Complexidade de Algoritmos

Prof. Alberto Costa Neto

Definição de Algoritmo

"É qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída."

Cormen (2002)

Definição de Estrutura de Dados

"É um meio para armazenar e organizar dados com o objetivo de facilitar o acesso e as modificações."

Cormen (2002)

Algoritmos e Estruturas de Dados

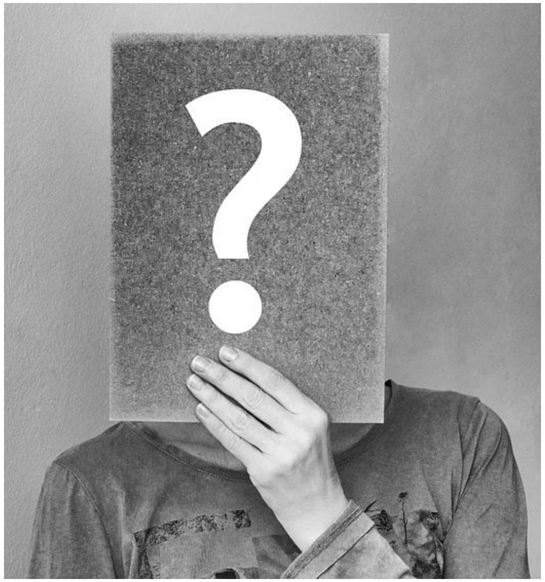
- Um algoritmo é projetado em função de tipos abstratos de dados.

- As EDs diferem umas das outras pela disposição ou manipulação de seus dados
  - Não se pode separar as EDs e os algoritmos associados a elas.

- A escolha de uma ED deve ser orientada pela eficiência dos algoritmos de suas operações.

O Problema

- Ao criar um algoritmo, **como saber se é bom**?
- **Como comparar com outros algoritmos**?



Análise de Algoritmos

Análise de Algoritmos

- Ao criar um algoritmo, como saber se é bom?
- Como comparar com outros algoritmos?

Qual critério devo usar?

- **Uso de memória?**
- **Uso da CPU?**

Análise de Algoritmos

"Analisar um algoritmo significa prever os recursos de que ele necessitará."

Cormen (2002)

"Em geral, memória, largura de banda de comunicação ou hardware de computação são a preocupação primordial, mas frequentemente é o tempo de computação que se deseja medir."

Cormen (2002)

Análise de Algoritmos

- **Análise de um algoritmo particular**
  - Determinar quanto cada parte do algoritmo será executada
  - Calcular quanto de memória será necessária

- **Análise de uma classe de algoritmos**
  - Considerando um problema particular
  - Determinar aquele de menor custo para resolvê-lo

Medição do custo/tempo de execução

- Implementar o algoritmo e **realizar um experimento controlado:**
    - Usando um computador real
    - Utilização do mesmo interpretador ou compilador
    - Utilizar uma boa base de dados de teste

RAM (Random Access Machine)

- Outra forma de medir o custo é por meio do uso de um **modelo matemático** ou modelo de computação genérico com um único processador, a **RAM (*Random Access Machine* – Máquina de Acesso Aleatório)**.

- As instruções são executadas de forma sequencial (sem concorrência)

O Modelo de RAM

- Contém instruções existentes nos computadores reais:
  - Instruções aritméticas (soma, subtração, multiplicação, divisão, piso, teto, resto)
  - Instruções de controle (decisão, chamada e retorno de funções)
- Normalmente neste modelo considera-se que as instruções demoram um tempo constante.
- As instruções são executadas de forma sequencial (sem concorrência)

Função de custo T(n)

- T(n) é a **função de complexidade de tempo** do algoritmo.

  Quando T(n) é a medida do tempo necessário para executar um algoritmo para um problema de tamanho n.

- T(n) é a **função de complexidade de espaço** do algoritmo.

  Quando T(n) é a medida de memória necessária para executar um algoritmo para um problema de tamanho n.

