Isai Daniel Chacón Silva - 201912015

Nicolás Aparicio Claros - 201911357

Estructuras de datos y algoritmos

Reto 1

En este documento se realiza el análisis de complejidad para cada uno de los algoritmos implementados para resolver cada requerimiento. En primer lugar, es importante resaltar que la estructura de datos usada para todas las listas del proyecto fue la de Array List. Esta decisión se tomó basada en el laboratorio *LabSorts-S05-G01*, debido a que realizando estimaciones sobre los tiempos de ejecución de ordenamiento para diferentes tamaños, se encontró que al utilizar una estructura Single Linked, el tiempo para correr cada requerimiento aumentaba en ordenes de magnitud considerables.

La segunda decisión tomada que cabe destacar fue que se escogió el ordenamiento de MergeSort para todos los ordenamientos llevados a cabo. Esto debido a que las pruebas ejecutadas también durante *LabSorts-S05-G01* permitieron identificarlo, junto con Insertion Sort como uno de los más eficientes teniendo en cuenta el tiempo de ejecución. En este orden de ideas, y basados en que no se obtuvieron limitaciones de almacenamiento, decidimos utilizar a Merge Sort dado que durante el reto se utilizarían muchos más datos de la base de datos (no solo -small y 10% como en el laboratorio) y este algoritmo presenta un orden de complejidad $O(Nlog_N)$ independiente del tamaño de los datos a organizar. Así pues, inferimos que este algoritmo recursivo sería el que mejor se desempeñaría para las tareas indicadas.

Tabla 1. Especificaciones de la máquina para ejecutar las pruebas de rendimiento.

Características	**Máquina de Prueba 1**
Procesadores	Intel ® core TM i5-8250U CPU @1,60GHz
Memoria RAM /GB)	8 GB
Sistema Operativo	Microsoft Windows 10 Pro basado en x64

La segunda decisión tomada que cabe destacar fue que se escogió el ordenamiento de MergeSort para todos los ordenamientos llevados a cabo. Esto debido a que las pruebas ejecutadas también durante *LabSorts-S05-G01* permitieron identificarlo, junto con Insertion Sort como uno de los más eficientes teniendo en cuenta el tiempo de ejecución. En este orden de ideas, y basados en que no se obtuvieron limitaciones de almacenamiento, decidimos utilizar a Merge Sort dado que durante el reto se utilizarían muchos más datos de la base de datos (no solo -small y 10% como en el laboratorio) y este algoritmo presenta un orden de complejidad $O(Nlog_N)$ independiente del tamaño de los datos a organizar. Así pues,

inferimos que este algoritmo recursivo sería el que mejor se desempeñaría para las tareas indicadas.

Análisis Teórico

Sea n el número de datos en la lista de artistas

Sea m el número de datos en la lista de obras de arte

Requerimiento 1

Para este requerimiento se llaman principalmente 2 funciones que influyen sobre el orden del algoritmo. El primero de estos es *artistDates* que recorre la lista de artistas una vez, por lo que es O(n) y añade elementos al final de una lista, lo cual incurre en un orden O(1) al ser un Array List, lo cual se llega a realizar n veces en el peor de los casos. Luego de que se recorre esta lista, se ordena por medio de mergesort, para el cual se asume que el peor caso es que se hayan añadido todos los artistas a la sublista por organizar, de modo que el orden vendría dado por O(nlog(n)).

Posteriormente, se llama a la función *printResultsArtists*, la cual se encarga de imprimir los primeros 3 elementos de la lista y los últimos 3. Dado que se trabaja con Array List, al llamar a getElement, su orden será constante. Y toda la función tendrá un orden dado por O(6).

Así pues, la suma de todos estos órdenes será O (n+ nlog(n) + 6 + n), lo cual se aproxima a O(2n) por ser el polinomio más representativo.

