• Nombres, código y correo Uniandes de los integrantes del grupo:

Juan Sebastián Rueda 202226482

Diego Alejandro Rodríguez 202225217

Juan Pablo Mendoza Arias 202221917

• Para los requerimientos individuales se debe indicar que estudiante del equipo lo realizó.

Req3: Juan Sebastián

Req4: Juan Pablo

Req5: Diego Alejandro

• Análisis de complejidad de cada uno de los requerimientos en Notación O.

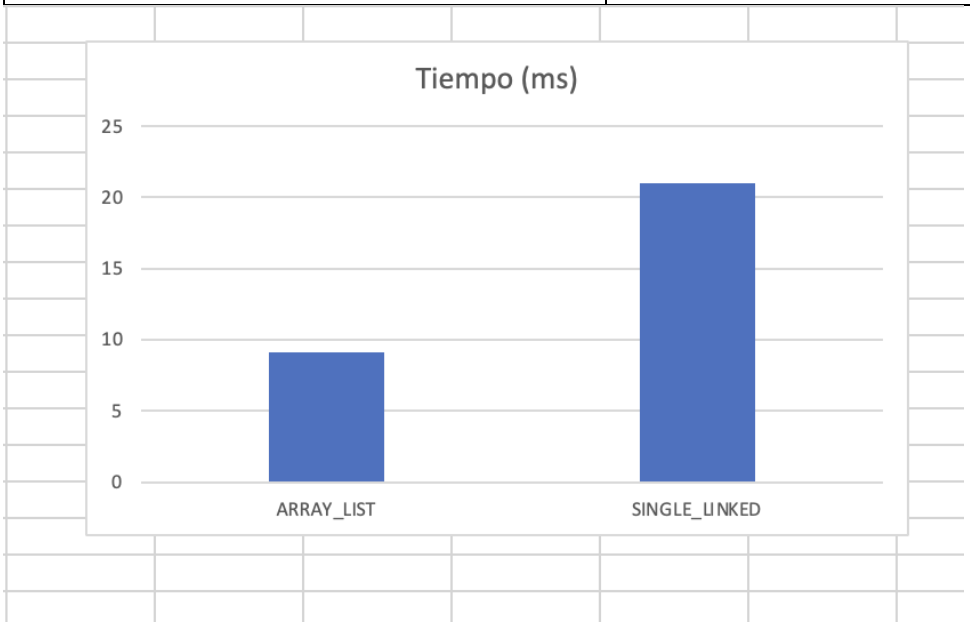
• Pruebas de tiempos de ejecución para cada uno de los requerimientos. En estas pruebas se deben incluir: o Las tablas de tiempos de ejecución registrados. o Las gráficas comparativas de los experimentos. o Un análisis de resultados comparándolo los resultados obtenidos con el análisis de complejidad realizado.

Carga de datos: Los datos fueron cargados en ARRAY_LIST puesto que en todos los requerimientos es necesario extraer los datos muchas veces, y para ello se utiliza getElement. Arraylist funciona mejor (O(1)), que Single linked para esta tarea (O(n)).

Requerimiento 1:

Complejidad: O(n)

