**Notação Big O**

Se refere a uma classificação de algoritmos de acordo com o tempo de execução, à medida em que aumenta a quantidade de dados a serem manupulados e a quantidade de memória exigida.

Exemplos de notação big O:

O(1): representa um algoritmo que é executado em tempo constante.

O(n): algoritmo que é executado em tempo linear, as execuções aumentam de acordo com as entradas – como a busca linear.

O(n log(n)): algoritmo que reduz pela metade uma lista a cada vez que é executado – *mergeSort* e *quickSort*

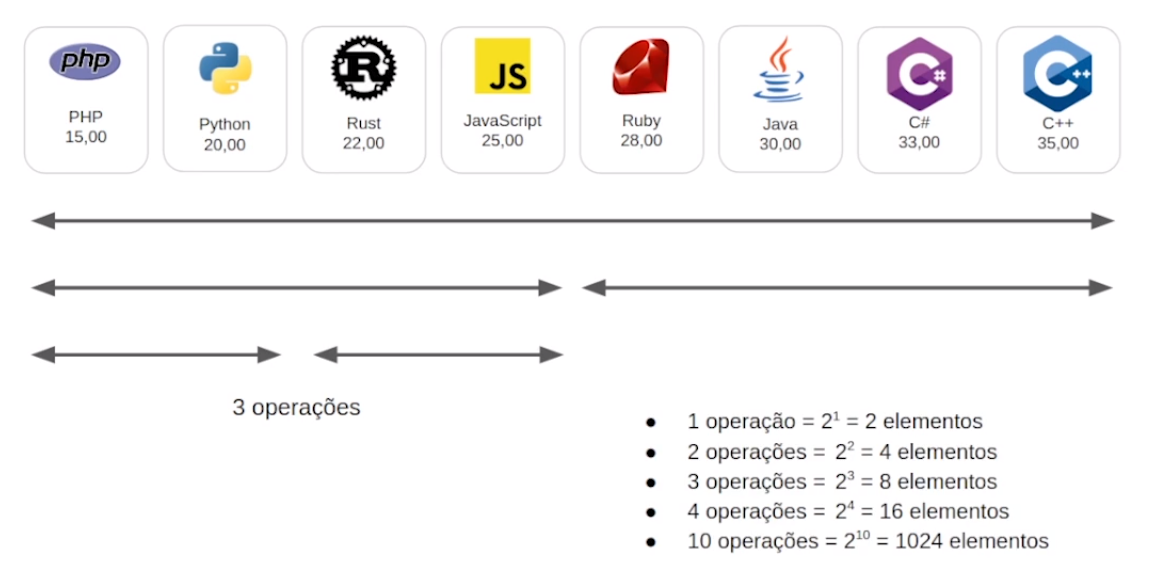
O(n²): algoritmo com tempo quadrático, significa que assim que o número de elementos na entrada aumenta, as execuções aumentam quadraticamente – *selectionSort* e *insertionSort*  **evitar**

**\*\*** *em computação sempre é utilizado o logaritmo com base 2(e, o inverso é a potência de 2). Assim, log n representa de forma abreviada log2n, ou seja, log de n na base 2. Esta característica está ligada ao sistema numérico utilizado pelos computadores: o sistema binário. Assim, sempre que trabalharmos com valores O(log n), estamos nos referindo a base 2.*

**Busca linear**

É um algoritmo de crescimento linear, pois passa em todos os elementos, e cresce em complexidade em relação direta com a quantidade de elementos. Como o crescimento é linear, a complexidade é expressa por **O(n)**

**Busca binária**



O número de operações cresce de acordo com o log desses elementos. Sendo assim a complexidade desse algoritmo é representado por **O(log n).** Lembrando que para tanto a busca linear quanto a busca binária funcionar, é necessário que a lista esteja ordenada

**MergSort**

Analisando o código da função de mergeSort observamos que o código percorre toda a lista no while, ou seja, percorre linearmente, e a cada vez que chamamos a função mergeSort dividimos o array em 2 e percorremos, e essa pequena parte cresce de forma logarítimica. Por isso chegamos à conclusão de que é uma função de complexidade: **n\*log n**

**QuickSort**

Assim como o mergSort o quickSort é uma função de complexidade: **n\* log n**.

Como os dois algoritmos tem a mesma complexidade O pode ser usado outras métricas para medir a performance de ambos, na média, observou-se que o quickSort é um pouco mais performático.

Esses dois algoritmos utilizam a abordagem de “dividir para conquistar” o que é uma estratégia melhor do que tentar resolver um problema grande de cara, os algoritmos que usam essa abordagem direta tendem a ser menos performáticos, mais lentos

***Curiosidade ->*** No javaScript, a forma como o método é implementado depende do *motor* que faz a interpretação. Os *vendor* (navegadores/interpretadores) decidem e fazem a implementação das funcionalidades. O chrome/NodeJS usa o motor V8, e o sort() usa o quick sort e o insertion sort. Já o Firefox usa o motor SpiderMonkey e o algoritmo usado é o mergeSort em C++

\*\*Algumas linguagens de programação tem o seu próprio método de ordenação/busca, e é interessante sabermos qual algoritmo essas linguagens utilizam por trás dos seus métodos