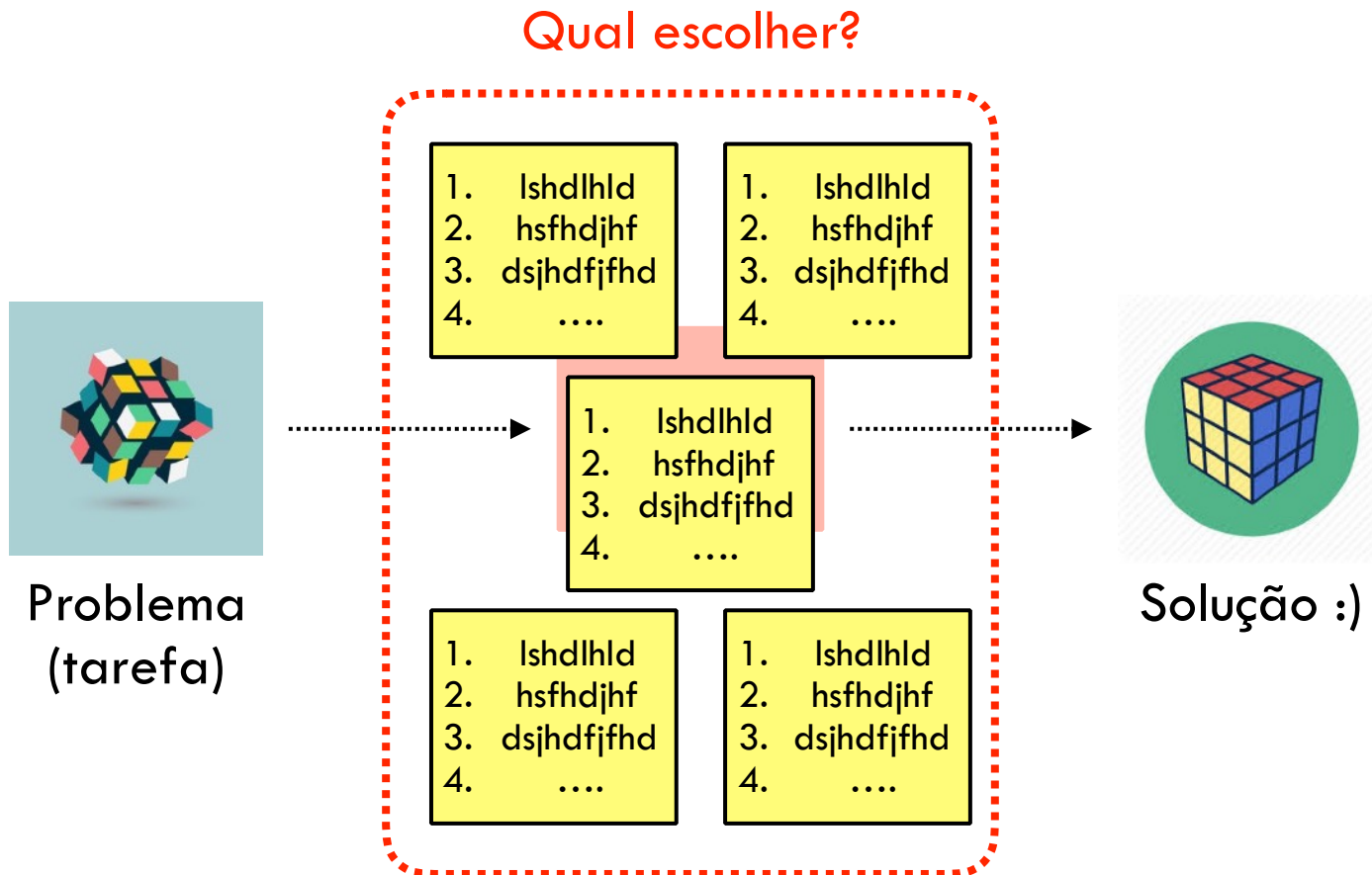


Solução :)

Introdução



Introdução



Introdução

Comparação de Funções de Complexidade

Função de custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
2^n	0,001 s	1 s	17,9 min	12,7 dias	35,7 anos	366 séc.
3^n	0,059 s	58 min	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