Datastructure	Tiempo (ms)
ARRAY_LIST	9,14
SINGLE_LINKED	20,98



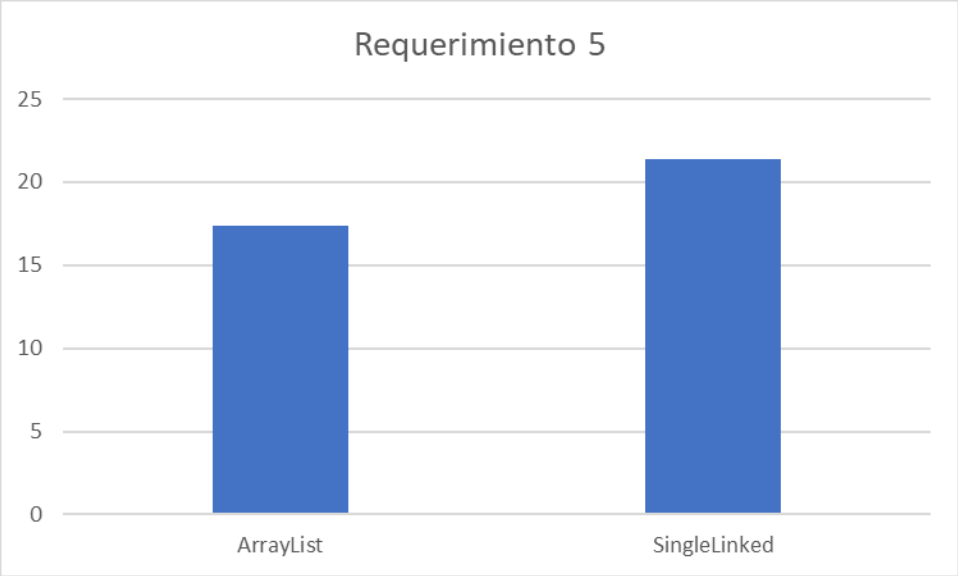
La diferencia de tiempos se debe

Requerimiento 2:

La complejidad temporal del Requerimiento 2 es O(N) ya que se recorren todos los datos que entran 1 vez

Estructura de datos	Tiempo (ms)
ARRAY_LIST	17,34
SINGLE_LINKED	21,35

Los tiempos de ejecución de ambos algoritmos fueron similares, sin embargo, arraylist funcionó ligeramente mejor en promedio.



Requerimiento 3:

Complejidad: O(N^2)

Estructura de datos para todos los datastructires creados	Tiempo (ms)
ARRAY_LIST	1669, 83
SINGLE_LINKED	1893,33

Como son muy pocos los datos que se ingresan a los datastructure, no muy importante el tipo de lista pues el rendimiento tiende a ser similar. Se elige, por homogeneidad, ARRAY_LIST.

Requerimiento 4:

Complejidad: O(N^2)

Puesto que se recorren todos los datos que vienen por parámetro dentro del DataStructure y luego se tienen que volver a recorrer una parte de estos, a lo sumo su complejidad será de O(N^2)

Estructura de datos	Tiempo (ms)
ARRAY_LIST	7957,812
SINGLE_LINKED	8756,34

En este caso, es mucho mejor usar el ArrayList, debido a que hay una diferencia de milisegundos importante. Sin embargo, como no se usan demasiados datos, es un poco insignificante la elección con pocas cantidades.

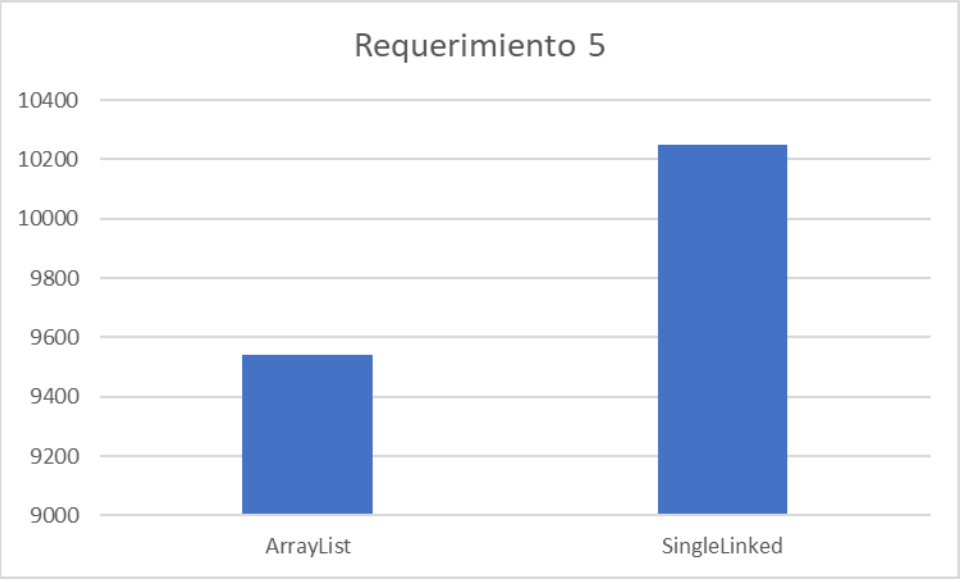
Requerimiento 5:

Complejidad: O(N^2)

La complejidad de este algoritmo en el peor de los casos es O(N^2) ya que se tienen que recorrer todos los datos y dentro de cada uno se itera cada elemento.