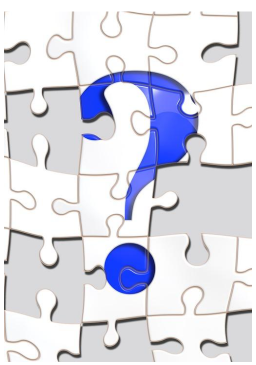
Exemplo de Função de custo T(n)

- Obtenção do **menor valor em um vetor**

```
int calculamenor (int A[], int n)
{
        int i, menor;

        menor = A[0];
        for (i = 1; i < n; i++)
        {
                if (A[i] < menor)
                        menor = A[i];
        }
        return menor;
}
```

Qual é o trecho de código mais executado (mais representativo)?



Exemplo de Função de custo T(n)

- Obtenção do **menor valor em um vetor**

```
int calculamenor (int A[], int n)
{
        int i, menor;

        menor = A[0];
        for (i = 1; i < n; i++)
        {
                if (A[i] < menor)
                        menor = A[i];
        }
        return menor;
}
```

+executado

Exemplo de Função de custo T(n)

- Obtenção do menor valor em um vetor
- Comparação do primeiro valor com todos os demais (n-1 comparações)
- A função de custo seria:

$$T(n) = n - 1$$

Computando o tempo de execução

- **Comandos de atribuição, leitura ou escrita:** são considerados O(1);
- **Sequência de comandos:** pelo maior tempo de qualquer comando da sequência;
- **Comando de decisão:** tempo de avaliação da condição O(1) mais o tempo dos comandos dentro do comando condicional;
- **Comando de repetição:** tempo de execução do corpo mais o tempo de avaliar a condição de término, multiplicado pelo número de iterações.
- **Procedimentos/Funções:** computados separadamente, começando pelos que não chamam outros.

Outro exemplo: Busca Sequencial

- Neste caso, o algoritmo **não se comporta de maneira uniforme**.

- Pode ser que **o valor buscado esteja em qualquer lugar do vetor**, fazendo o tempo variar de 1 a n.

30	15	20	8	3	17	25	90	55	43

Outro exemplo: Busca Sequencial

- Podemos identificar 3 casos:
  - Melhor caso: valor procurado está na **primeira** posição (30).
  - Pior caso: valor procurado está na **última** posição (43).
  - Caso médio: valor procurado no **meio** (3).

30	15	20	8	3	17	25	90	55	43

Busca Sequencial: Tempo de execução

- **Melhor caso:** `T(n) = 1`
- **Pior caso:** `T(n) = n`
- **Caso médio:** `T(n) = (n+1)/2`

Busca Sequencial: Calculando T(n) do caso médio

- Considerando **p_i** a probabilidade de se encontrar o valor na **posição i** e que ao encontrá-lo na posição em i realizou-se **i comparações**, temos:

$$T(n) = 1 \cdot p_1 + 2 \cdot p_2 + 3 \cdot p_3 + \ldots + n \cdot p_n$$

$$T(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + 3 \cdot \frac{1}{n} + \ldots + n \cdot \frac{1}{n}$$

$$T(n) = \frac{1}{n} \cdot (1 + 2 + 3 + \ldots + n) \qquad \text{(PA)}$$

$$T(n) = \frac{1}{n} \cdot \left(\frac{n \cdot (n+1)}{2}\right)$$

$$T(n) = \frac{n+1}{2}.$$

$$S_n = \frac{(a_1 + a_n)n}{2}$$

posição do termo

soma dos **n** termos da P.A.

primeiro termo da P.A.

ocupa a enésima posição na sequência

Análise Assintótica

Eficiência assintótica

- **Para pequenos valores de n**, a eficiência do algoritmo não afeta muito o resultado.
  - Às vezes é mais produtivo usar um algoritmo simples.