Introdução

Vetores Estáticos

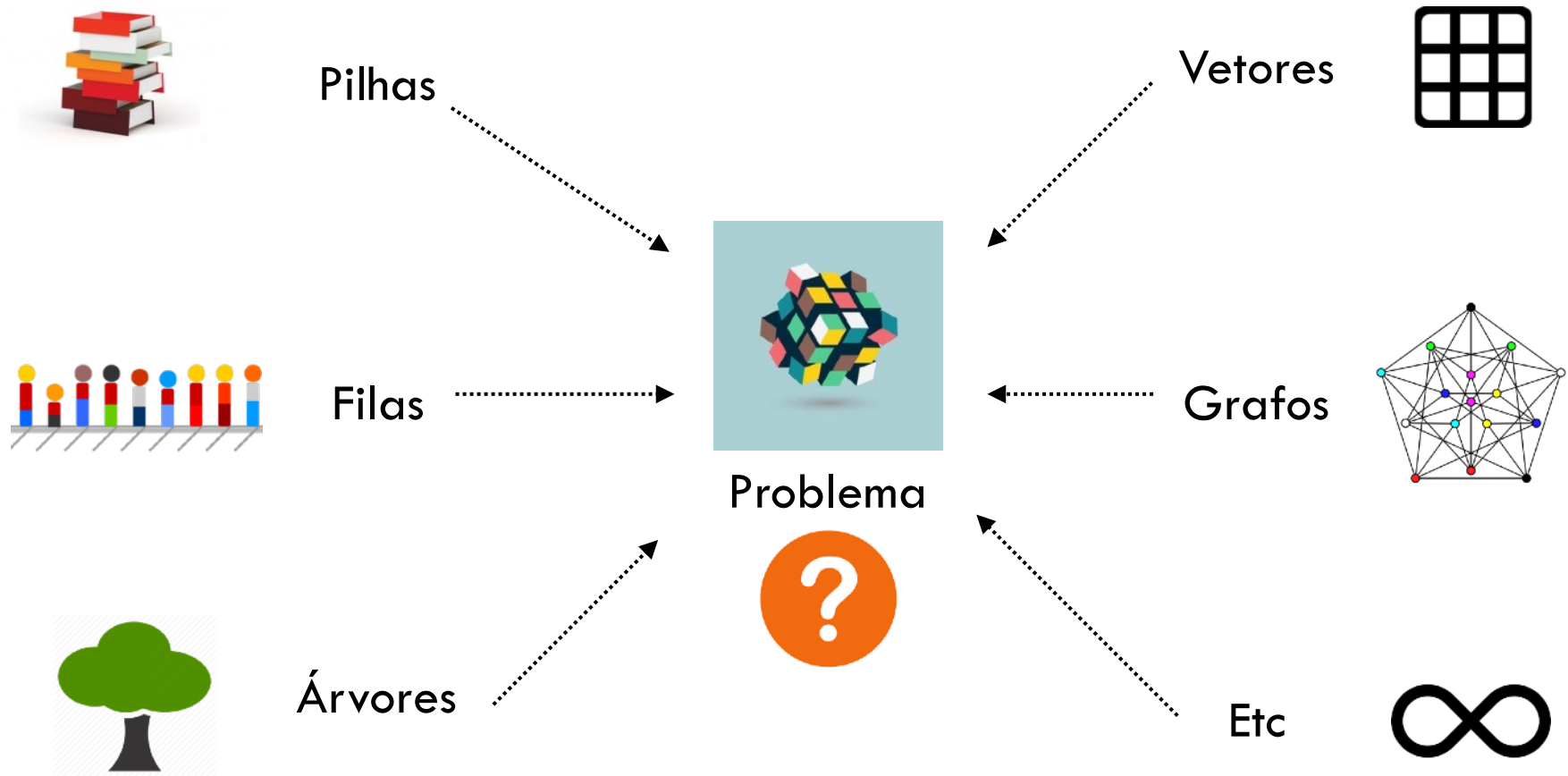
vs

Estruturas Dinâmicas

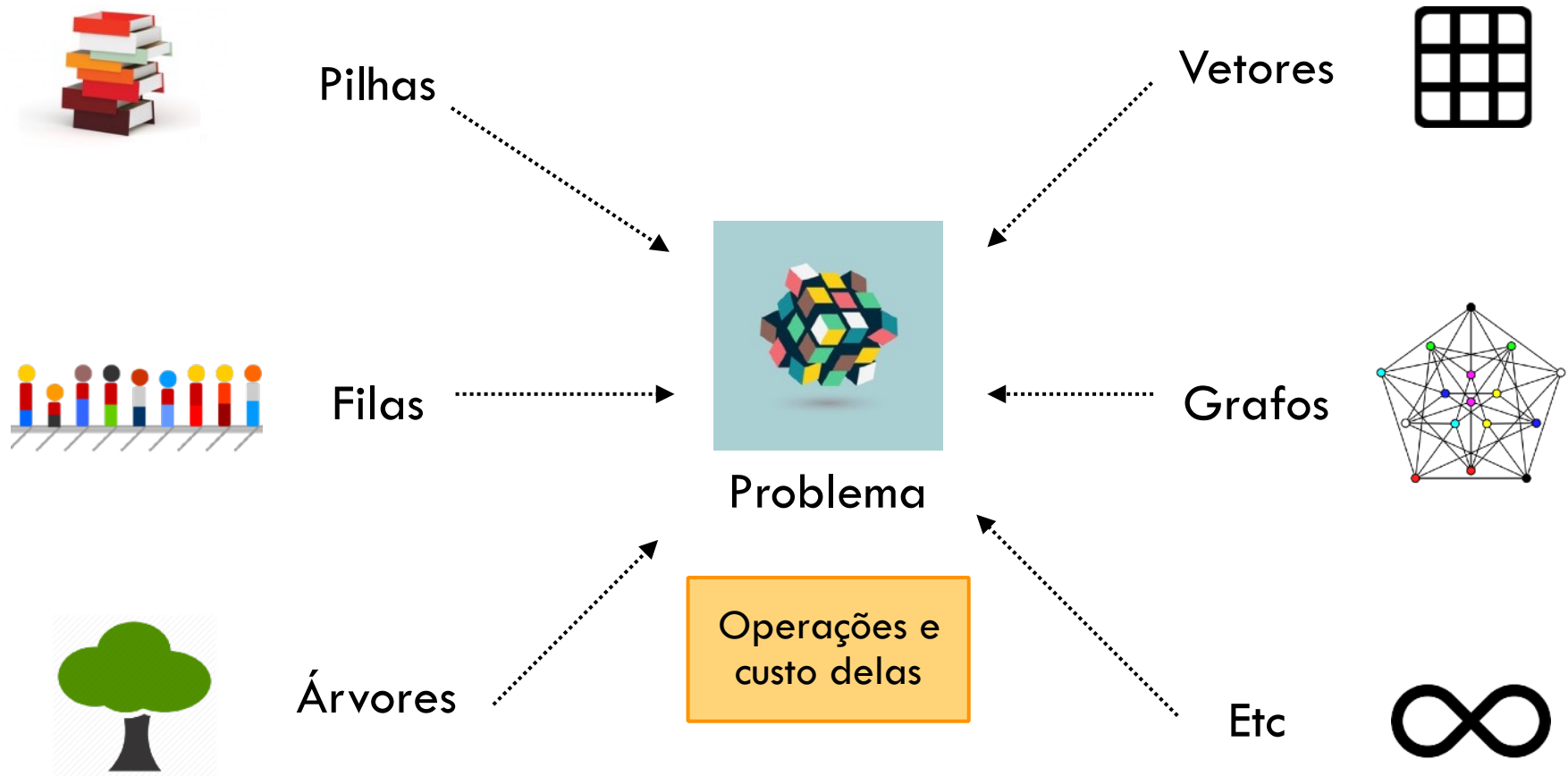
```
27
28 int *lerSeq (int *n) {
29     *n = 2;
30     int *seq = malloc((*n)* sizeof (int));
31
32     if(!seq) {          // Há memória disponível?
33         *n = 0;
34         return NULL;
35     }
36
37     scanf("%d", &seq[0]);
38     scanf("%d", &seq[1]);
39
40     int i=1, num;
41     for(;;i++) {
42         scanf("%d", &num);
43
44         if (num==0 && (seq[i]==0) && (seq[i-1]==0)) {
45             *n=i-1;
46             return seq;
47         }
48
49         if (i==(*n)-1) {
50             seq = dobrarSeq(seq, n);
51             if (seq==NULL) return NULL;
52         }
53         seq[i+1]=num;
54     }
55 }
56
57
58
```

```
68
69 ListNode *lerSeq (int mode) {
70
71     ListNode* first = NULL;
72     ListNode* last = NULL;
73
74     int item1, item2;
75     scanf("%d", &item1);
76     scanf("%d", &item2);
77
78     int num;
79     for(;;) {
80         scanf("%d", &num);
81
82         if (num==0 && (item1==0) && (item2==0)) {
83             return first;
84         }
85
86         if (mode==1) {
87             last = appendNode(last, item1);
88             if (first == NULL) {
89                 first = last;
90             }
91         } else {
92             first = insertFirst(first, item1);
93         }
94
95         item1 = item2;
96         item2 = num;
97     }
98 }
99
```

