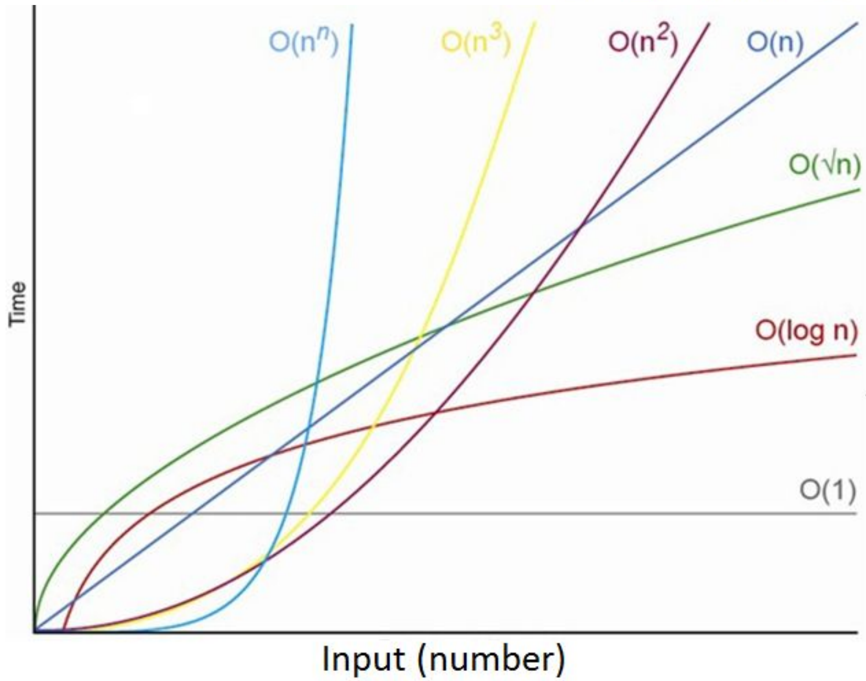
COMPLEXIDADE TEMPORAL E ESPACIAL

LAMFO

O QUE É COMPLEXIDADE ALGORÍTMICA?



Definição Resumida
É o ramo da ciência da computação que classifica a utilização de recursos na resolução de tarefas.

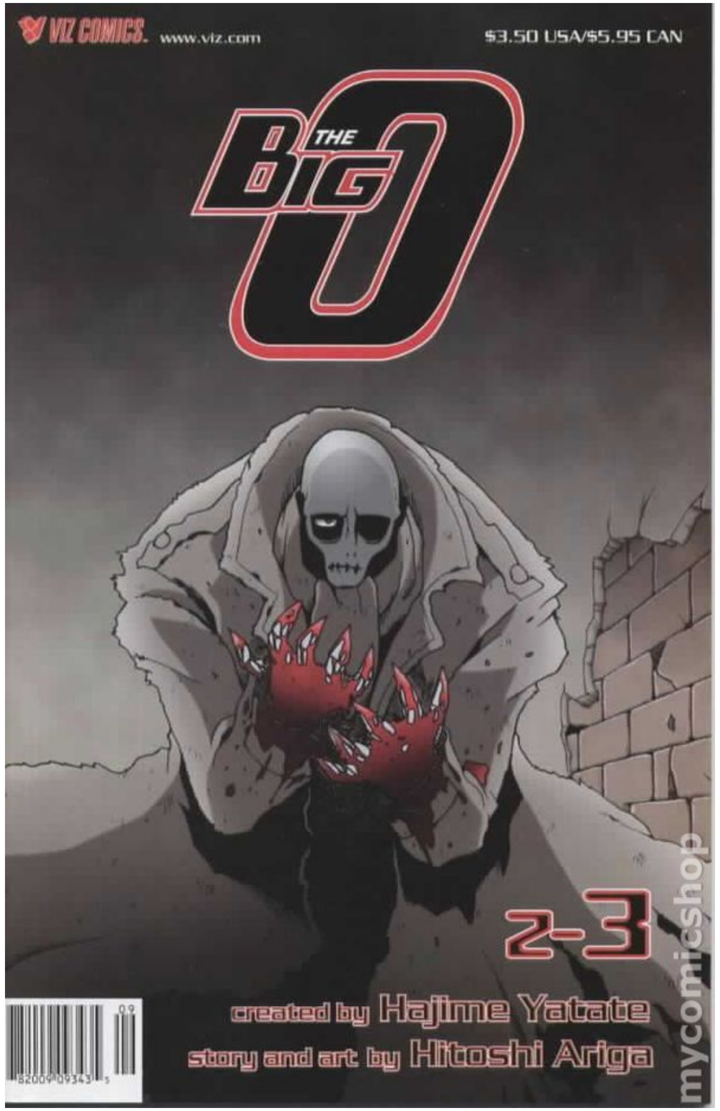
Definição Longa
A análise da Complexidade de algoritmos se preocupa em quão rápido um algoritmo qualquer executa. A definição de complexidade é dada como uma função T(n) sendo T o tempo da execução e **n** o tamanho do dado de entrada. Queremos definir o tempo gasto por um algoritmo sem a dependência de detalhes de implementação. Mas, é óbvio que T(n) depende da ***implementação!*** Um algoritmo vai levar diferentes tempos processando as mesmas entradas dependendo de fatores como Velocidade do processador, conjunto de instruções, velocidade do disco e compilador e etc.