- **Para altos valores de n**, estuda-se o comportamento assintótico da função de complexidade de tempo dos algoritmos.
  - Dizemos que f é assintoticamente menor que g se f(V) < g(V) para todos os valores suficientemente grandes de V

ASSÍNTOTA

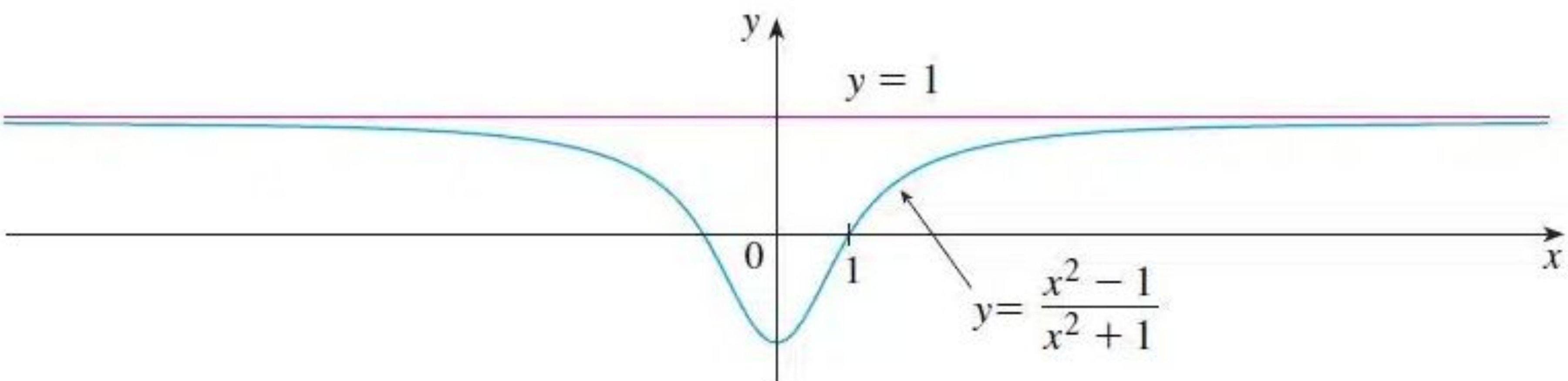
- É um termo com origem num vocábulo grego que **faz referência a algo que não tem coincidência**.

- O conceito é usado no âmbito da geometria para designar uma reta que, ao se prolongar de forma indefinida, **tende a se aproximar de uma certa curva ou função, embora sem alcançá-la**.

Exemplo de Assíntota

$$f(x) = \frac{x^2 - 1}{x^2 + 1}$$

x	$f(x)$
0	−1
±1	0
±2	0,600000
±3	0,800000
±4	0,882353
±5	0,923077
±10	0,980198
±50	0,999200
±100	0,999800
±1000	0,999998



Fonte: https://breakthescience.com.br/limites-no-infinito-assintotas-horizontais/

Notações Assintóticas

- Utilizadas para representar o comportamento assintótico das funções de complexidade de tempo de algoritmos

- Definidas em termos de funções cujo domínio são os números naturais N = {0,1,2,3…}

Análise Assintótica

Definição:

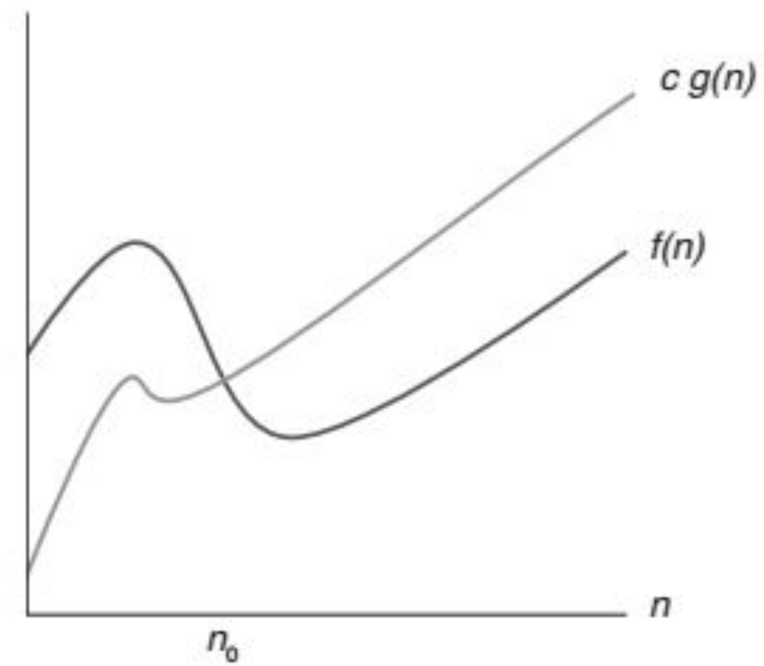
Uma função ***g(n) domina assintoticamente uma função f(n)*** se existem duas constantes positivas c e n_0 tais que, para n >= n_0, temos que:

$$|f(n)| < c \cdot |g(n)|$$

Análise Assintótica

Definição:

Uma função ***g(n) domina assintoticamente uma função f(n)*** se existem duas constantes positivas c e n_0 tais que, para $n \geq n_0$, temos que:



$$|f(n)| \leq c \cdot |g(n)|$$

Exemplo 1

Considere $f(n) = n$ e $g(n) = -n^2$

Verifica-se que $g(n)$ domina assintoticamente $f(n)$, já que $|n| \leq c \cdot |-n^2|$ para todo $n \in N$.

Considerando $c = 1$ e $n_0 = 1$, temos que:

$$|n| \leq 1 \cdot |-n^2|$$

n	$\|n\| \leq c \cdot \|-n^2\| \; (c = 1)$
1	$1 \leq 1$
2	$2 \leq 4$
3	$3 \leq 9$
4	$4 \leq 16$
...	...

Exemplo 2

Considere $f(n) = (n+1)^3$ e $g(n) = n^3$.

Temos que $g(n)$ domina assintoticamente $f(n)$. Com $c = 3$ e $n_0 = 3$, tem-se que $|(n+1)^3| \leq 3 \cdot |n^3|$, para todo $n \geq 3$.

$$|(n+1)^3| \leq 3 \cdot |n^3|$$

n	$\|(n+1)^3\| \leq c \cdot \|n^3\|, (c = 3)$
3	$64 \leq 81$
4	$125 \leq 192$
5	$216 \leq 375$
...	...

Desafio (em sala)

Considere $f(n) = (n+1)^3$ e $g(n) = n^3$.

É possível afirmar que f(n) domina assintoticamente g(n)?

$$|n^3| \leq c \cdot |(n+1)^3|$$

Encontre valores para c e n_0. Podem existir várias combinações, mas é suficiente encontrar apenas um exemplo.

Desafio (Resposta)

Considere $f(n) = (n+1)^3$ e $g(n) = n^3$.

É possível afirmar que $f(n)$ domina assintoticamente $g(n)$?

$$|n^3| \leq c \cdot |(n+1)^3|$$

Resposta: Prova-se que f(n) domina assintoticamente g(n) para **$c = 1$ e $n_0 = 1$.**

Big O

Notação O (Big O)

- Uma função $f(n)$ é $O(g(n))$ se existem duas constantes positivas c e n_0 tais que:

  $$f(n) \leq c \cdot g(n), \text{ para todo } n \geq n_0$$

- Pode-se representar também que $f(n) = O(n^2)$ ou $f(n) \in O(n^2)$.