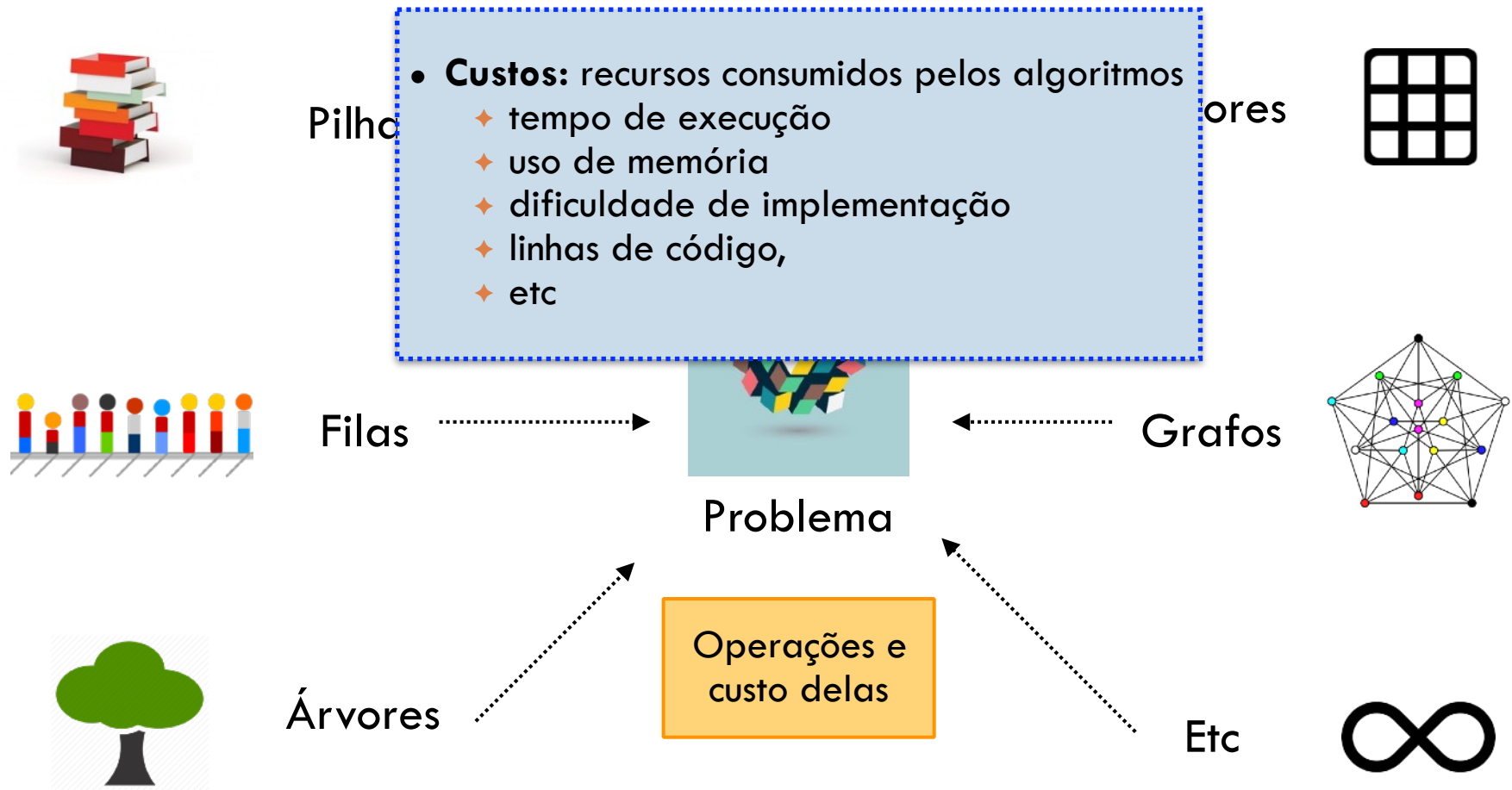
Introdução



Introdução

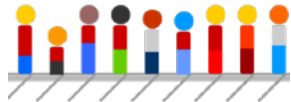


Introdução



Introdução

- **Ideal:** medir os custos independentemente de:
 - ♦ hardware/software
 - ♦ sistema operacional
 - ♦ compilador/linguagem
 - ♦ etc



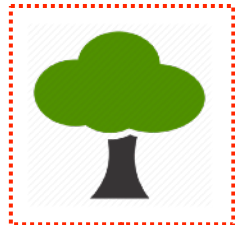
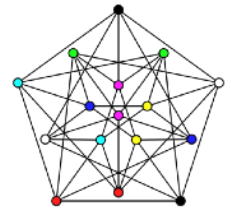
Filas



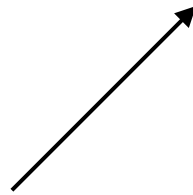
Problema



Grafos



Árvores



Análise de
Complexidade
(unidades lógicas)



Etc



Roteiro

- 1 Introdução / Motivação
- 2 **Análise de Complexidade / Assintótica**
- 3 Notações Assintóticas
- 4 Classes de Complexidade
- 5 Síntese / Revisão
- 6 Referências

Análise de Complexidade de Algoritmos

Objetivo:

- ajudar a determinar qual algoritmo é mais eficiente para resolver um problema
- Medir como o tempo ou espaço aumenta com relação ao **tamanho da entrada (N)**

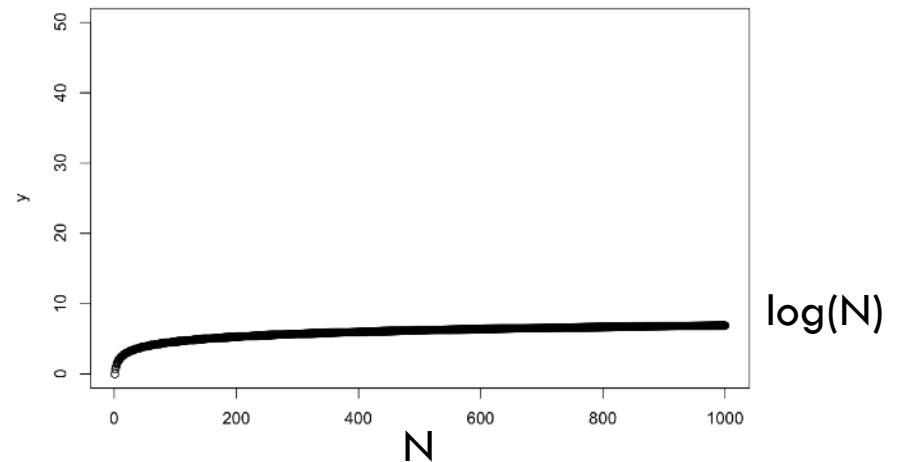


Árvores

+



Problema

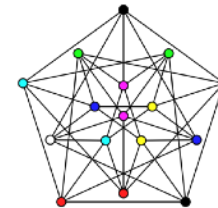


Análise de Complexidade de Algoritmos

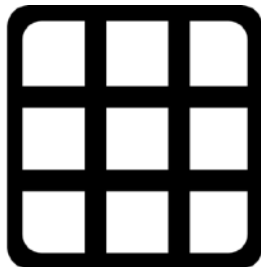
- Tamanho da entrada (N):
 - número de elementos de dados que são relevantes na entrada do algoritmo. Varia dependendo do problema.