POR QUE É IMPORTANTE ?

- Performance
- Tempo de Execução
- Dinheiro
- A análise de algoritmos é importante, em prática, porque a execução acidental de algoritmos ineficientes pode impactar significativamente a performance do sistema.

NOTAÇÃO BIG O

- Medida usada para definir a complexidade de um algoritmo.
- Representa o relacionamento entre input e os passos necessários para execução de dada tarefa.
- Usado para medir a complexidade independente do Hardware.



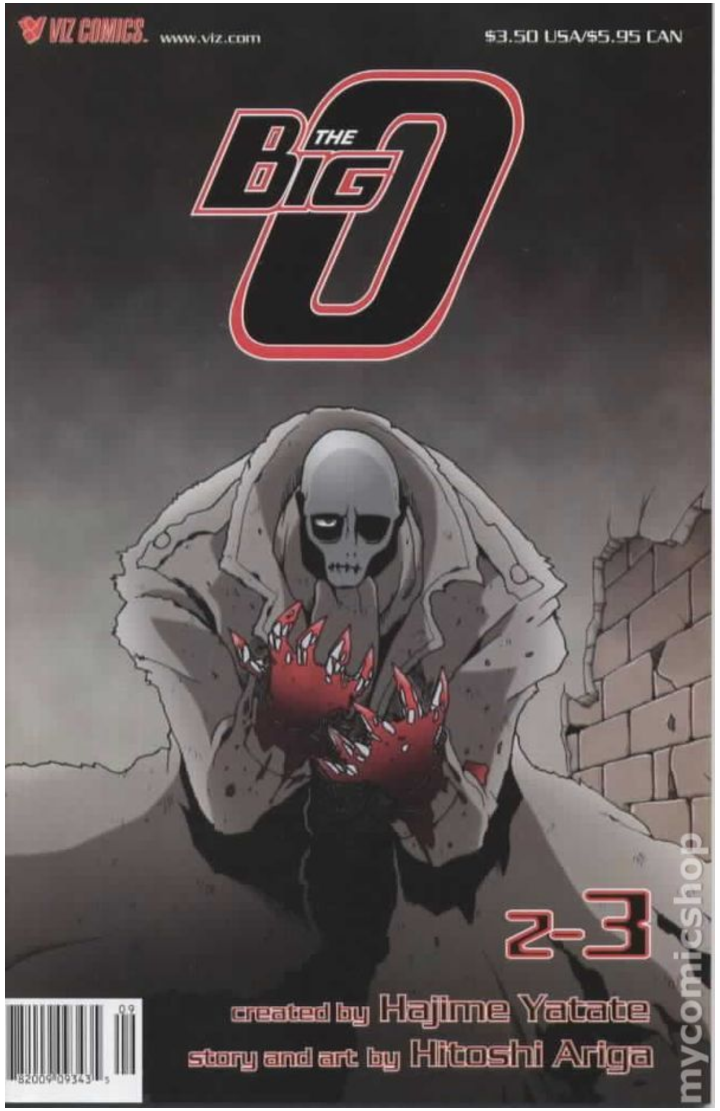
NOTAÇÃO BIG O

Definição formal:

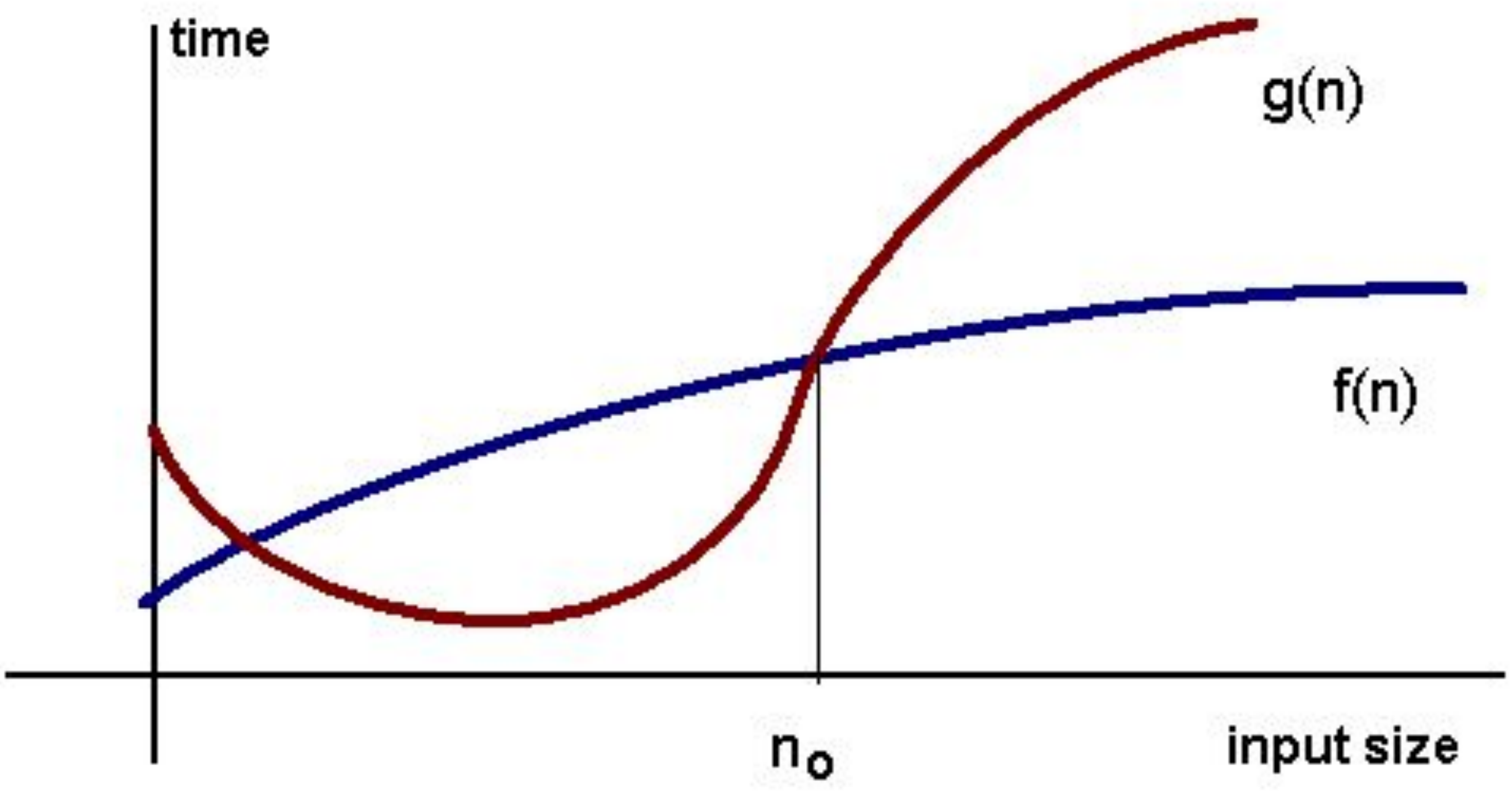
Sejam f e g duas funções definidas no mesmo subconjunto dos números reais pode-se dizer que $f(x) = O(g(x)) \text{ as } x \to \infty$