Tabla 2. Resultado tiempo de ejecución para el requerimiento 1

Porcentaje de la muestra [pct]	**T**amaño de la muestra (ARRAYLIST)	**Merge Sort [ms]**	
5,00%	12568,00	40,00	**REQUERIMIENTO 1**
10,00%	21664,00	58,00	
20,00%	38213,00	83,00	
50,00%	83569,00	66,00	

Requerimiento 2

Este requerimiento es análogo al 1. La primera función que influye en el orden del algoritmo es *artworksDates* que recorre la lista de obras de arte una sola vez, por lo que es O(m) y añade elementos en la última posición de la lista, lo cual incurre en un orden O(1) al ser un Array List, proceso que se llega a repetir m veces en el peor de los casos. Luego de que se recorre esta lista, se ordena por medio de mergesort, para el cual se asume que el peor caso es que se hayan añadido todas las obras de arte a la sublista por organizar, de modo que el orden vendría dado por O(mlog(m)).

Posteriormente, se llama a la función *printResultsArtworks*, la cual se encarga de imprimir los primeros 3 elementos de la lista y los últimos 3. Dado que se trabaja con Array List, al llamar a getElement, su orden será constante. Y toda la función tendrá un orden dado por O(6).

De modo que la suma de todos estos órdenes será O (m+ mlog(m) + 6 + m), lo cual se aproxima a O(2m) por ser el polinomio más representativo.

Tabla 3. Resultado tiempo de ejecución para el requerimiento 2

Porcentaje de la muestra [pct]	**T**amaño de la muestra (ARRAYLIST)	**Merge Sort [ms]**	
5,00%	12568,00	48,00	**REQUERIMIENTO 2**
10,00%	21664,00	1045,00	
20,00%	38213,00	2023,00	
50,00%	83569,00	5089,00	

Requerimiento 3

Para el caso de seleccionar la técnica, primeramente, se llama a la función *artista_technique.* Esta función recorre tanto la lista de artistas como de obras de arte. Así pues, su orden es dado por O(m+n+m) ya que se añade también al final las artworks a una nueva lista m veces, lo cual sería O(1). Luego, para la función *most_used_technique* recorre toda la lista de técnicas utilizadas por un artista para pintas las obras, lo cual sería O(m) ya que busca, a su vez, la técnica más usada. La función final de print de este requerimiento es O(1) ya que recurre al diccionario conociendo la llave a utilizar. Así pues, en total se tendría un orden de O(3m+n), en donde normalmente las obras de arte son más que la cantidad de artistas para la base de datos dada, por lo que esto se sintetiza en O(3m).

Tabla 4. Resultado tiempo de ejecución para el requerimiento 3

Porcentaje de la muestra [pct]	**T**amaño de la muestra (ARRAYLIST)	**Merge Sort [ms]**	
5,00%	12568,00	25,00	**REQUERIMIENTO 3**
10,00%	21664,00	35,00	
20,00%	38213,00	82,00	
50,00%	83569,00	1013,00	

Requerimiento 4

Este requerimiento llama a 4 funciones. La primera de estas, *artworks_artistnationality* posee un orden de O(m^2+n^2) ya que realiza un doble recorrido sobre artistas y obras de arte. El print siguiente es O(k) ya que simplemente se encarga de imprimir la tabla para un input dado. Luego, InfoArtWorksNationality incurre en O(n + m^2). Finalmente, el último print, al tener un sample definido de 3, sería O(k), que influye mínimamente sobre el orden

total, ya que los términos más grandes serán quienes dominen sobre el orden del algoritmo, siendo este en total O(2m^2+n^2 + n), lo que se aproxima a O(2m^2+n^2) por ser los polinomios más representativos.