17	23	24	31	44	52	70	90
----	----	----	----	----	----	----	----

N = elementos/sequencia



N = Vértices/Arestas



N = Dimensões da matriz

010110
101100
010111

N = Quantidade de bits

Exemplo

Programa: fatorial de um número N
Tamanho da entrada: N;

```
void fatorial(int n) {  
    int i;  
    int fat = 1;  
    for(i = 1; i <= n; i++) {  
        fat = fat * i;  
    }  
    return(fat);  
}
```

Exemplo

Programa: fatorial de um número N
Tamanho da entrada: N;

```
void fatorial(int n) {  
    int i;  
    int fat = 1;  
    for(i = 1; i <= n; i++) {  
        fat = fat * i;  
    }  
    return(fat);  
}
```

executa **n** vezes
tempo constante **c1**

Exemplo

Programa: fatorial de um número N
Tamanho da entrada: N;

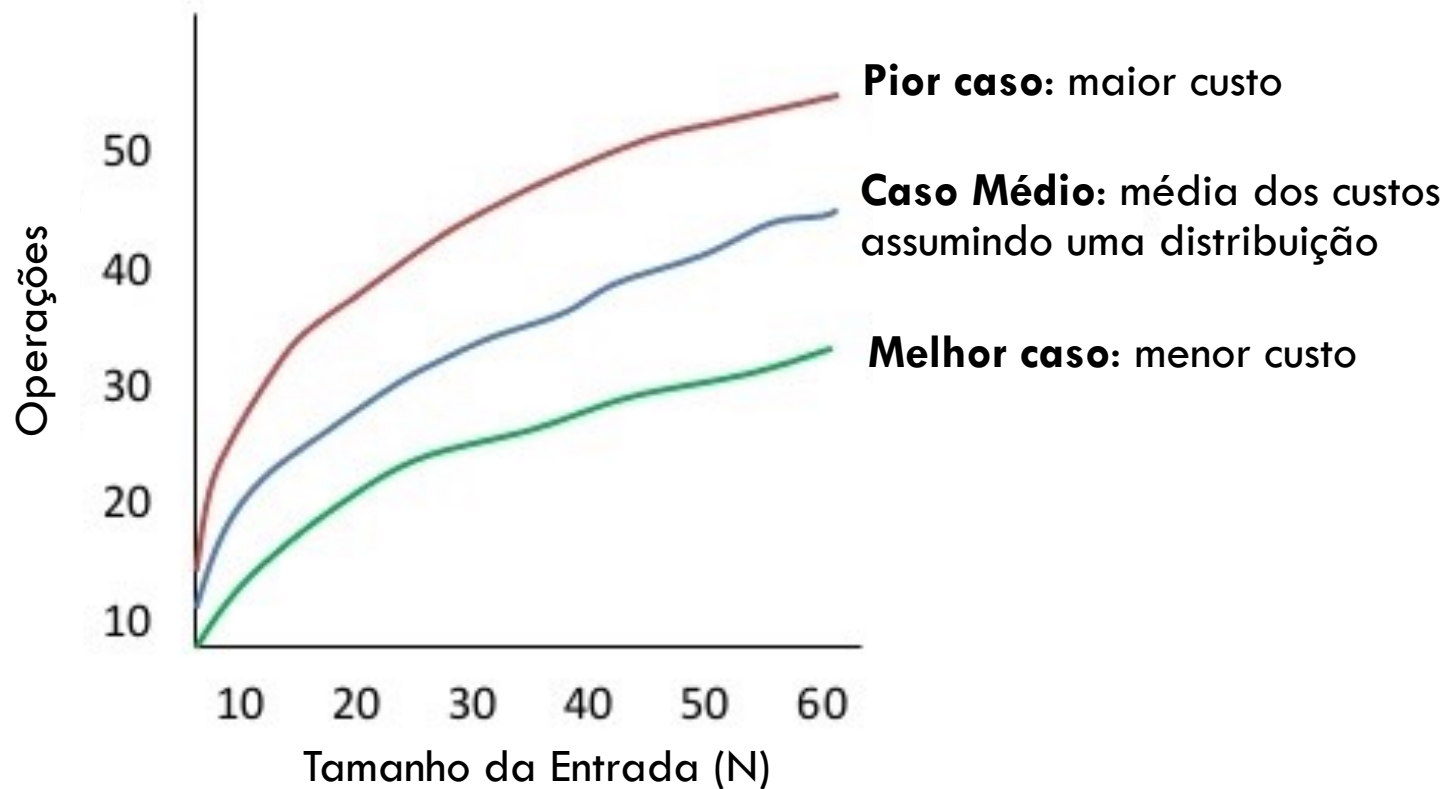
```
void fatorial(int n) {  
    int i;  
    int fat = 1;  
    for(i = 1; i <= n; i++) {  
        fat = fat * i;  
    }  
    return(fat);  
}
```

executa **n** vezes

tempo constante **c1**

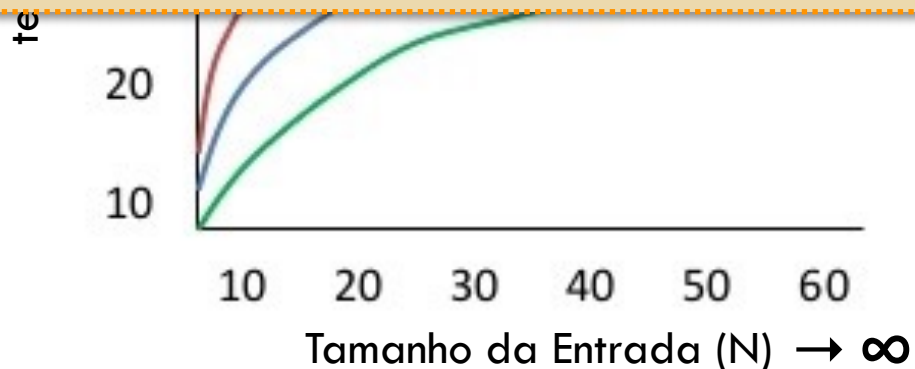
Custo total: **$c1 * n$**

Análise de Complexidade de Algoritmos



Análise de Complexidade de Algoritmos

- O custo exato do algoritmo é irrelevante. O importante é obter uma **boa aproximação** ou limite (*tight bound*)
- Entradas pequenas são irrelevantes. O importante é o comportamento do algoritmo quando o tamanho da entrada é grande (*complexidade assintótica*).