Estructura de datos	Tiempo (ms)
ARRAY_LIST	9542.23
SINGLE_LINKED	10250.82

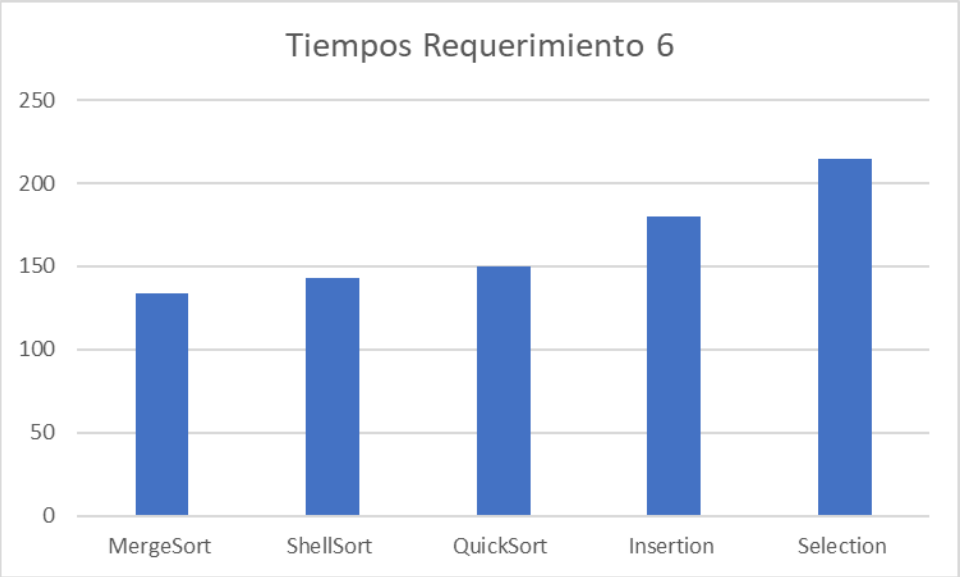
Los tiempos de ejecución del array list fueron inferiores a los del single linked, esto ya que en la función se usa de forma constante el getElement y el addLast, los cuales para el arrayList, tienen una complejidad de O(1) mientras que en la lista encadenada son de O(n)



Requerimiento 6:

El requerimiento 6 se tarda 134 ms en ejecutarse con la mayor cantidad de datos brindada (4903, los cuales técnicamente siguen siendo pocos), usando MergeSort como algoritmo de ordenamiento. ShellSort se comporta de manera similar, con un promedio de 143 ms. QuickSort también lo hace rápidamente, pero con un promedio de 150 ms. Selection lo hace con 215 ms e Insertion lo hace con 180 ms de media. Esto se respalda en las observaciones hechas previamente, en los laboratorios 4 y 5.