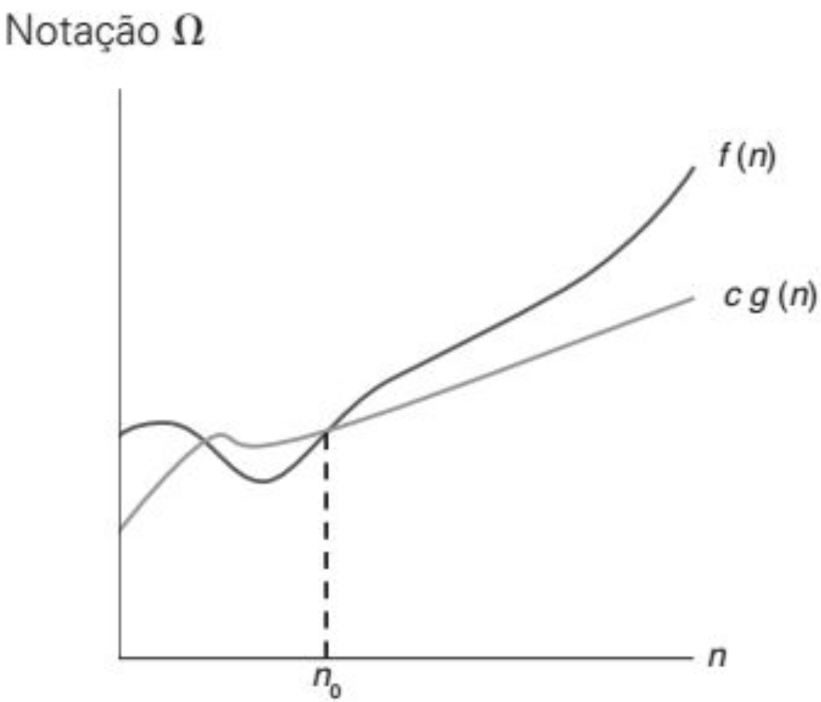
- A notação O dá um **limite assintótico superior** sobre uma função, dentro de um fator constante.

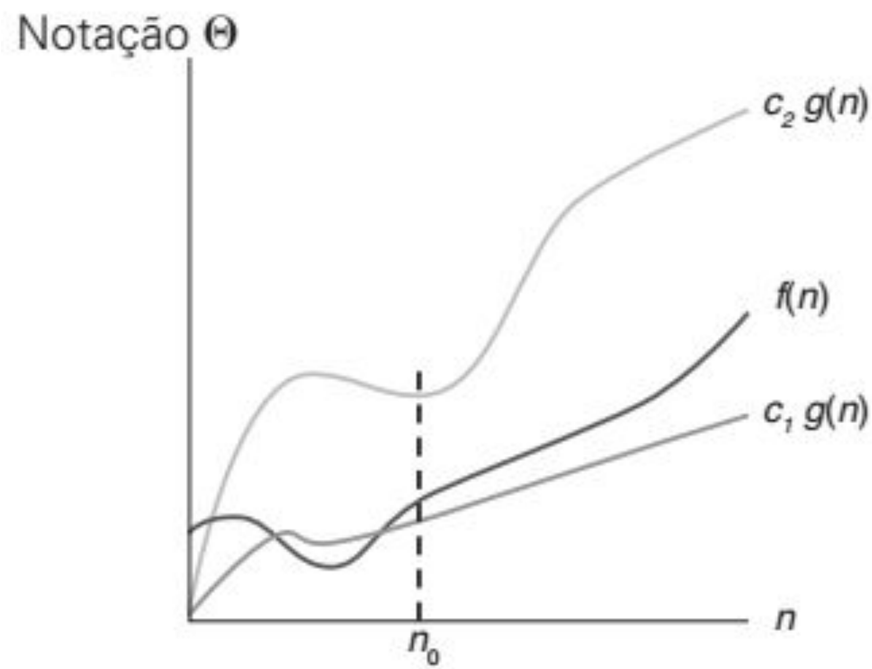
Notação O (Big O)

- Quando se afirma que o tempo de execução de um algoritmo é $O(n^2)$, significa que **existe uma função $f(n)$ que é $O(n^2)$** tal que, **para qualquer entrada de tamanho n, o tempo de execução** sobre ela **tem um limite superior determinado pelo valor $c \cdot n^2$.**