Análise Assintótica

- Assume um modelo abstrato de computador com um conjunto básico de operações e seus custos
- O custo de tempo é uma função $T(N)$
 - N representa o tamanho da entrada
- Exemplo: busca sequencial em um *array* de números inteiros

$A =$

17	23	24	31	44	52	70	90
----	----	----	----	----	----	----	----

Exemplo 01: melhor caso

Melhor caso:

	0	1	2	3	4	5	6	7	(posição)
A =	17	23	24	31	44	52	70	90	

Exemplo 01: melhor caso

Melhor caso: elemento a ser encontrado está na primeira posição de A

Ex: valor de consulta = 17

	0	1	2	3	4	5	6	7	(posição)
A =	17	23	24	31	44	52	70	90	
	↑								

Array: **N** posições

Total de operações: 1

$T(N) = 1$

Exemplo 02: pior caso

Pior caso:

	0	1	2	3	4	5	6	7	(posição)
A =	17	23	24	31	44	52	70	90	

Exemplo 02: pior caso

Pior caso: elemento a ser encontrado não está em A

Ex: valor de consulta = 12

	0	1	2	3	4	5	6	7	(posição)
A =	17	23	24	31	44	52	70	90	
	↑	↑	↑	↑	↑	↑	↑	↑	

Array: **N** posições

Total de operações: N

$T(N) = N$

Exercício 1: caso médio?

Qual a complexidade do caso médio da busca linear ?

	0	1	2	3	4	5	6	7	(posição)
A =	17	23	24	31	44	52	70	90	

Exercício 1: caso médio?

Qual a complexidade do caso médio da busca linear ?

	0	1	2	3	4	5	6	7	(posição)
A =	17	23	24	31	44	52	70	90	

Dica: média de todos os casos

Roteiro

- 1 Introdução / Motivação
- 2 Análise de Complexidade / Assintótica
- 3 Notações Assintóticas
- 4 Classes de Complexidade
- 5 Síntese / Revisão
- 6 Referências

Notações Assintóticas

- Permitem descrever o comportamento assintótico de uma função, quando o argumento N (entrada, quantidade de dados):
 - $N \rightarrow \infty$
- Notação O : limite superior
- Notação Ω : limite inferior
- Notação Θ : limite firme ou restrito

Notação O: limite superior

- Expressa um limite superior para o comportamento assintótico de uma função:

$$O(g(n)) = \{ f(n) \mid \exists c > 0, n_0, \forall n > n_0, f(n) \leq c * g(n) \}$$

- Informalmente, $f(n) \in O(g(n))$ não cresce mais rapidamente que $g(n)$, para valores de n suficientemente grandes (como se $f(n) \leq g(n)$).

Problema: estabelece que precisam existir c e n_0 , mas não se diz como calculá-los.

- Geralmente existem vários pares (c, n_0)

Exemplos: $5n = O(n^2)$, $10n^2 + 5n = O(n^2)$, $n^3 \neq O(n^2)$, $\log_5 n = O(\log n)$.

□

Notações Ω e Θ

- Notação Ω : expressa um limite inferior:

$$\Omega(g(n)) = \{ f(n) \mid \exists c > 0, n_0, \forall n > n_0, f(n) \geq c * g(n) \}$$

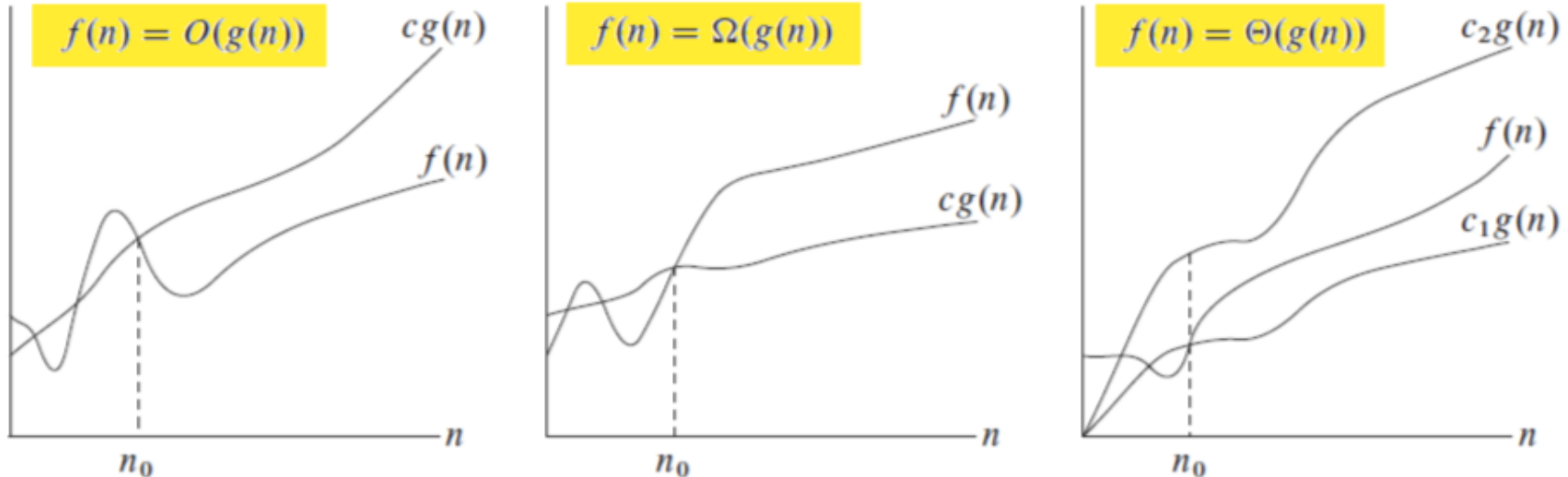
Exemplos: $300n + 100 = \Omega(n)$, $10n^2 + 5n = \Omega(n^2)$

- Notação Θ : expressa um limite firme ou restrito

$$\Theta(g(n)) = \{ f(n) \mid \exists c_1 > 0, c_2 > 0, n_0, \forall n > n_0, \\ c_1 * g(n) \leq f(n) \leq c_2 * g(n) \}$$

Exemplos: $10n^2 + 5n = \Theta(n^2)$, $n \neq \Theta(n^2)$, $n^3 \neq \Theta(10n^2)$