Tabla 5. Resultado tiempo de ejecución para el requerimiento 4

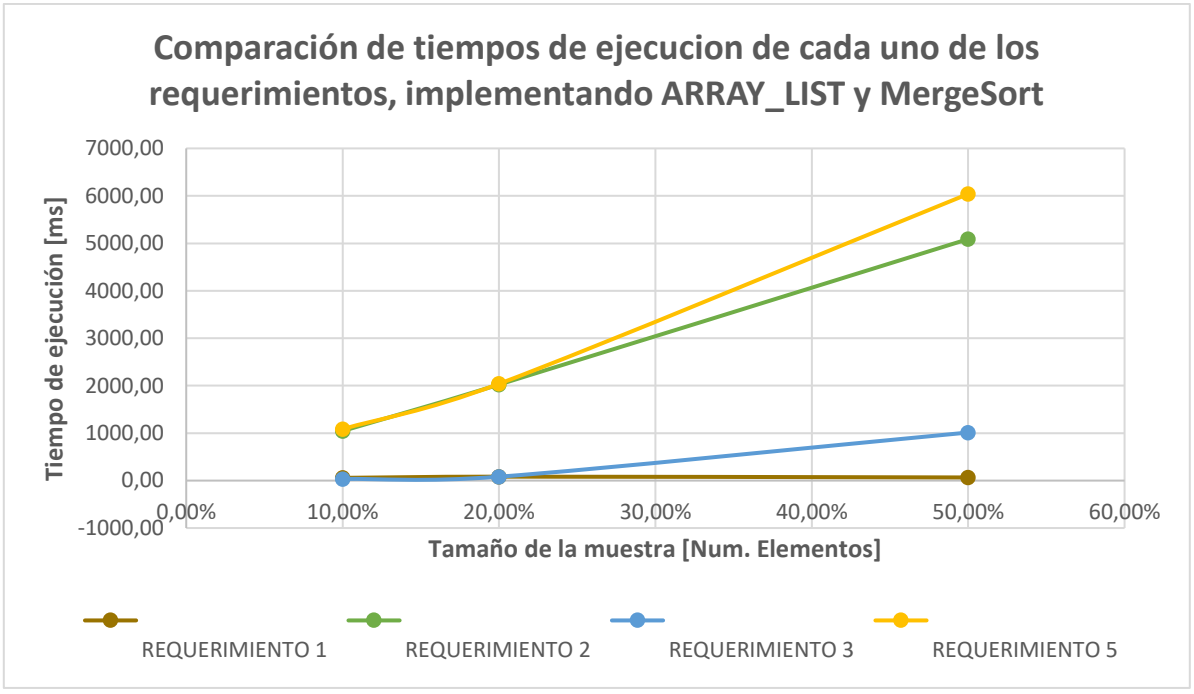
Porcentaje de la muestra [pct]	**T**amaño de la muestra (ARRAYLIST)	**Merge Sort [ms]**	
5,00%	12568,00	26093,00	**REQUERIMIENTO 4**
10,00%	21664,00	100025,00	
20,00%	38213,00	197087,00	
50,00%	83569,00	554750,00	

