Documento de análisis- Reto 1

Estudiante 1: Nicholas Barake//202020664

Estudiante 2: Jesed Domínguez//202011992

*NOTA: “ms” es milisegundos

Para este reto, utilizamos el algoritmo de ordenamiento Mergesort, por lo que consideramos que sería el que nos brindará mayor rapidez en tiempos de ejecución debido a las características que proporciona. Además de esto, nuestra estructura de datos base fue el Array_list ya que analizando cada requerimiento, logramos darnos cuenta que el común denominador de ellos era buscar dentro de las listas las respuestas solicitadas. Entonces, por su complejidad O(1) buscando elementos, decidimos tomarlo como base.

Requerimiento 1

Complejidad: O(1), esto es debido a que utilicé API TAD ARRAY LIST con complejidad O(1). Lo que quiero decir, es que las funciones que usé sólo tienen esa complejidad, entonces la complejidad general del requerimiento será esa.

Pruebas de tiempo de ejecución:

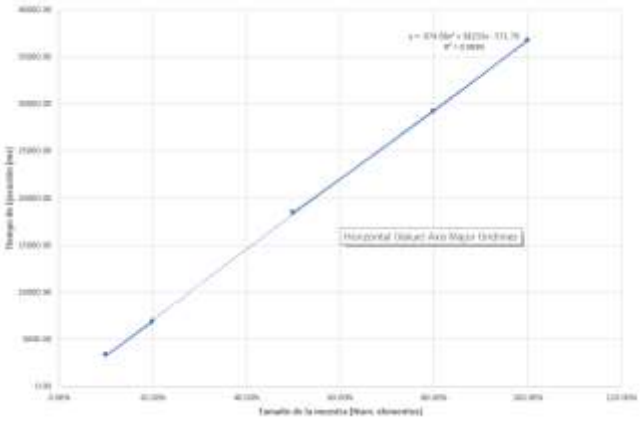
- 10pct: 2ms
- 20pct: 2ms
- 50pct: 2ms
- 80pct: 2ms
- Large: 2ms

Requerimiento 2

Complejidad: O(2n), n siendo el número de elementos en la lista de valores correspondientes a “Artwork” en el catálogo principal.

Pruebas de tiempo de ejecución:

- 10pct: 3312.5 ms
- 20pct: 6890.625 ms
- 50pct: 18468.75 ms
- 80pct: 29234.375 ms
- Large: 36750.0 ms



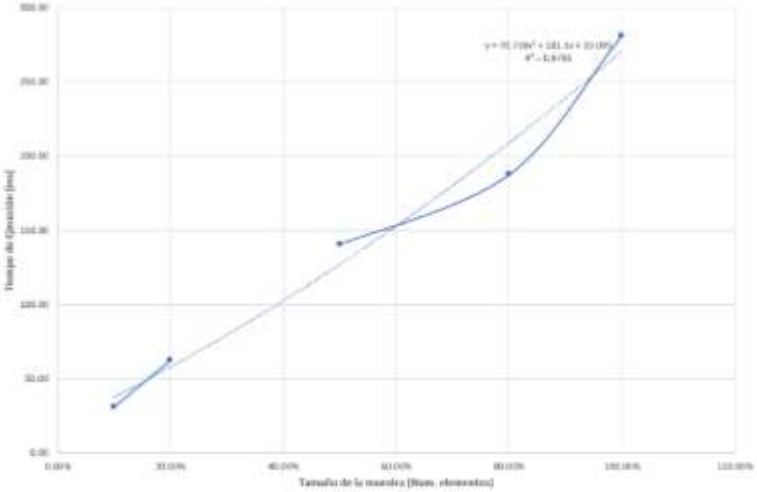
Por lo que se puede evidenciar en la gráfica, el código que se implementó, a medida que se van aumentando los datos, los tiempos de ejecución crecen linealmente. Así como se estipuló en la complejidad. Es como si la pendiente de la recta en la gráfica fuese el coeficiente que acompaña a “n” en O(2n), en este caso 2.

Requerimiento 3 (realizado por Nicholas Barake)

Complejidad: O(4n), n siendo el número de elementos en la lista de valores correspondientes a “Artist” en el catálogo principal.

Pruebas de tiempo de ejecución:

- 10pct: 31.25 ms
- 20pct: 62.5 ms
- 50pct: 140.625 ms
- 80pct: 187.5 ms
- Large: 281.25 ms



Por lo que se puede evidenciar en la gráfica, el código que se implementó, a medida que se van aumentando los datos, los tiempos de ejecución crecen linealmente. Así como se estipuló en la complejidad. Es como si la pendiente de la recta en la gráfica fuese el coeficiente que acompaña a “n” en O(4n), en este caso 4.

Requerimiento 4 (realizado por Jessed Domínguez)

Complejidad: O(n) Este requerimiento presenta una complejidad algoritmica O(n) debido a la cantidad de ciclos que en él hay. A pesar estos ciclos deben ir comparando 1 datos con otros miles de datos para encontrar su pareja ID. Haciendo que sea ineficiente a pesar de usarse un array_list para buscar esos datos.