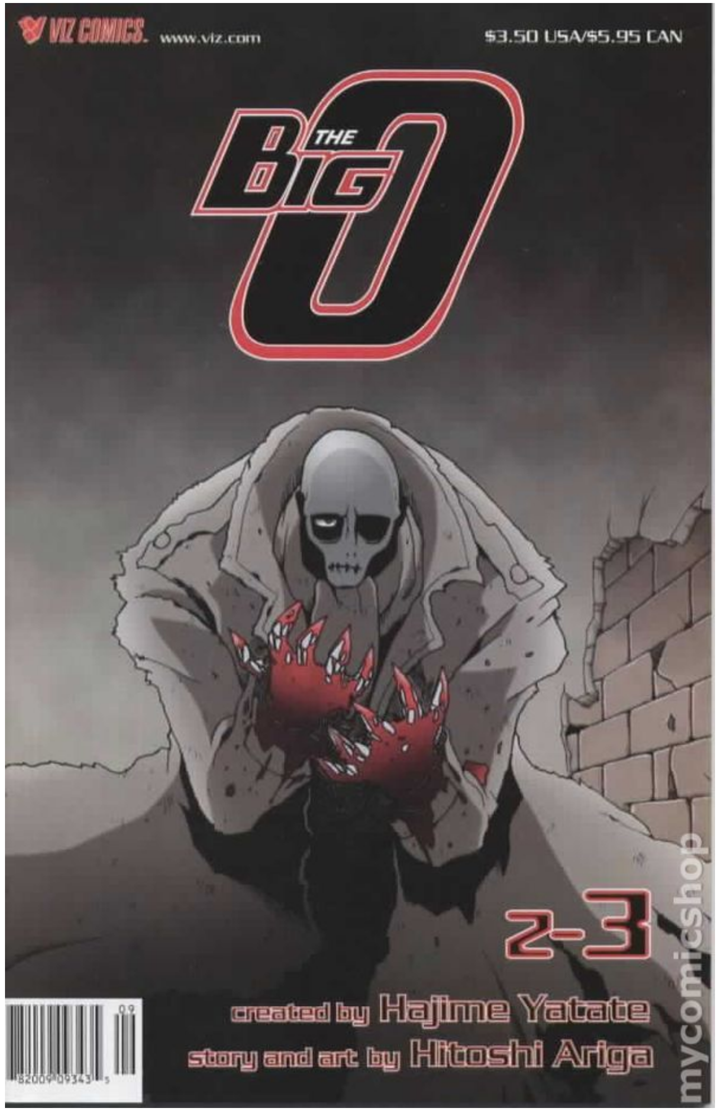
se e somente se existe uma constante positiva M tal que para todo valor suficientemente grande de x, o valor absoluto de f(x) é no máximo M multiplicado pelo valor absoluto de g(x). Isto é, f(x) = O(g(x)) se e somente se existe um número real positivo M e um número real x0 tal que $|f(x)| \leq M|g(x)| \text{ para todo } x \geq x_0.$

Em muitos contextos, a premissa que estamos interessados, a taxa de crescimento quando a variável x tende ao infinito, é deixada implícita, e é possível representá-la de forma mais simples em, f(x) = O(g(x)).