Outras notações

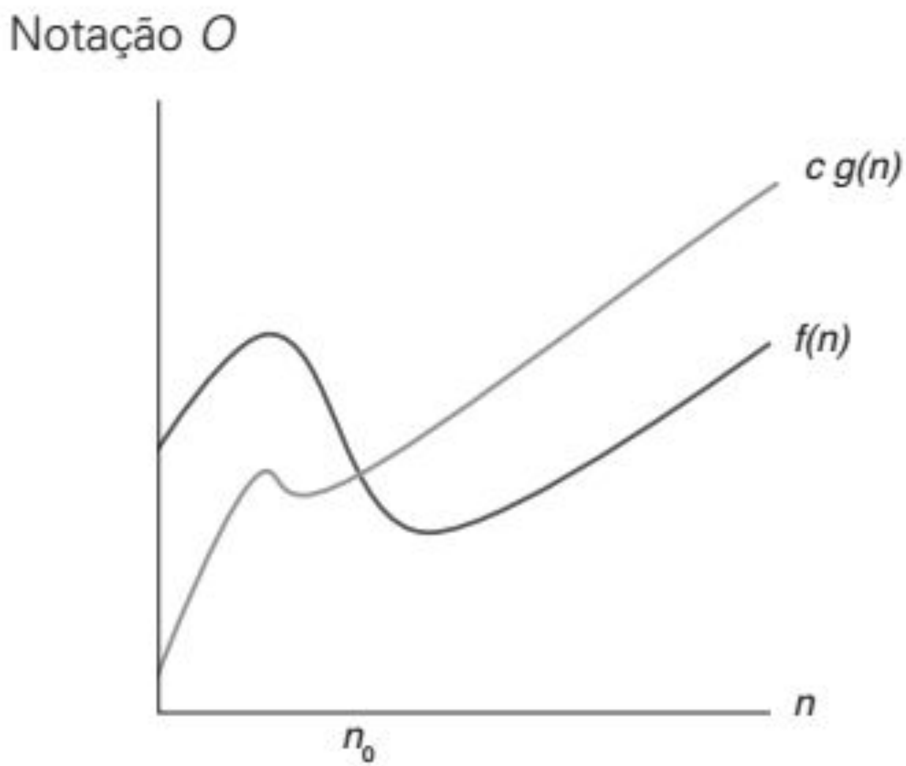
- **Existem outras notações**, que determinam o *limite assintótico inferior* (ômega), *limite assintótico firme* (theta).





