OBSERVACIONES DEL RETO 1

Samuel Alejandro Jiménez Ramírez - 202116652
Marilyn Stephany Joven Fonseca - 202021346

Ambientes de pruebas

	Máquina 1	Máquina 2
Procesadores	Intel® Core™ i5 - 10310U CPU @ 1.70 GHz 2.21GHz 64 bits	AMD Ryzen 5 4500U Radeon Graphics 2.38GHz
Memoria RAM (GB)	16 GB (15,6 utilizable)	16 GB
Sistema Operativo	Windows 10 Pro	Windows 10 Home

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Resultados de las pruebas

Para tomar el tiempo promedio por ejecución de cada requerimiento acá enlistado, usamos el archivo con terminación **small.** Tomamos 3 muestras de cada requerimiento y sacamos la media aritmética que es la que está escrita en la tabla.

El tipo de guardado de datos que encontramos más optimo es **"ARRAY_LIST"**, aunque entendemos que es el que maneja memoria de forma menos óptima, es el más rápido cargando y nos permite ordenar también de una manera menos compleja temporalmente cuando es necesario.

Cabe destacar que el tiempo que se demora guardando datos del archivo csv con este tipo de estructura de datos es de **0.03125 s** en la **Máquina 1**, y de **0.015625s** para la **Máquina 2**.

	Tiempo promedio Máquina 1 (s)	Tiempo promedio Máquina 2 (s)
Requerimiento 1	0.109375	0.0625
Requerimiento 2	0.203125	0.140625
Requerimiento 3	0.015625	0.015625
Requerimiento 4	0.71875	0.52382
Requerimiento 5	0.046875	0.015625

Análisis complejidad temporal de cada requerimiento

Requerimiento 1:

En este requerimiento solo manejamos el catalogo para sacar información de el apartado de artistas. En general la complejidad se podría deducir de un MergeSort en esta lista de menor a mayor comparando

sus datos de adquisición, luego un recorrido a esta lista ordenada haciendo una sublista desde las fechas de inicio y fin dadas por el usuario. El resto de procesos es despreciable para el tiempo de ejecución.

Así que se podría decir que el tiempo está dado por **o((n*log(n)) + (n)).**

Requerimiento 2:

En este requerimiento se presentó un caso idéntico al anterior en el que la complejidad del algoritmo está dada por **o((n*log(n)) + (n))** ya que se realiza un ordenamiento por merge sort, y luego se vuelven a recorrer todos los datos sacando una sublista con los datos que están entre el rango deseado.

Requerimiento 3:

En este requerimiento se presenta un caso de recorridos **o(n_1+2n_2)** donde **n_1** es igual a la lista de artistas, y **n_2** es igual a la lista de artworks. No tienen ningún ordenamiento en el n general y el resto de procesos se puede tomar como procedimientos despreciables.

Requerimiento 4:

Este fue nuestro requerimiento más complejo en tema de implementación en medidas temporales. Primero hacemos un recorrido a todos los elementos de artworks **(n_1)** y luego por cada elemento obtenido en la busqueda **(x)** , hacemos un recorrido por cada elemento en **(n_2)** que se refiere al tamaño de artistas, dándonos una complejidad temporal de **o(n_1+xn_2)**. El resto de procedimientos es despreciable para la toma del tiempo ya que se dan sobre grupos muy pequeños o son constantes.

Requerimiento 5:

Este, junto con el requerimiento 3, fueron nuestros algoritmos más rápidos, la complejidad de este es solamente **o(n)** donde el recorrido se hace en **artworks**. El resto de recorridos y ordenamientos se hacen en una sublista más pequeña que es despreciable ya que puede variar según el tamaño de esta.

Conclusiones

- Teóricamente debería ser más eficiente el requerimiento 5 en comparación con el requerimiento 3. Pero al parecer los datos despreciables o constantes si tuvieron una importancia en el tiempo de ejecución de este código.
- Nuestras implementaciones buscaron siempre la mayor efectividad en cuanto a tiempo de respuesta, pero debido al tipo de lista que usábamos y los tipos de ordenamientos que en su totalidad fueron recursivos, estas implementaciones no serían viables en ambientes donde presentemos limitaciones de memoria.
- El requerimiento 3 tuvo el mejor rendimiento temporal en ambas máquinas, llegando a ser comparable con el tiempo de carga de los datos.