Estrutura de dados e complexidade de algoritmos

vídeo: https://www.youtube.com/watch?v=KjJwx-AB4KI

Big O — Theta

É uma notação que usamos para descrever o quanto o algoritmo irá custar.

$$f(x) \in O(g(x))$$

Notação PADRÃO DE BIG O:

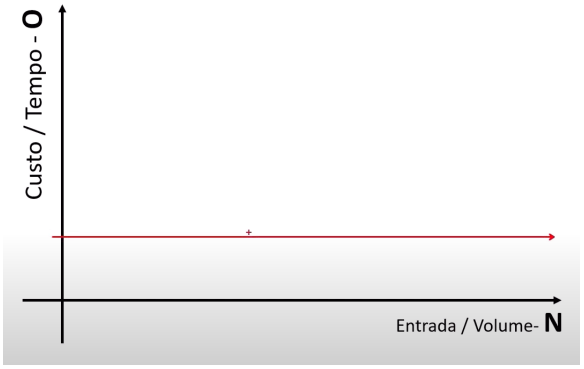
💡 **O(n)**

O = custo/tempo de processamento

n = entrada/volume de elementos

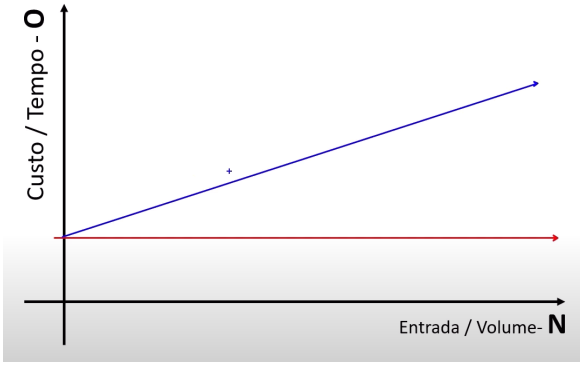
O gráfico é assim:

Se o processamento de algo é MUITO BOM ele performa assim:



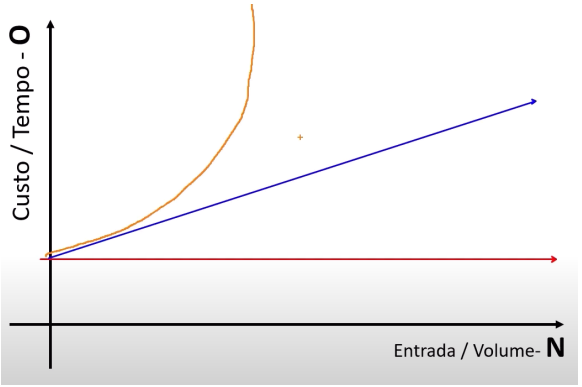
Ele fica com uma reta, igual a essa vermelha. Porque ele é constante. Processamento bom é constante. Não importa o número de entrada, o custo e o tempo de processamento não aumenta, continua estável.

Se o processamento é ruim, ele fica igual a reta azul:



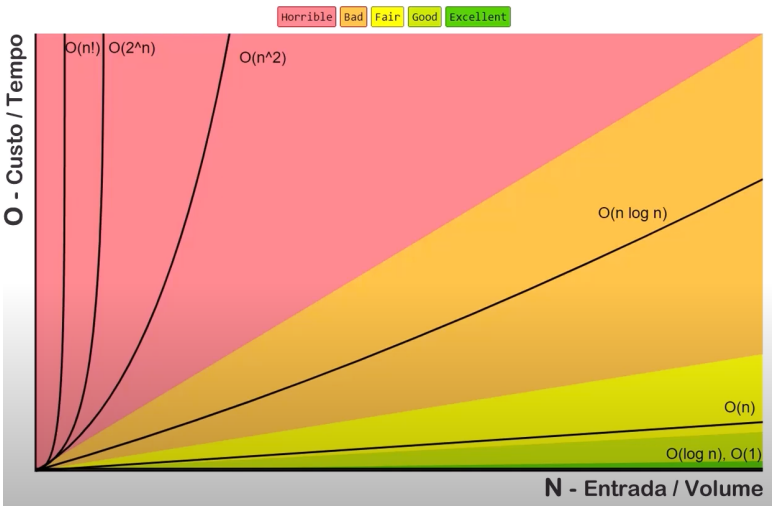
Porque o custo vai aumentando de acordo com a entrada.

Também é possível acontecer como o laranja:



Isso ocorre de forma exponencial (elevado a 2), por isso ele não é linear, ele sobe muitoooo. E isso é muito muito ruim. Quanto mais lá pra cima PIOR.

Gráfico de Complexidade do Big O:



A zona O(n) é a mais normal de todas, um simples for loop já chega a habitar essa zona. É bem razoável.

A partir da laranja já consideramos uma zona ruim de se estar.

A partir do rosa nosso código é muito sensível e não aguenta muito escalar.

Notações do Big O

O(1) - é uma ***constante***. Não há crescimento do número de operações, pois não depende do volume de dados de entrada (n).

Exemplo: um método simples que realiza apenas o retorno de uma divisão.

O(log(n)) - é de ***logaritmo.*** O crescimento do número de operações é menor do que o do número de itens.

Exemplo: uma binary search.

O(n) - é ***linear***! O crescimento no número de operações é diretamente proporcional ao crescimento do número de itens. A ação se repete conforme o número de entradas n.

Exemplo: um for loop simples.

O(n log n) - é ***linearitmica ou quasilinear***. É resultado das operações (log n) executada n vezes.

Exemplo: um merge sort

O(n²) - é quadrático. Ocorre quando os itens de dados são processados aos pares. Muitas vezes com repetições dentro da outra. For loops dentro de for loops ou recursividade por exemplo.

```
for (int i = 0; i < example.length; i++) { //O(n)
    for (int j = 0; j < example.length; i++) { O(n)
        counter++;
    }
}
//Ou seja: O(n * n) = O(n²)
```

O(2^n) *(dois elevado a n)* - é ***exponencial***. A medida que n aumenta, o fator analisado (tempo ou espaço) aumenta exponencialmente. O que é péssimo!

Exemplo: algoritmo de torre de Hanoi.

O(n!) - é ***fatorial.*** Pior caso possível. O número de instruções executadas cresce muito rapidamente para um pequeno número de dados.

Exemplo: problema do caixeiro-viajante.

Big-O	Alternativa
O(1)	O(yeah)
O(log n)	O(nice)
O(n)	O(ok)
O(n log n)	O(uch)
O(n^2)	O(my)
O(2^n)	O(no)
O(n!)	O(mg!)