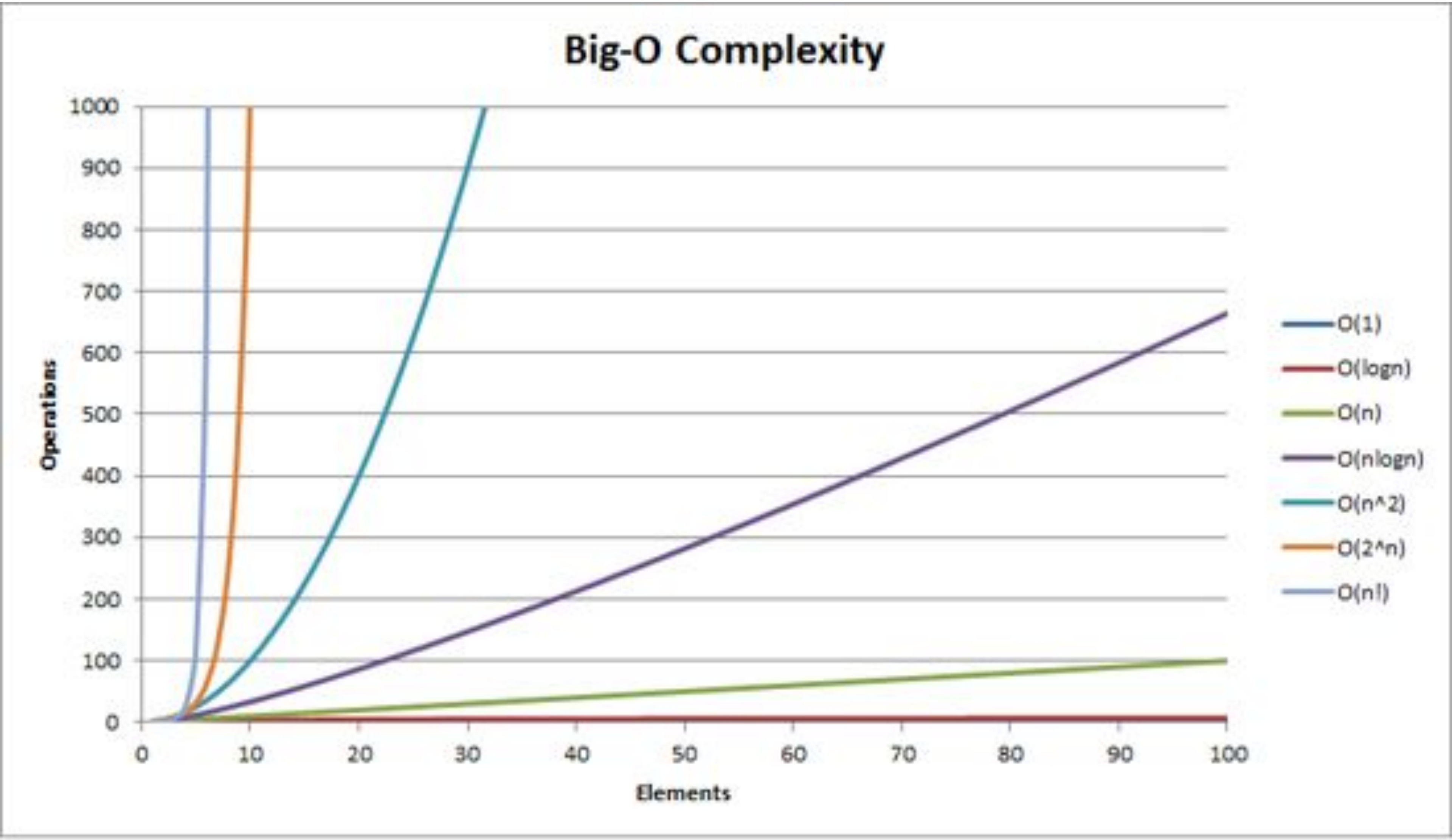
NOTAÇÃO ADOTADA

- **Utilizaremos a notação O** porque desejamos analisar o *limite assintótico superior*.

