

Detonando no Code Challenge

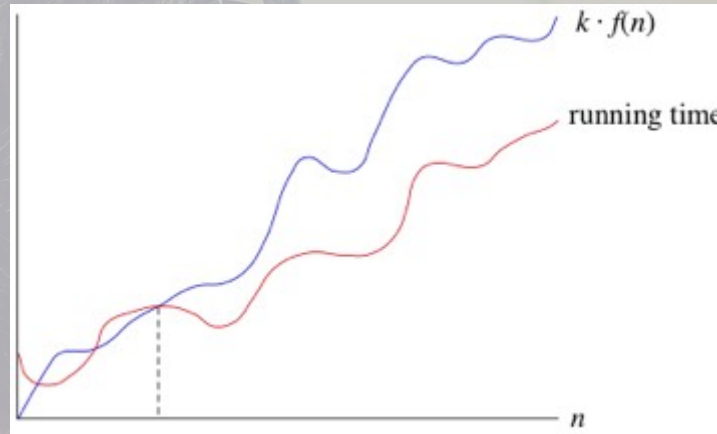
Com Python e Java

Cleuton Sampaio



Complexidade de tempo

Geralmente utiliza-se a notação Big O para determinar o limite máximo do tempo de execução esperado de um algoritmo, dado um conjunto de dados de entrada cujo tamanho é “n”.




Exemplos:

- **$O(n)$** – O tempo é linear e proporcional à quantidade de dados de entrada;
- **$O(n^2)$** – Quadrático! O tempo é o quadrado da quantidade de dados de entrada (simplificação);



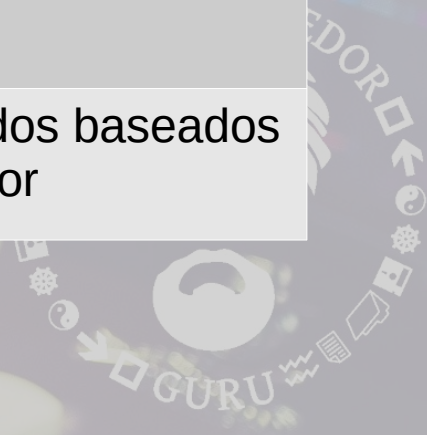
Complexidade de tempo

Menor tempo



$O(1)$	Constante	Acessar uma variável
$O(\log n)$	Logarítmico	Busca em árvores binárias
$O(n)$	Linear	Busca em um vetor
$O(n \log n)$	Linearitmico	Quicksort
$O(n^2)$	Quadrático	Loops aninhados baseados no mesmo vetor

Maior tempo



Exemplo anterior

```
class Solution:
    def twoSum(self, nums, target):
        for i in range(len(nums)):
            complemento = target - nums[i]
            for j in range(len(nums)):
                if complemento == nums[j]:
                    if complemento !=
nums[i]:
                        return [j, i]
            return None
```

Ordem
 $O(n^2)$

```
class Solution1 {
    public int[] twoSum(int[] nums, int target) {
        for (int i=0; i<nums.length; i++) {
            int complemento = target - nums[i];
            for (int j=0; j<nums.length; j++) {
                if (complemento == nums[j]) {
                    if (complemento != nums[i])
                        return new int [] {i,j};
                }
            }
        }
        return null;
    }
}
```


Melhor solução - python

```
class Solution:
    def twoSum(self, nums, target):
        ▶ dictValues = dict(zip(nums,[x for x in range(len(nums))]))
        ▶ for ix,n in enumerate(nums):
            complemento = target - n
        ▶ if complemento in dictValues:
            if dictValues[complemento] != ix:
                return [ix,dictValues[complemento]]
        return None
```

$O(n)$ →

$O(1)$ →

$O(2n)$ é linear, ou seja: $O(n)$ e tempo constante podemos desprezar.



Melhor solução - java

```
class Solution {  
    public int[] twoSum(int[] nums, int target) {  
        Map<Integer,Integer> dictValues = new HashMap<Integer,Integer>();  
        O(n) → for (int x=0; x<nums.length; x++) {  
                dictValues.put(nums[x], x);  
            }  
        for (int i=0; i<nums.length; i++) {  
            int complemento = target - nums[i];  
            O(1) → if (dictValues.containsKey(complemento)  
                    && dictValues.get(complemento) != i) {  
                return new int [] {i,dictValues.get(complemento)};  
            }  
        }  
        return null;  
    }  
}
```

$O(2n)$ é linear, ou seja: $O(n)$ e tempo constante podemos desprezar.