Reflexión | A01571214 Lautaro Gabriel Coteja

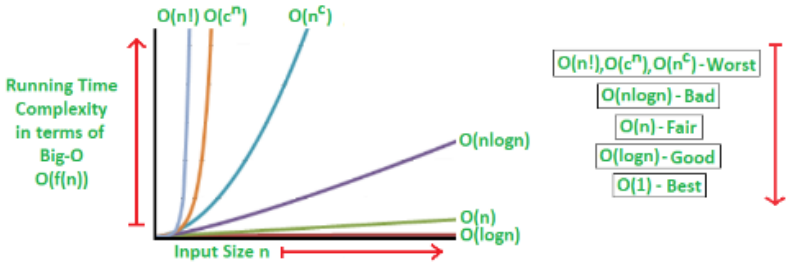
Teniendo en cuenta lo que nos plantea el problema que sería desarrollar una aplicación la cual lea un archivo y sea capaz de ordenar los elementos de tal archivo con el algoritmo de nuestra preferencia para después darle un límite inicial y final para que este funcione a través de busqueda binaria y luego nos de un intervalo de elementos entre tales límites y que nos lo imprima en la consola, debemos considerar que algoritmo usaremos ya que debemos tener en mente que los datos que nos dan son 16808, por lo que un algoritmo que sea lento haría que nuestra aplicación tarda demasiado y sea menos eficaz, mientras que si escogemos un algoritmo rápido y eficaz, nuestro programa será muy rápido, los algoritmos de búsqueda también tienen su importancia aunque en este caso usamos binaria. Hablando de características que poseen ciertos algoritmos que los hacen más o menos eficaces, están los espacios y estabilidad.

Tabla: Complejidad de los algoritmos de ordenamiento.

Algoritmo	Mejor	Promedio	Peor	Estable	Espacio
Swap sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Sí	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	$O(1)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	Sí	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	Sí	$O(1)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Sí	$O(n)$
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	Sí	$O(\log n)$

Ej. Complejidades de algunos algoritmos

Discutiendo de la importancia y eficacia de los algoritmos, de forma general, se considera una solución a un problema eficiente y sencilla, si el tiempo que tarda en ejecutarse la aplicación es de una función polinomial, osea se O(n^k), en caso de ser algo diferente a lo anterior mencionado la solución se puede considerar como ineficiente o compleja.



Ej. Tiempo de ejecución contra tamaño de input

En la imagen de arriba podemos ver una comparativa de cómo dependiendo de nuestra función O, el tiempo que se tarda en ejecutar el algoritmo es rápido o lento, siendo como podemos, O(1) el mejor y más rápido de todos.

En nuestra aplicación, nosotros anotamos todos nuestras complejidades computacionales de todos nuestros métodos pero discutiremos solo los de los algoritmos que usamos, para ordenar los datos usamos el merge sort ya que este tiene una complejidad de O(nlogn) la cual no es la mejor pero es mejor que el método que usamos anteriormente que era el bubble sort, lo cambiamos ya que el bubble sort tiene una complejidad de O(n^2) la cual es peor como pudimos ver en la imagen que O(nlogn). También hay que tener en cuenta que escogimos merge ya que mientras que con una cantidad pequeña de datos no importa mucho, hablamos de que nuestros datos son 16808, por lo cual merge resulta mejor para tal cantidad. Finalmente para nuestra búsqueda entre rango utilizamos la binaria, ya que de igual manera que en el anterior caso, son 16808 datos y Búsqueda binaria tiene el mejor peor caso con un O(logn) el cual es el segundo mejor en respuesta.

Búsqueda binaria	$O(1)$	$O(\log_2 n)$	$O(\log n)$