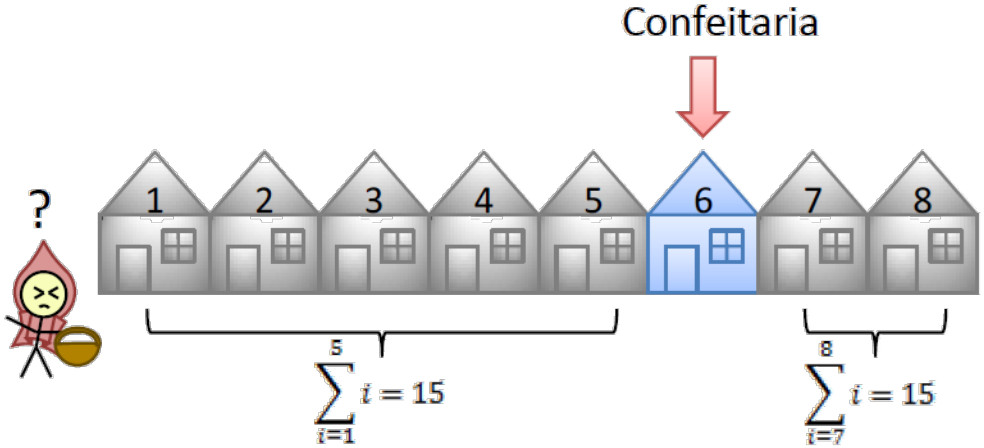
Complexidade de Algoritmos e Algoritmos de Ordenação - Introdução

por Rossana B Queiroz

Motivação

- Problema: “A Rua Encantada”



- Algoritmos de ordenação:
  - Bubblesort vs. Quicksort

Introdução a Complexidade de Algoritmos

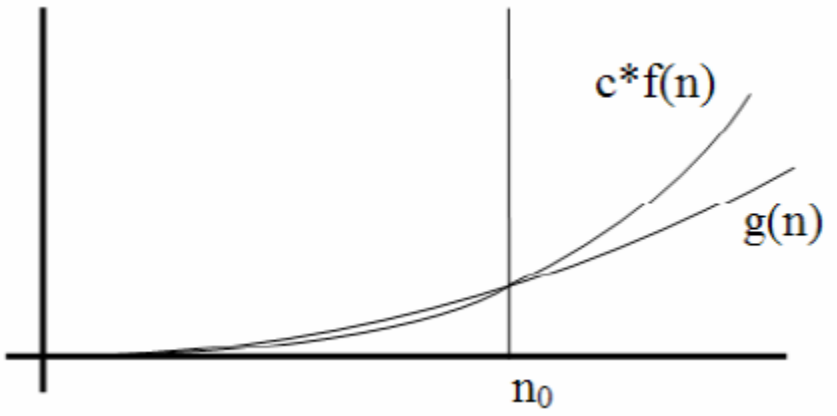
- Como podemos comparar se um algoritmo é mais “rápido” do que outro?
- Como podemos determinar se um algoritmo necessita de “menos memória” para armazenar seus dados do que outro?
- A análise da complexidade de algoritmos pode nos auxiliar a responder a estas questões.

Notação Big-O

- A notação Big-O de um algoritmo é uma função que expressa o comportamento do algoritmo quando ele é utilizado com conjuntos de dados de diferentes tamanhos.
  - O(função)
  - n – número de elementos
  - c – valor constante
- Exemplo
  - O(1)
  - O(n)
  - O($\log_2 n$)

Notação Big-O

- Diz-se que uma função **g(n)** é **O(f(n)),** notando-se **g = O(f(n))** se existir alguma constante c > 0 e um inteiro n_0, tal que **n > n_0** implica **g(n) <= c* f(n).**



Notação Big-O

- Na análise de um algoritmo podemos estar interessados no:
  - Melhor caso;
  - Caso médio;
  - Pior caso.
- Conhecer o comportamento do algoritmo no seu pior caso costuma ser mais útil para as análises de performance
  - Melhor caso somente ocorre em situações ótimas, o que é raro.
  - Pior caso ... Ocorre no pior caso !
    - Murphy's Law Rules! ☺

Notação Big-O

- Considere um vetor com 10 posições contendo números inteiros. Você quer encontrar o índice do vetor, começando a partir do índice zero, que contém o valor 20.
  - Qual o melhor caso?
  - Qual o caso médio?
  - Qual o pior caso?