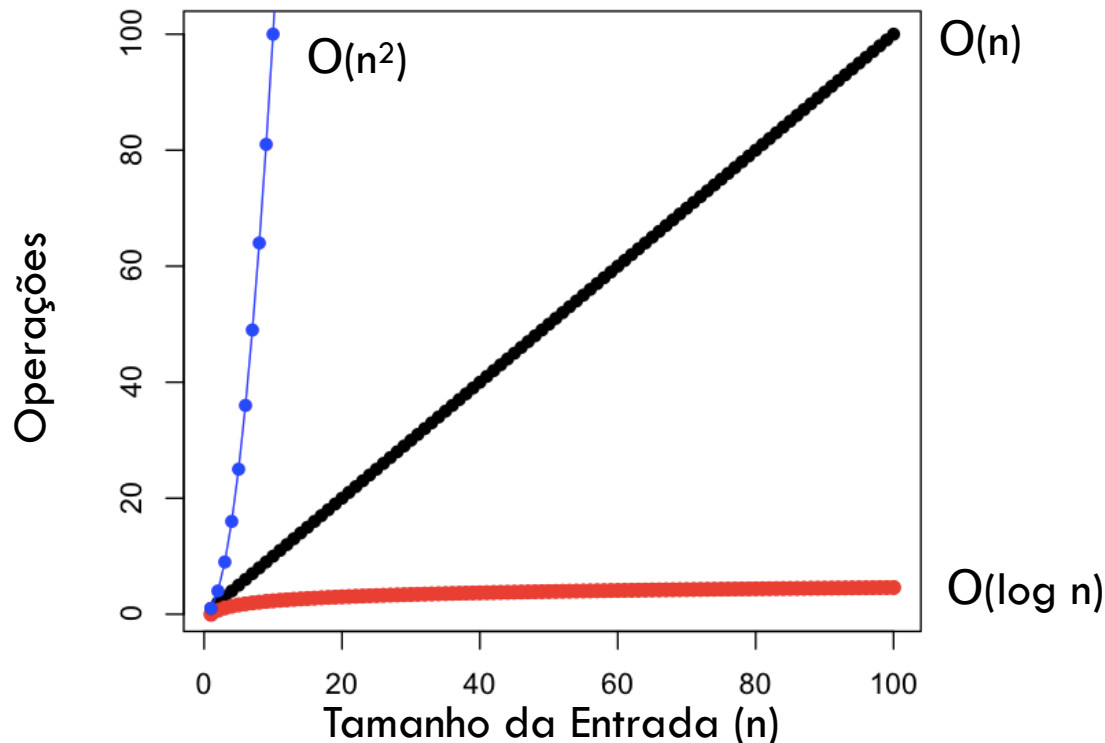
Notações Assintóticas



1. Todas são reflexivas e transitivas; Θ é simétrica.
2. $f(x) = O(g(x))$ sse $g(x) = \Omega(f(x))$
3. $f(x) = \Theta(g(x))$ sse $f(x) = O(g(x))$ e $f(x) = \Omega(g(x))$
4. $O(k \cdot f(x)) = O(f(x))$, $\forall k \neq 0$.
5. $O(f(x)) + O(g(x)) = O(\max(f(x), g(x)))$

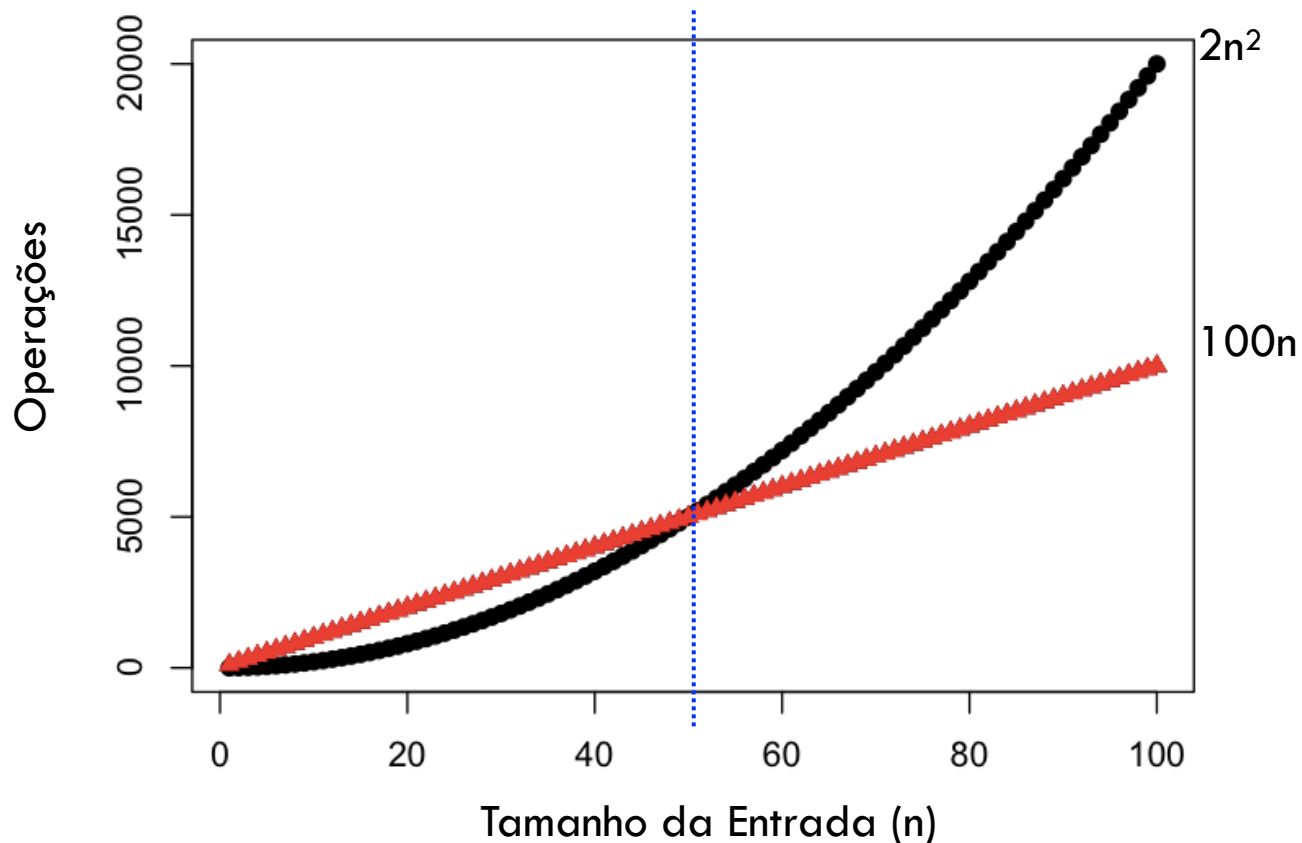
Complexidade Assintótica

- Se f é uma função de complexidade para o algoritmo A , então $O(f)$ é considerada a complexidade assintótica do algoritmo A .

