BIG O NOTATION

Big O notation nos permite dar una nomenclatura o simbología a la complejidad de los algoritmos. Pongamos como ejemplo un algoritmo de búsqueda, del cual queremos saber su velocidad. Es necesario tener en cuenta que no podemos hablar de tiempo, ya que eso depende del hardware en el que se ejecute: en una MacBook, el mismo algoritmo funcionará más rápido que en un smartphone de gama baja.

O(1) — notación constante

Esta expresión indica tiempo constante, lo que significa que el algoritmo se ejecutará con el mismo rendimiento sin importar el tamaño del input. Esto no quiere decir que sólo tendrá un input, más bien no se verá afectado por la cantidad de datos que estemos manejando.

O(log n) — notación logarítmica

Búsqueda en un array que se encuentra ordenado. Búsqueda binaria

O(n) — notación lineal

Esta es la expresión de crecimiento lineal, la complejidad del algoritmo aumenta de manera proporcional al tamaño del input.

O(n2) — notación cuadrática

Indica que el crecimiento en complejidad es proporcional al cuadrado del tamaño del input. Estos algoritmos suelen ser los menos eficientes, no se recomiendan para datos grandes y normalmente se dan cuando usamos ciclos for o iteraciones anidadas; es decir, si dentro de cada iteración de un arreglo lo vuelves a iterar, tendrás un algoritmo de complejidad cuadrada. Estos pueden llegar a complejidades cúbicas o factoriales.

O(n!) -- Factorial.

Por ejemplo, algoritmos de fuerza bruta que prueban todas las permutaciones.

Complejidad para las colecciones de Python:

https://wiki.python.org/moin/TimeComplexity