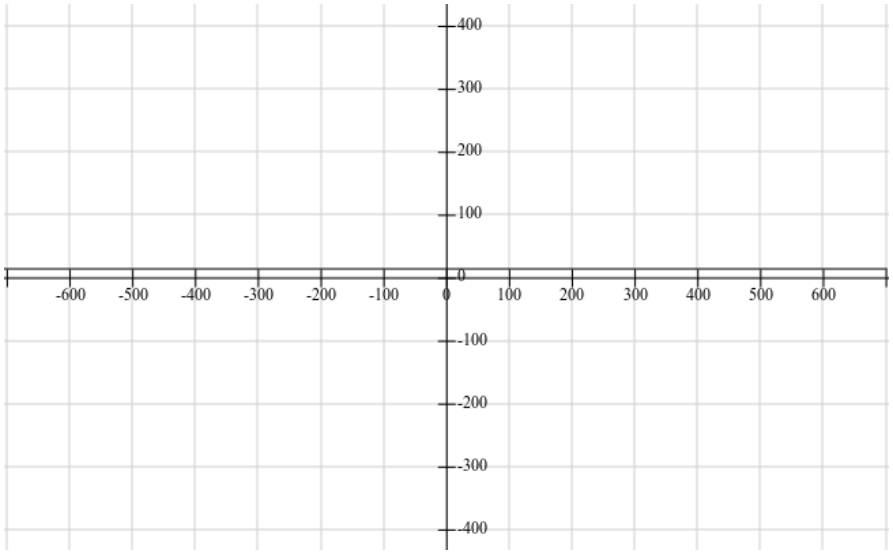
Exemplo

Instrução	Custo	Iterações
`for(i = 0; i < 10; i++)`	c1	n+1 (11)
`{`		
`if(vetor[i] == 20)`	c2	n
`Encontrou;`	c3	1
`Break;`	c4	1
`}`		

- T(n) = c1*n + c2*n + c3 +c4 = a*n + b
- O(n)

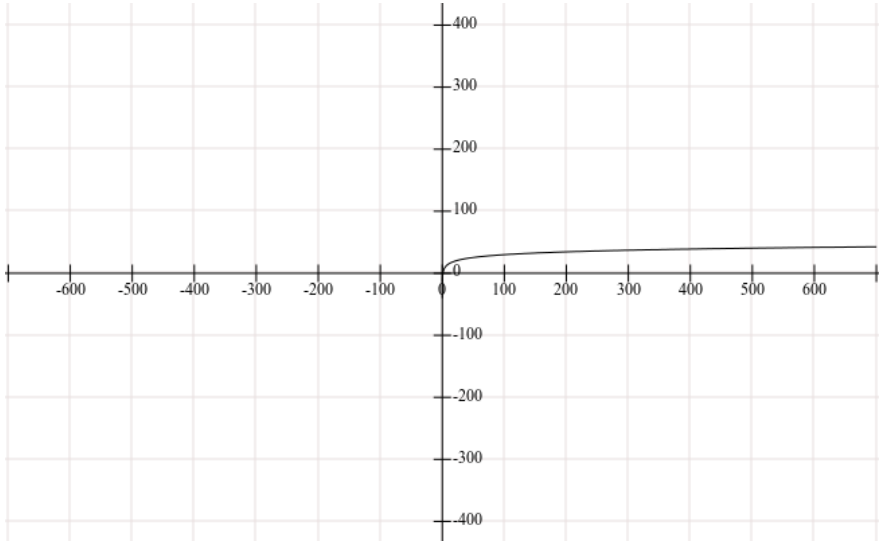
Complexidades mais Comuns

- O(c) – constante
  - Complexidade não depende da quantidade de dados.
  - É o algoritmo mais eficiente.