MergeSort	ShellSort	QuickSort	Insertion	Selection
134	143	150	180	215

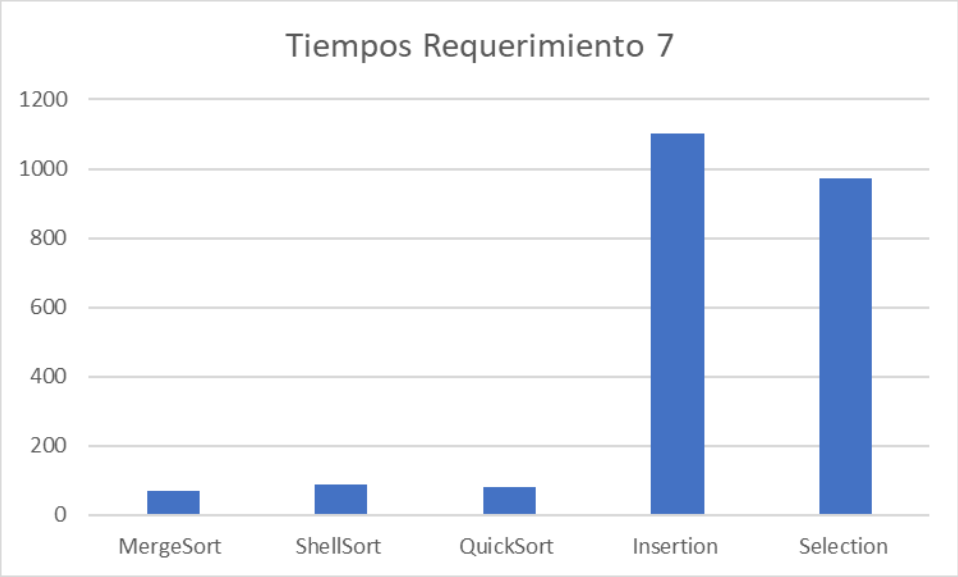


La complejidad del requerimiento 6 es O(N^2), puesto que se está recorriendo una parte de los elementos del DataStructure que llega por parámetro, diga N/9, ya que los años van desde 2012 hasta 2021 y si se supone que la información está distribuida por igual, se aproximaría. A su vez, dentro de este ciclo se vuelven a iterar una parte de estos datos, la que corresponde a la de cada subsector. Es por esto que la complejidad es aproximadamente O(N^2).

Requerimiento 7

El requerimiento 6 toma 71,3 ms en ejecutarse con el máximo de datos al utilizar MergeSort, muy cerca a estos tiempos se encuentran QuickSort y ShellSort con 80,3 y 88,1 ms respectivamente. Por otro lado, al utilizar Selection, los tiempos de ejecución se elevan a 973,98 ms y aumentan aun más al usar insertion sort hasta llegar a 1100,09 ms en promedio.

MergeSort	ShellSort	QuickSort	Insertion	Selection
71,3	88,1	80,3	1100,09	973,98

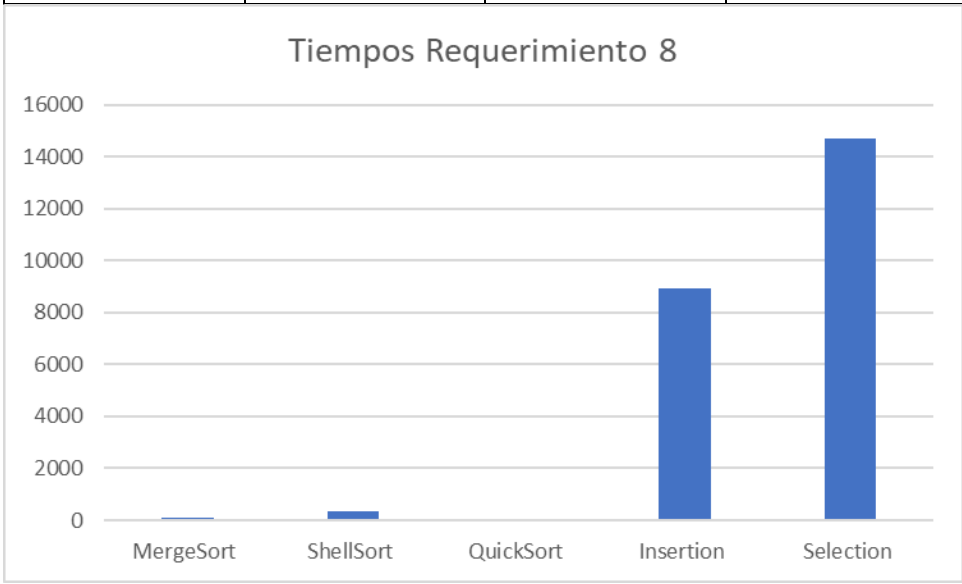


La complejidad de este requerimiento es O(N^2) esto ya que primero se recorre cada año donde se recorren cada uno de los elementos en el datastruct hasta que se completen la cantidad de N datos que se solicitan, por lo que puede repetirse N veces esto.

Requerimiento 8

Este requerimiento tardó en promedio 111 ms en ejecutarse usando MergeSort, 8950 ms con Insertion, 14701 ms con Selection, 150 con QuickSort y 327 con ShellSort

MergeSort	ShellSort	QuickSort	Insertion	Selection
111	327	150	8950	14701



La complejidad de este requerimiento es de N^2 puesto que se recorren los sectores y a su vez los subsectores dentro de cada sector, por lo que usando un argumento similar al del requerimiento 6, es O(N^2).