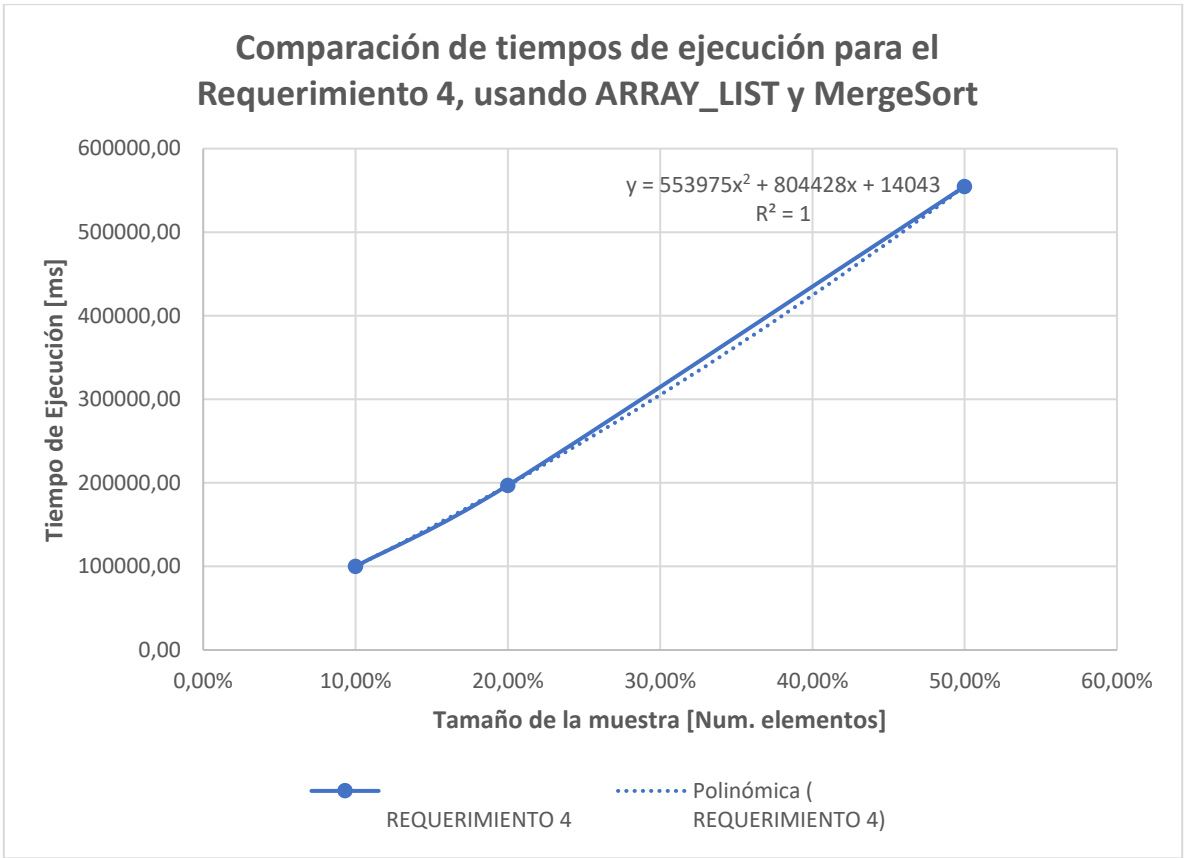
Requerimiento 5

En este caso se llama la función *artworks_department*, la cual es la que influye principalmente sobre el orden del algoritmo. Esta función realiza un recorrido sobre las obras de arte y realiza una serie de comparaciones para conocer el precio de las obras, teniendo en cuenta si esta está en m^2, m^3 o kg. Por tanto, su orden vendría dado por O(m). Posteriormente, se realizan dos ordenamientos sobre dichas obras de arte por medio de merge teniendo en cuenta tanto el precio, como la antigüedad. De modo que estos ordenamientos serían O(2mlog(m)). En total, por tanto, el orden sería de O(m+2mlog(m)). Los prints de esta función recurren en O(k) ya que se sabe a priori que se necesitan los primeros 5 y últimos 5 de las categorías a buscar.

Tabla 6. Resultado tiempo de ejecución para el requerimiento 5

Porcentaje de la muestra [pct]	**T**amaño de la muestra (ARRAYLIST)	**Merge Sort [ms]**	
5,00%	12568,00	55,00	**REQUERIMIENTO 5**
10,00%	21664,00	1085,00	
20,00%	38213,00	2040,00	
50,00%	83569,00	6040,00	





Esto indica que las gráficas encontradas de manera experimental concuerdan en cierta medida con los análisis de complejidad, sobretodo en el requerimiento 4, debido a que este

tenía mayor cantidad de for y fue el que tomó más tiempo para llevarse a cabo. Así pues, se puede concluir que el array list y el merge sort fueron probablemente las mejores decisiones para incurrir en una mayor eficiencia al momento de correr los algoritmos para los requerimientos dados.