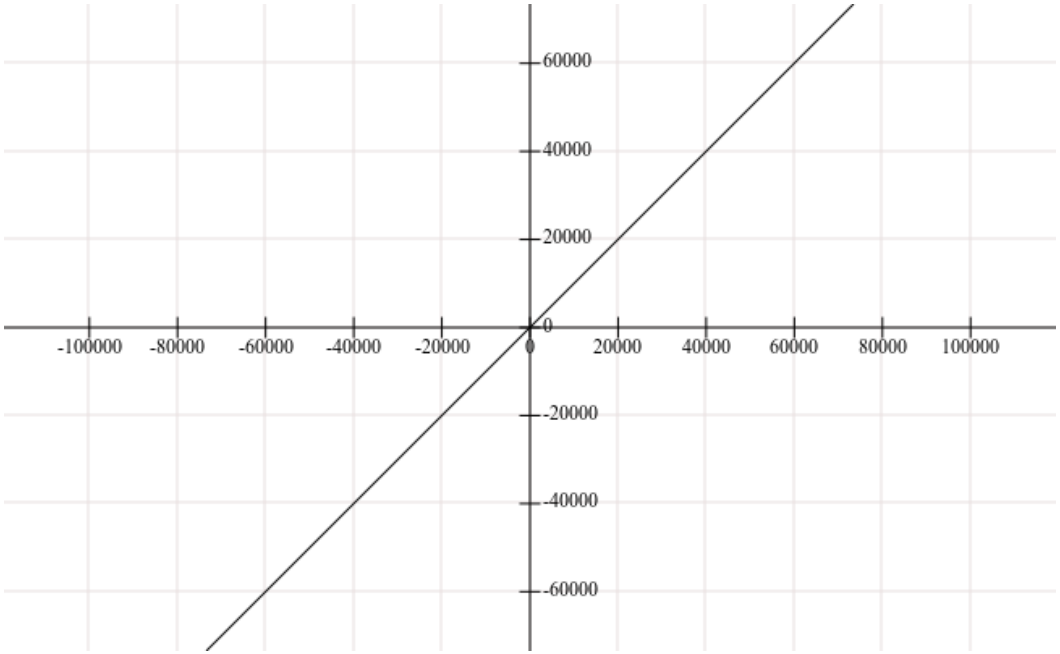
Complexidades mais Comuns

- $O(\log_2 n)$
  - É o segundo tipo de algoritmo mais eficiente
  - O algoritmo apresente um comportamento mais eficiente conforme mais dados são adicionados