NOTAÇÃO BIG O

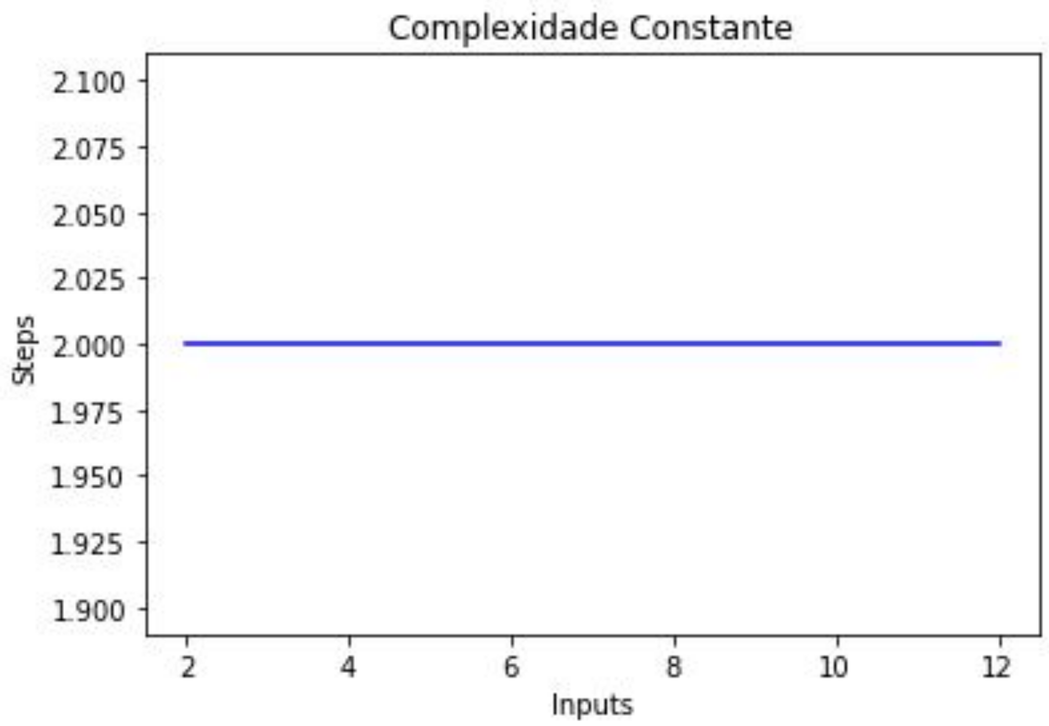




NOTAÇÃO BIG O

Complexidade Constante (O(C))
Exemplos:

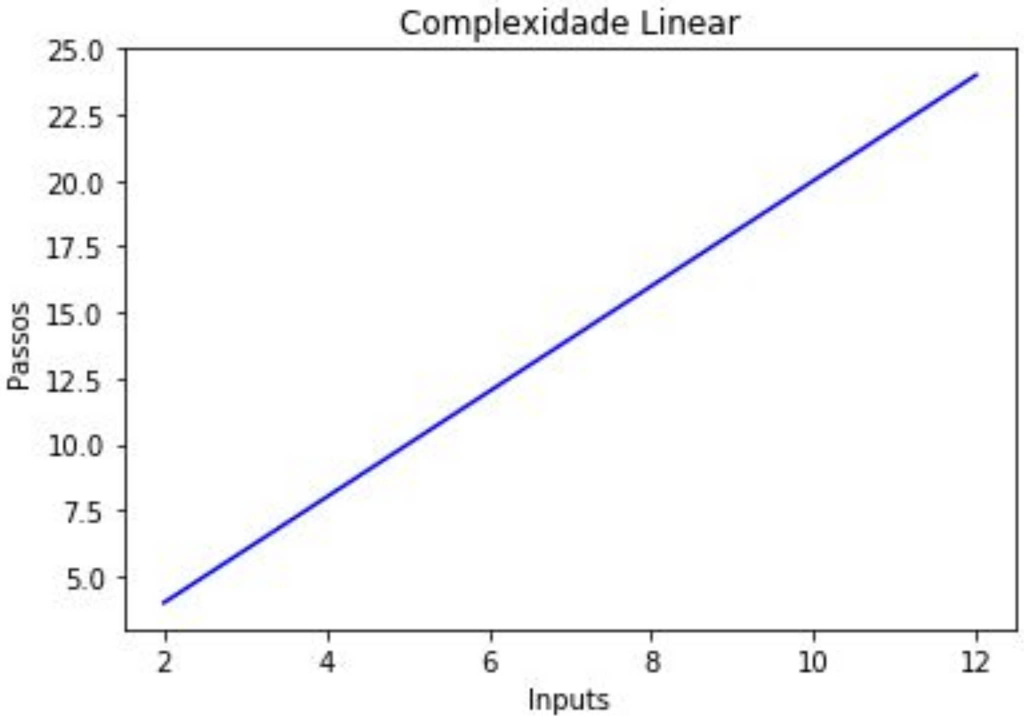
1. Acessando um array (int a = ARR[5];)
2. Inserindo um nó em uma lista
3. Inserindo e retirando nós de pilhas
4. Inserindo e retirando nós de listas