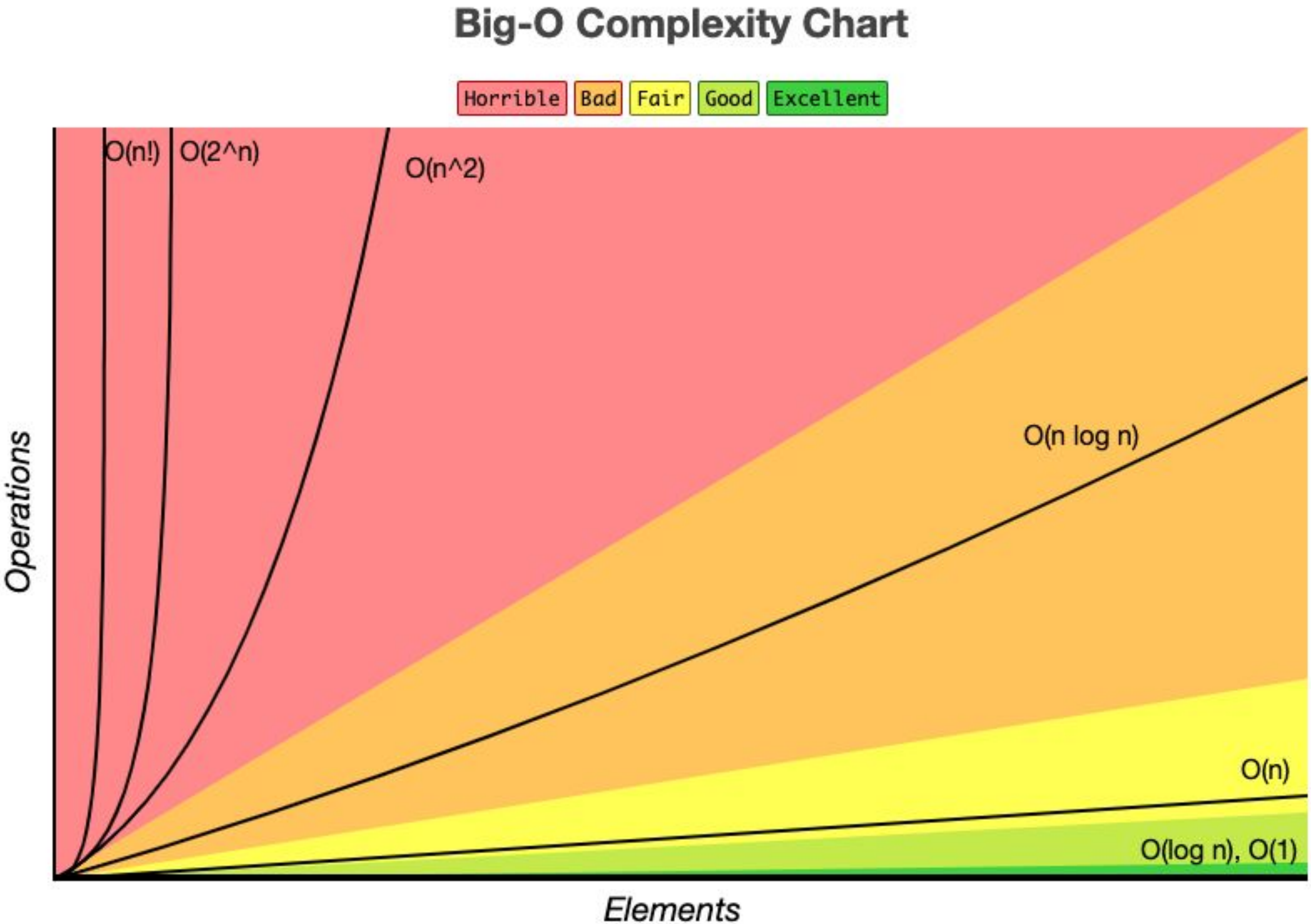


Comparativo numérico de Big O's

N	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(2^n)	O(n!)
1	1	0,00	1	0,00	1	2	1
2	1	1,00	2	2,00	4	4	2
3	1	1,58	3	4,75	9	8	6
4	1	2,00	4	8,00	16	16	24
5	1	2,32	5	11,61	25	32	120
6	1	2,58	6	15,51	36	64	720
7	1	2,81	7	19,65	49	128	5.040
8	1	3,00	8	24,00	64	256	40.320
9	1	3,17	9	28,53	81	512	362.880
10	1	3,32	10	33,22	100	1.024	3.628.800
100	1	6,64	100	664,39	10.000	1,26765E+30	9,3326E+157
1.000	1	9,97	1.000	9.965,78	1.000.000	1,0715E+301	
10.000	1	13,29	10.000	132.877,12	100.000.000		
100.000	1	16,61	100.000	1.660.964,05	10.000.000.000		



Fonte: https://stackoverflow.com/questions/7830727/n-log-n-is-on



Fonte: https://www.bigocheatsheet.com

Exemplos de Complexidade de algoritmos

- O(**1**)
  - Acesso a elemento de array
  - Acesso a primeiro nó de lista encadeada

- O(**log n**)
  - Busca binária
  - Busca de valor em árvore AVL

- O(**n**)
  - Encontrar menor ou maior em um array ou lista encadeada
  - Acesso ao último nó de uma lista encadeada

Exemplos de Complexidade de algoritmos

- O(**n log n**)
  - Merge Sort

- O(**n^2**) - quadrática
  - Bubble Sort

- O(**n!**) - fatorial
  - Torre de Hanoi (recursivo)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Fonte: https://www.bigocheatsheet.com

Common Data Structure Operations

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

Fonte: https://www.bigocheatsheet.com

Sugestões de Estudo

Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++ (Ascencio, Ana Fernanda Gomes; Araújo, Graziela Santos)

- Capítulo 1

Projeto de Algoritmos com implementações em Java e C++ (Nivio Ziviani)

- Seções 1.1 a 1.4

https://www.bigocheatsheet.com