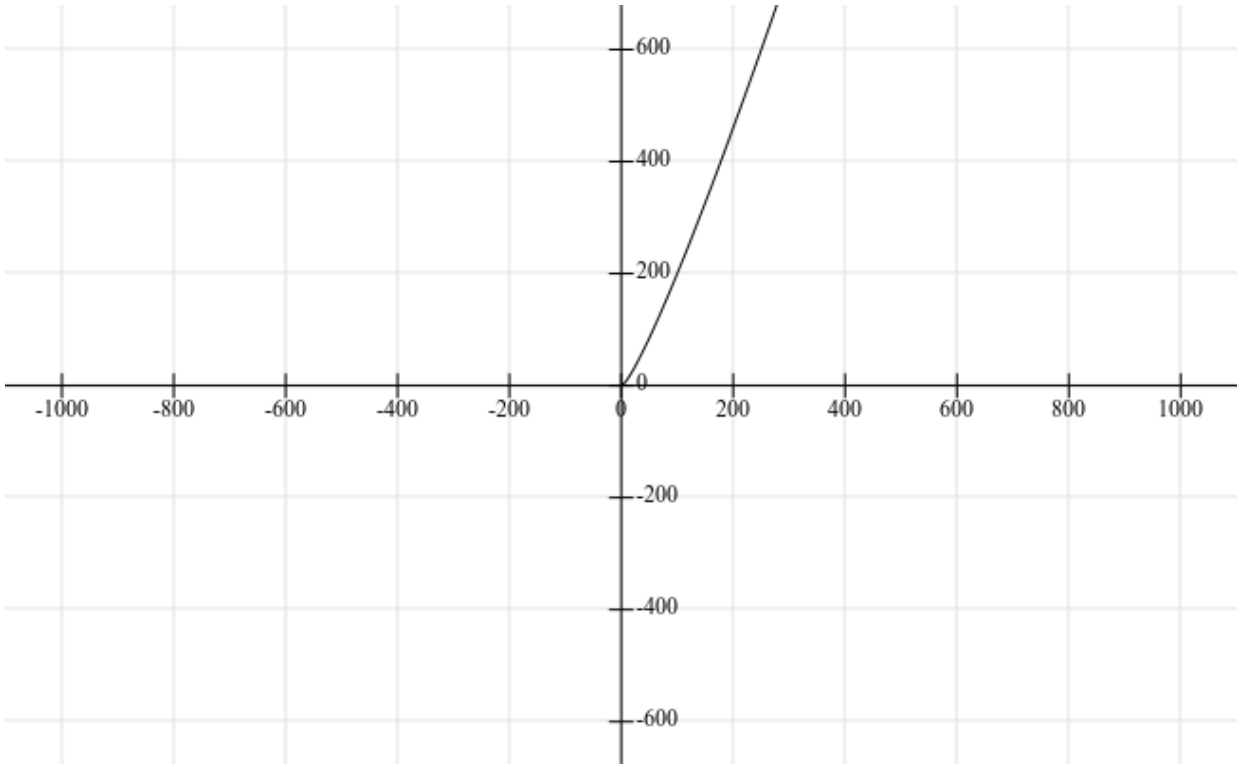
Complexidades mais Comuns

- O(n) – é uma função linear da quantidade de dados



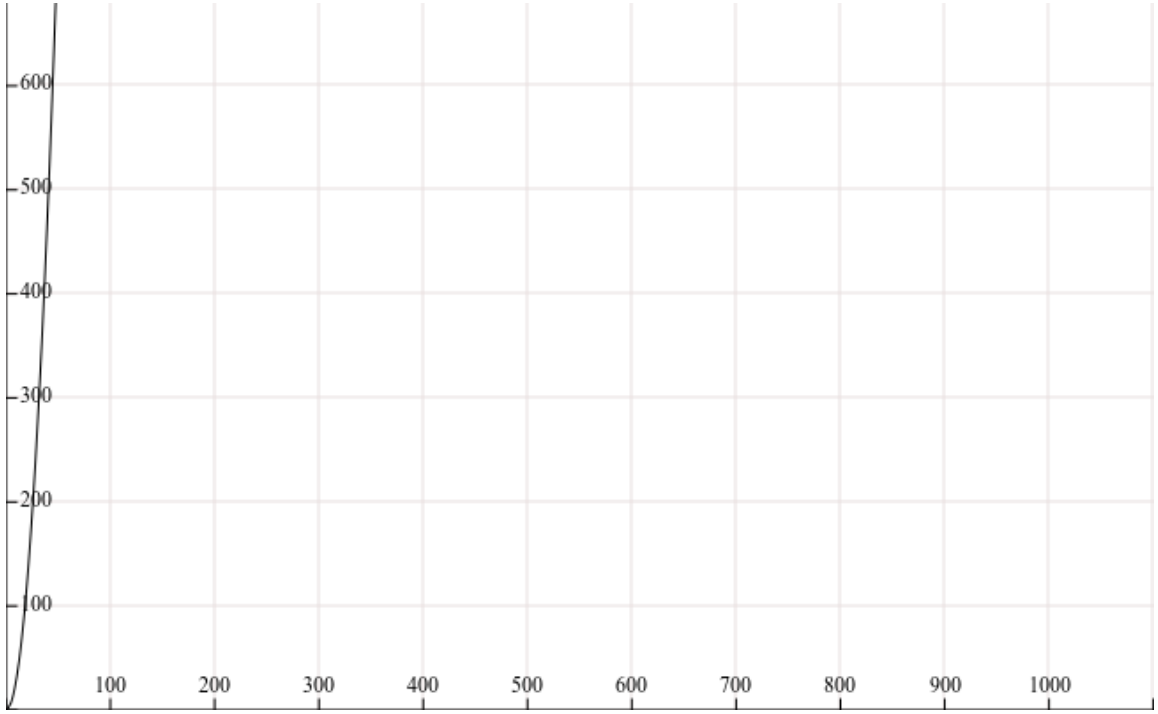
Complexidades mais Comuns

- $O(n\log_2 n)$



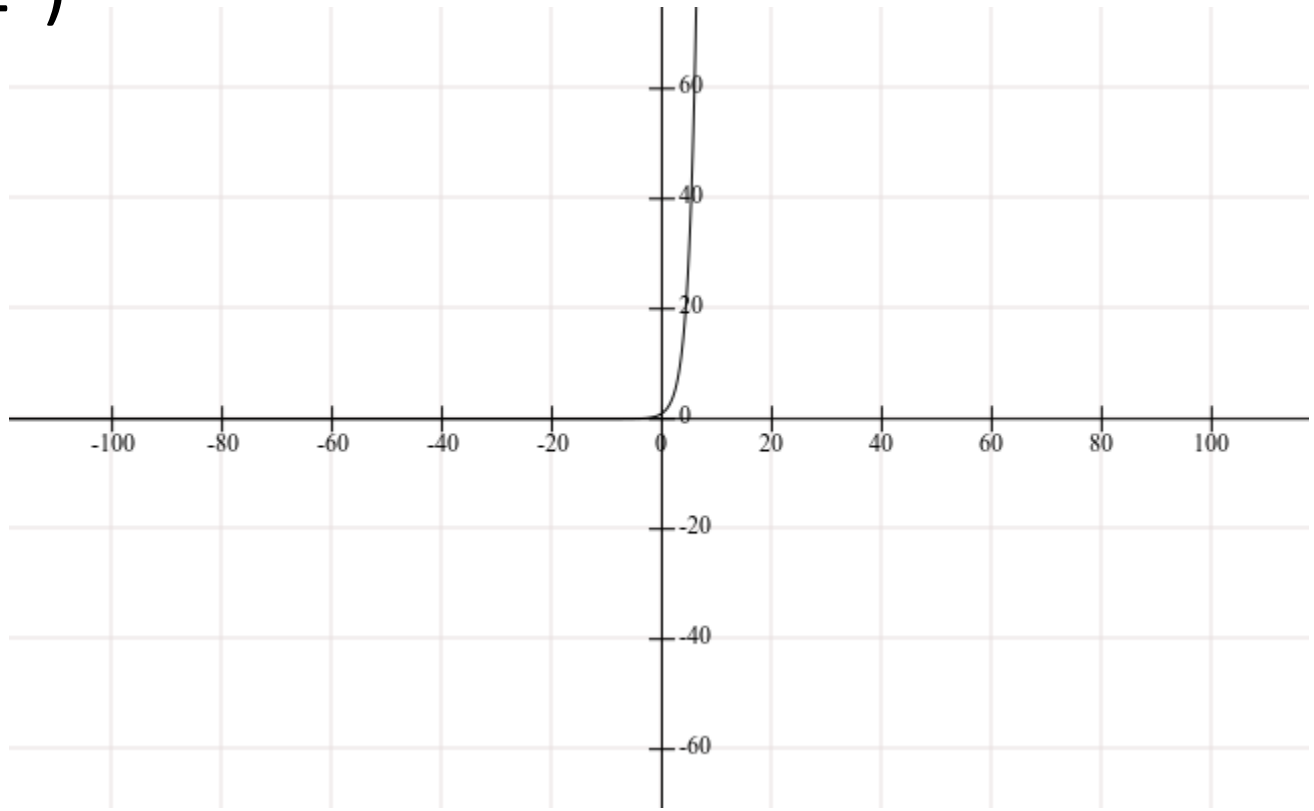
Complexidades mais Comuns

- $O(n^2)$

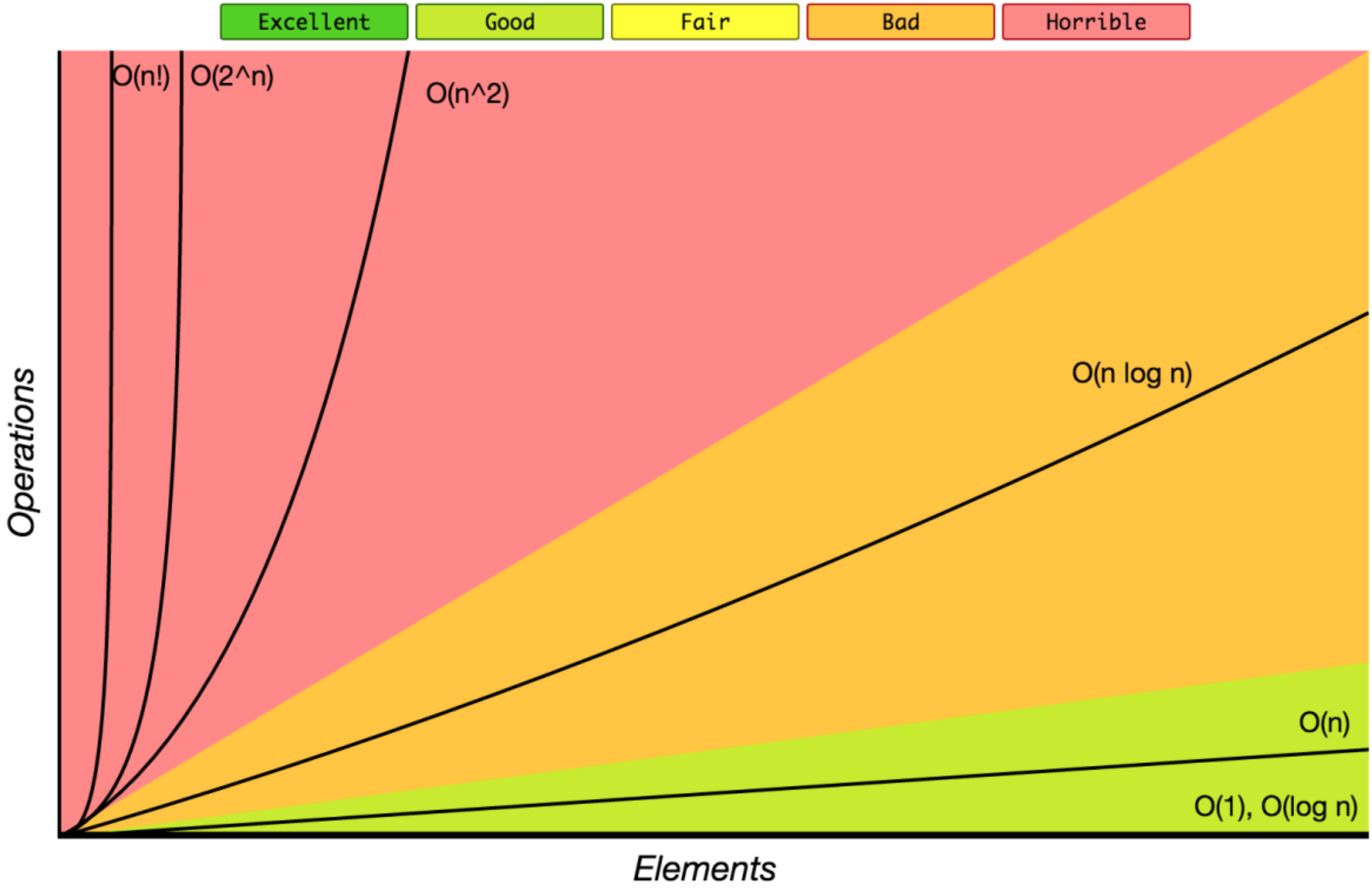


Complexidades mais Comuns

- $O(2^n)$



Big-O Complexity Chart



Growth Rate	Name	Code e.g.	description
1	Constant	a+=1;	statement (one line of code)
log(n)	Logarithmic	`while(n>1){ n=n/2; }`	Divide in half (binary search)
n	Linear	`for(c=0; c<n; c++){ a+=1; }`	Loop