NOTAÇÃO BIG O

Complexidade Linear (O(n))
Exemplos:

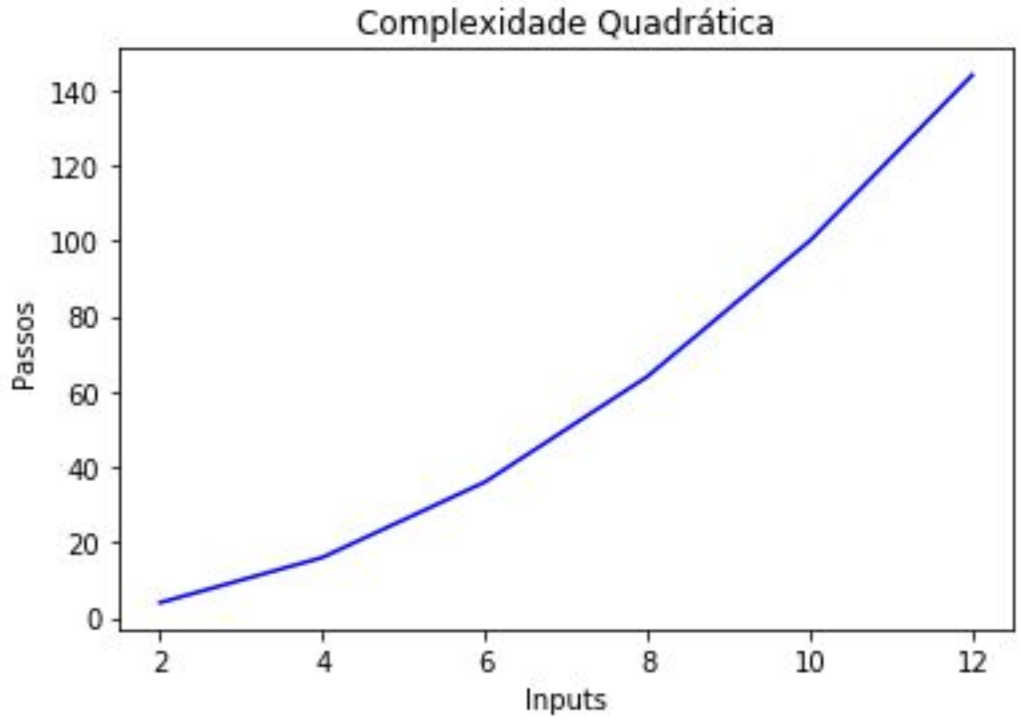
1. Pesquisa linear
2. Deleção de elemento específico em uma lista encadeada (Não ordenada)
3. Comparação de duas strings



NOTAÇÃO BIG O

Complexidade Quadrática ($O(n^2)$)
Exemplos:

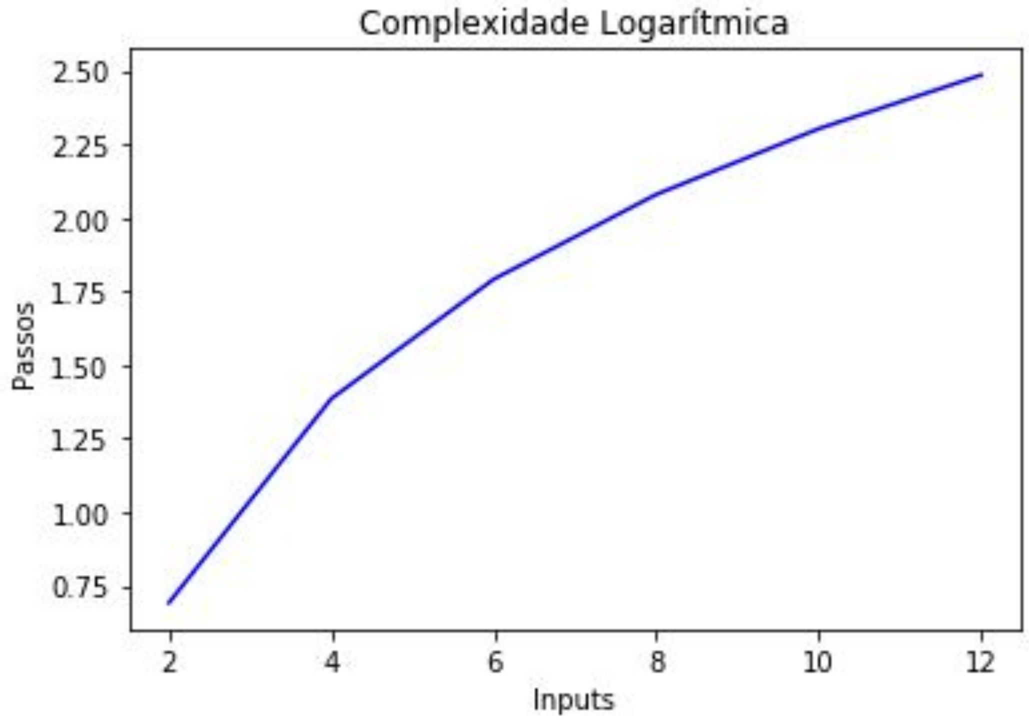
1. Insertion sort
2. Selection sort
3. Bubble sort
4. Quicksort



NOTAÇÃO BIG O

Complexidade Logarítmica(O(log(n)))
Exemplos:

1. Busca Binária
2. Encontrando maior/menor em uma árvore de busca binária
3. Alguns algoritmos de dividir e conquista baseados em funcionalidade linear
4. Calculando Fibonacci



CALCULANDO A COMPLEXIDADE

Exemplos:

Constante

O tempo de execução não é alterado em relação a n.

```
1. statement;
```

Linear

O tempo de execução do loop é diretamente alterado em proporção a n. Quando n dobra o tempo de execução dobra.

```
1. for ( i = 0; i < N; i++ )
2.   statement;
```

CALCULANDO A COMPLEXIDADE

Exemplos:

Logarítmico

O tempo de execução do algoritmo é proporcional ao número de vezes que n pode ser dividido por 2. Isto é devido ao algoritmo dividir a área de trabalho ao meio em cada iteração.

```
while ( low <= high ) {
  mid = ( low + high ) / 2;

  if ( target < list[mid] )
    high = mid - 1;
  else if ( target > list[mid] )
    low = mid + 1;
  else break;
}
```

CALCULANDO A COMPLEXIDADE

Exemplos:
Quadrático
O tempo de execução de dois loops é proporcional ao quadrado de N. Quando N dobra, o tempo de execução aumenta em N * N.

```
for ( i = 0; i < N; i++ ) {
  for ( j = 0; j < N; j++ )
    statement;
}
```

CALCULANDO A COMPLEXIDADE

Exemplos:

Quadrático

O tempo de execução de dois loops é proporcional ao quadrado de N. Quando N dobra, o tempo de execução aumenta em N * N.

```
for ( i = 0; i < N; i++ ) {
  for ( j = 0; j < N; j++ )
    statement;
}
```

COMPLEXIDADE TEMPORAL/ESPACIAL

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)

MUITO OBRIGADO!

BIBLIOGRAFIA

1. https://stackabuse.com/big-o-notation-and-algorithm-analysis-with-python-examples/
2. https://people.duke.edu/~ccc14/sta-663/AlgorithmicComplexity.html#space-complexity
3. https://www.cs.cmu.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html
4. https://en.wikipedia.org/wiki/Analysis_of_algorithms#Relevance
5. https://pt.wikipedia.org/wiki/Grande-O#Defini%C3%A7%C3%A3o_Formal
6. https://lamfo-unb.github.io/2019/04/21/Sorting-algorithms/
7. https://www.daniweb.com/programming/computer-science/threads/13488/time-complexity-of-algorithm#
8. https://www.quora.com/How-do-we-calculate-space-complexity