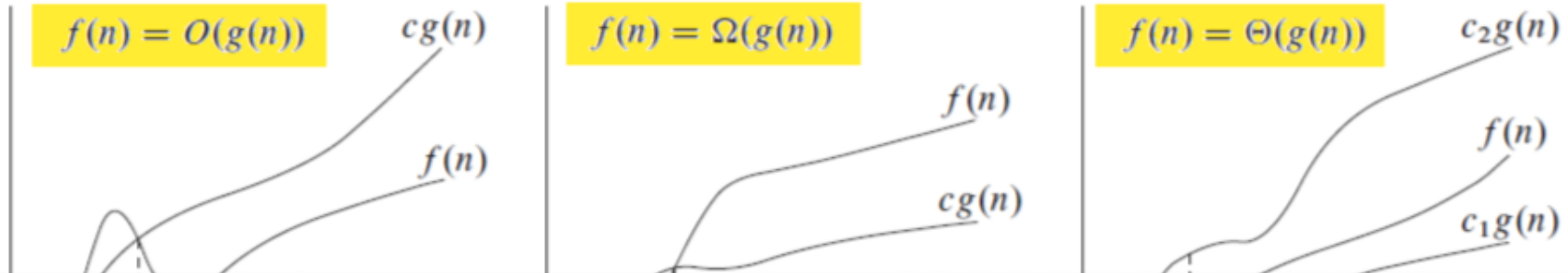


Exemplo 03: influência das constantes

- Exemplo: $f(n) = 100n$, $g(n) = 2n^2$



Notações Assintóticas



- Propriedades permitem desprezar constantes e termos de menor grau.
- Possível definir algumas regras p análise de limites superiores.

1. Todas são reflexivas e transitivas; Θ é simétrica.
2. $f(x) = O(g(x))$ sse $g(x) = \Omega(f(x))$
3. $f(x) = \Theta(g(x))$ sse $f(x) = O(g(x))$ e $f(x) = \Omega(g(x))$
4. $O(k \cdot f(x)) = O(f(x))$, $\forall k \neq 0$.
5. $O(f(x)) + O(g(x)) = O(\max(f(x), g(x)))$

Princípios sobre a notação O

Regra	Tipo de comando	Tempo
1	Atribuição/leitura/escrita	constante = $O(1)$
2	Sequência de comandos	maior tempo entre os comandos
3	Comando condicional	tempo dos comandos dentro da condição + $O(1)$. Se houver se-senão, é $\max(\text{teste} + \text{se}, \text{teste} + \text{senão})$.
4	Laço	soma do tempo do corpo do laço, mais o tempo de avaliar a condição de parada, multiplicado pelo número de iterações.
5	Procedimentos	tempo de cada procedimento computado separadamente. Iniciando dos que não tem outras chamadas de funções, depois os que tem chamada, até chegar ao programa principal.

Exemplo 04 - Loop simples

```
1  for (i = sum = 0; i < n; i++) {  
2      sum = sum + a[i]  
3  }  
4
```

- Considerando apenas atribuições (**DROZDEK, 2016**):
 - 2 atribuições antes do comando iniciar ($i = 0, \text{sum} = 0$) $\rightarrow 2$
 - dentro do comando, repete-se n vezes, e cada laço executa também duas atribuições $\rightarrow 2n$
 - total = $2n + 2$ operações
 - complexidade assintótica = **$O(n)$**

Exemplo 05 - Loop aninhado

```
1  for(i = 0; i < n; i++) {  
2      for(j = 1, soma = a[0]; j <= i; j++){  
3          soma = soma + a[j];  
4      }
```

- variável i iniciada $\rightarrow 1$
- laço externo executa n vezes, e em cada iteração é realizada a atribuição de três variáveis $\rightarrow 3n$
- laço interno é efetuado j vezes, com $j \in \{1, 2, 3, \dots, n-1\}$, com duas atribuições $\rightarrow \sum_{i=1}^{n-1} 2i = n(n-1)$
- total = $1 + 3n + n^2 - n$
- complexidade assintótica = $O(n^2)$

Roteiro

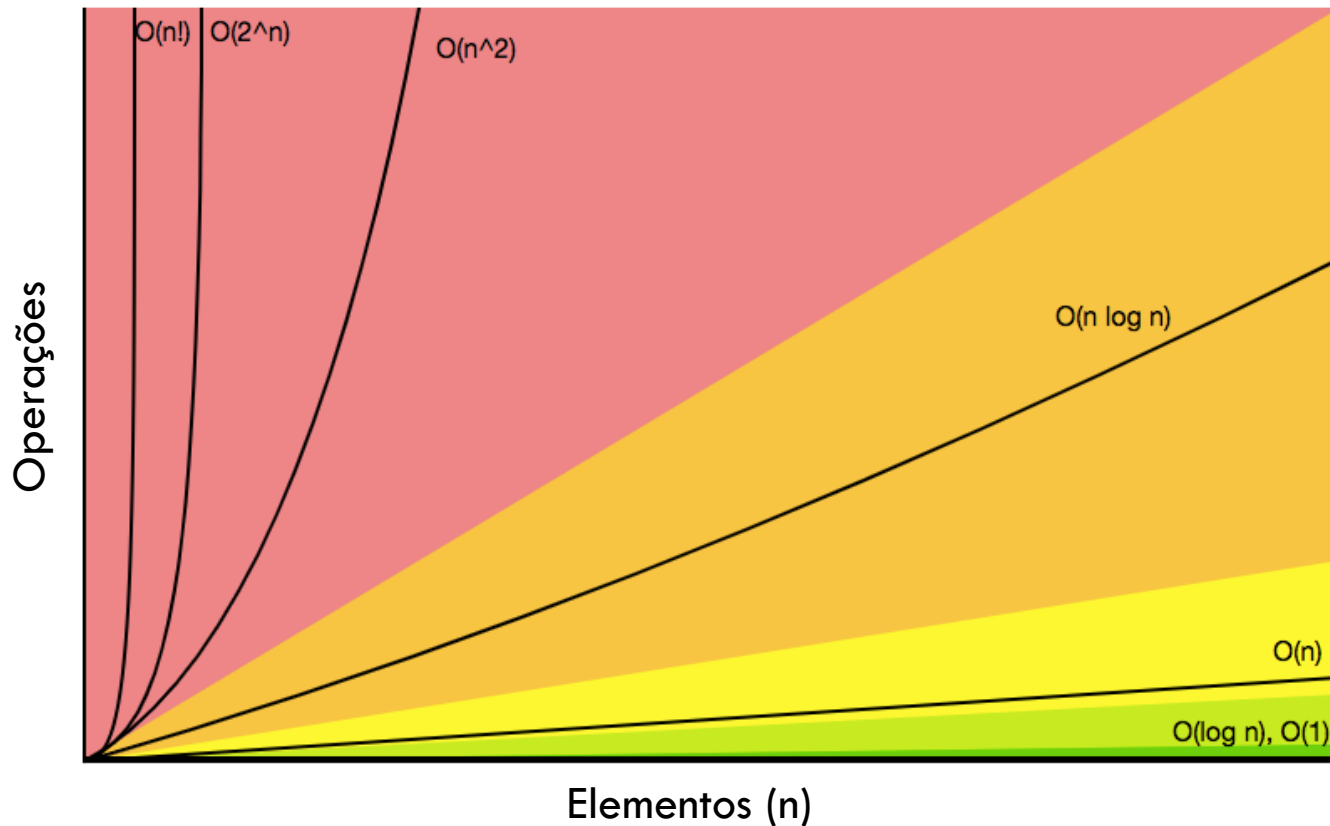
- 1 Introdução / Motivação
- 2 Análise de Complexidade / Assintótica
- 3 Notações Assintóticas
- 4 Classes de Complexidade
- 5 Síntese / Revisão
- 6 Referências