n*log(n)	Linearithmic	Mergesort, Quicksort, ...	Effective sorting algorithms
n^2	Quadratic	`for(c=0; c<n; c++){ for(i=0; i<n; i++){ a+=1; } }`	Double loop
n^3	Cubic	`for(c=0; c<n; c++){ for(i=0; i<n; i++){ for(x=0; x<n; x++){ a+=1; } } }`	Triple loop
2^n	Exponential	Trying to braeak a password generating all possible combinations	Exhaustive search

Complexidades mais Comuns

TABLE 1.1 Running Time Comparisons

Complexity	16 Items	32 Items	64 Items	128 Items
$O(\log_2 n)$	4 seconds	5 seconds	6 seconds	7 seconds
$O(n)$	16 seconds	32 seconds	64 seconds	128 seconds
$O(n\log_2 n)$	64 seconds	160 seconds	384 seconds	896 seconds
$O(n^2)$	256 seconds	17 minutes	68 minutes	273 minutes
$O(n^3)$	68 minutes	546 minutes	73 hours	24 days
$O(2^n)$	18 hours	136 years	500,000 millennia	----------*

* My calculator doesn't go this high.

Algumas referências

- http://bigocheatsheet.com/
- http://adrianmejia.com/blog/2014/02/13/algorithms-for-dummies-part-1-sorting/
- Videos robozinho: https://youtube.com/playlist?list=PL2aHrV9pFqNS79ZKnGLw-RG5gH01bcjRZ&si=tzN38ysXgOWQg2mc
- Videos dança húngara: https://youtube.com/playlist?list=PLOmdoKois7_FK-ySGwHBkltzB11snW7KQ&si=YuBDGEtoBMCkikdB
- Material de apoio do prof. Tulio Bender

Referências