Classes de Complexidades

Complexidade	Nome	Exemplo
$O(1)$	Constante	Expressões e atribuições inteiras e reais
$O(\log n)$	Logarítmica	Busca binária
$O(n)$	Linear	Busca Sequencial
$O(n \log n) = O(\log n!)$	Quase Linear	Métodos de Ordenação eficientes
$O(n^c)$	Polinomial	Métodos de Ordenação Simples
$O(c^n), c > 1$	Exponencial	Todas as combinações de elementos
$O(n!)$	Fatorial	Todas as permutações de elementos

Classes de Complexidades

Hierarquia na complexidade dos algoritmos



Fonte: <http://bigocheatsheet.com>

Deficiências da Análise Assintótica

- Complexidade de Código
 - Algoritmos melhores são frequentemente mais complexos.
 - Exige mais tempo no desenvolvimento.
- Tamanhos de entrada pequenos
 - Análise assintótica ignora tamanhos pequenos.
 - Pode ocorrer de constantes ou termos de menor ordem dominarem o tempo total, fazendo B ser melhor que A.

Roteiro

- 1 Introdução / Motivação
- 2 Análise de Complexidade / Assintótica
- 3 Notações Assintóticas
- 4 Classes de Complexidade
- 5 Síntese / Revisão
- 6 Referências

Síntese/Revisão da Aula

- Noções Básicas de Complexidade de Algoritmos
- Medidas genéricas para avaliar o custo de algoritmos
- Análise de complexidade → análise assintótica
- Notações assintóticas → O , Ω e Θ
- Limites superiores (O)
- Propriedades → simplificar a análise (regras)
- Classes de complexidade

Próximas Aulas

- Implementação de Listas Elementares
 - Filas
 - Pilhas
 - Listas
- Árvores de Busca (Árvores)
- ...

Próximas Aulas

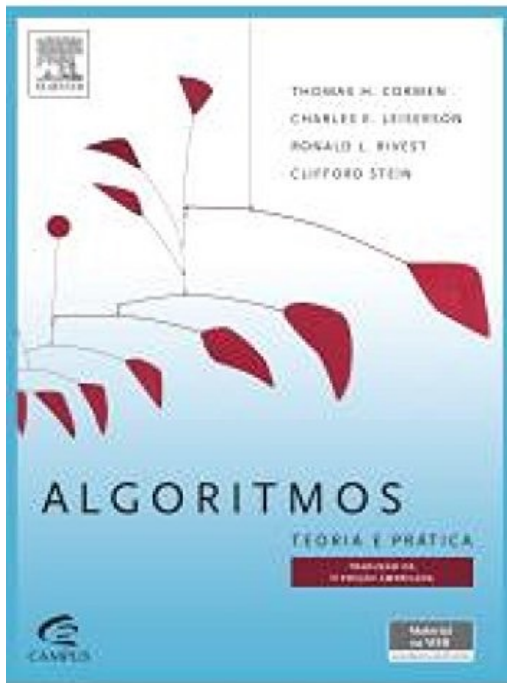
Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Queue</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Singly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Doubly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Skip List</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n \log(n))$
<u>Hash Table</u>	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Binary Search Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Cartesian Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>B-Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Red-Black Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Splay Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>AVL Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>KD Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$

Roteiro

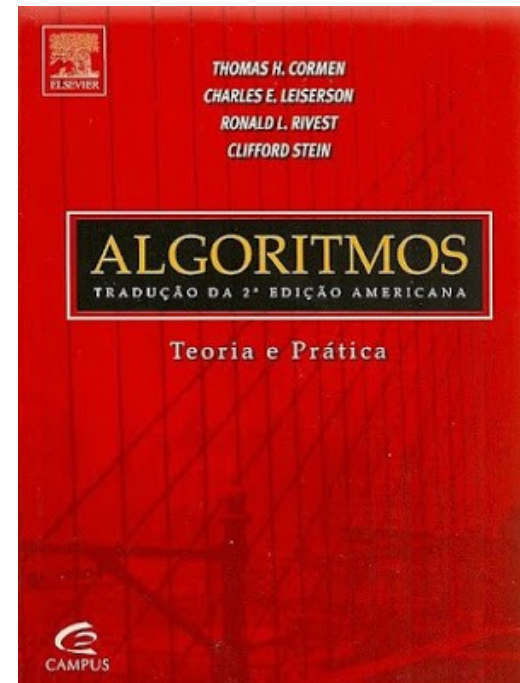
- 1 Introdução / Motivação
- 2 Análise de Complexidade / Assintótica
- 3 Notações Assintóticas
- 4 Classes de Complexidade
- 5 Síntese / Revisão
- 6 Referências

Referências



[Cormen et al, 2018]

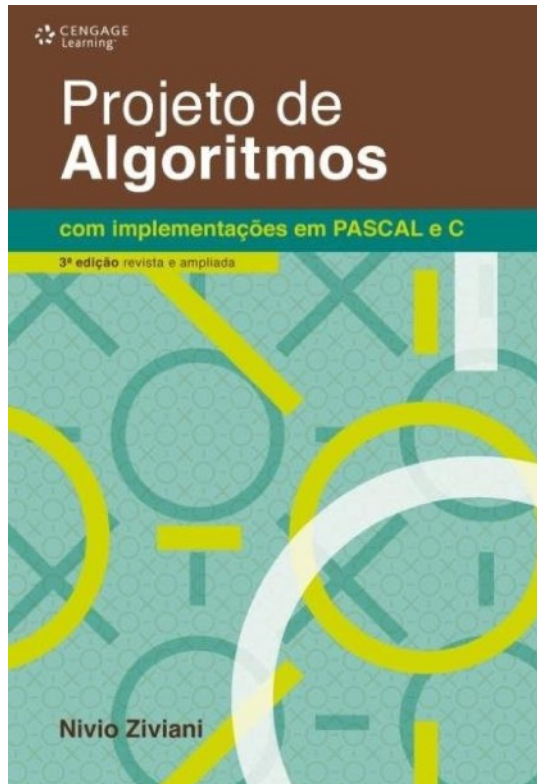
3 edição



[Cormen, 2012]

2 edição

Referências



[Ziviani, 2010]



[Drozdek, 2017]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br

Exercício 01: O?

Qual a classe de complexidade do algoritmo?

```
1  int fun(int n) {  
2      int count = 0;  
3      for (int i = n; i > 0; i /= 2)  
4          for (int j = 0; j < i; j++)  
5              count += 1;  
6      return count;  
7  }
```