Isai Daniel Chacón Silva – 201912015 - Correo: id.chacon@uniandes.edu.co

Nicolás Aparicio Claros – 201911357 – Correo: n.aparicioc@uniandes.edu.co

Estructuras de datos y algoritmos

Reto 2

En este documento se realiza el análisis de complejidad para cada uno de los algoritmos implementados para resolver cada requerimiento del reto 2 de la materia Estructuras de datos y algoritmos. En primer lugar, cabe destacar que notamos que, a pesar de mejorar en los tiempos de ejecución, esto implicaba el uso de una mayor cantidad de estructuras de datos, lo cual incurría a su vez en un mayor uso de memoria RAM del dispositivo. Asimismo, vale recalcar que el análisis teórico se realizó con la notación Big O, por lo que en algunos casos los peores escenarios podían tener la misma complejidad en general entre este reto y el reto 1, sin embargo, en la práctica, dado que no siempre ocurre el peor escenario, resultó más eficiente utilizar los mapas de hash.

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

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 keys de la tabla de hash dada por catalog[“BeginDate”] y se verifica si cada una de estas llaves se encuentra dentro del rango dado por el usuario, lo cual incurre en un tiempo constante O(k) dado el uso de mapas. Así, el peor caso se tendría si el rango dado por parámetro cubre toda la base de datos de artistas (n), ya que el segundo recorrido se haría sobre toda la lista de artistas, es decir que se tiene una complejidad O(n). Luego de que se recorre esta lista, se agrega cada artista a una nueva lista en forma de ArrayList que ahora sí puede ser ordenada por medio de mergesort,

para el cual se asume nuevamente 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 ArrayList, 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 + k), lo cual se aproxima a O(n) por ser el polinomio más representativo.

Tabla 2. Resultado del tiempo de ejecución para el requerimiento 1, cambiando el tamaño del conjunto de datos.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Merge Sort [ms]	
20,00%	38213,00	440,00	**REQUERIMIENTO 1**
30,00%	53767,00	550,00	
50,00%	83569,00	730,00	
80,00%	125924,00	550,00	
100,00%	153373,00	750,00	

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 keys, verificando si cada una de estas llaves se encuentra dentro del rango dado por el usuario, lo cual incurre en un tiempo constante O(k) dado el uso de mapas. Así, el peor caso se tendría si el rango que entra por parámetro cubre toda la base de datos de artworks (m), ya que el segundo recorrido se haría sobre toda la lista de obras de arte para agregarlas a una nueva lista de tipo ArrayList, es decir que se tiene una complejidad O(m). Luego, esta lista con todas las obras es recorrida y ordenada 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 +k), lo cual se aproxima a O(m) por ser el grado más representativo de la ecuación.

Tabla 3. Resultado del tiempo de ejecución para el requerimiento 2, cambiando el tamaño del conjunto de datos.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Merge Sort [ms]	
20,00%	38213,00	1230,00	**REQUERIMIENTO 2**
30,00%	53767,00	3590,00	
50,00%	83569,00	4360,00	
80,00%	125924,00	9410,00	
100,00%	153373,00	9690,00	

Requerimiento 3

Para el caso de seleccionar la técnica, primeramente, se llama a la función *artist_technique.* Esta función recorre solamente el mapa almacenado en Catalog[‘artistsDict’] dado un nombre que entra por parámetro. Dicho mapa posee como key el nombre del artista, para así facilitar su búsqueda e interacción con el usuario, y una lista encadenada como value con todas las obras de dicho artista. En comparación con el reto 1, esta función solía realizar un doble ciclo para recorrer la lista de artistas como de obras de arte. Así pues, se observa cómo se ve reducida la dimensionalidad del problema por medio de los mapas. El orden de esta operación, por lo tanto, viene dado por O(m) (En realidad, este valor sería mucho menor en la práctica porque para que se dé este caso, implicaría que todas las obras del dataset están asociadas a un solo artista, lo cual es poco plausible).

Luego, se llama a la función *most_used_technique,* la cual recorre toda la lista de key sets de las técnicas utilizadas por el artista en cuestión, lo cual sería O(m) ya se que a su vez, se busca 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(2m+1), lo que se aproxima a O(2m).

Tabla 4. Resultado del tiempo de ejecución para el requerimiento 3, cambiando el tamaño del conjunto de datos.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Merge Sort [ms]	
20,00%	38213,00	380,00	**REQUERIMIENTO 3**
30,00%	53767,00	450,00	
50,00%	83569,00	480,00	
80,00%	125924,00	1100,00	
100,00%	153373,00	710,00	

Requerimiento 4

Este requerimiento para el reto anterior llamaba a 4 funciones, en donde se realizaba 2 veces un doble recorrido. Para este reto, se logró reducir de manera considerable el rendimiento temporal del algoritmo, así como se explica a continuación. Ahora se llama a la función *artist_nationality* la cual se encarga de recorrer todas las nacionalidades guardadas en el mapa catalog['nationalityDict']. Dada la notación, Big O, este procedimiento incurre en O(n) ya que se recorre todas las obras de arte para todas las nacionalidades almacenadas. De acuerdo con la implementación del mapa, es posible obtener el número de obras que pertenecen a cada nacionalidad. Adicionalmente, se creo una función denominada *topNationalityArtist*, la cual se encarga de determinar el nombre de los artistas con la nacionalidad con más obras y así, poder presentar sus nombres cuando se impriman en la vista. Cabe aclarar que para este punto se implementan dos recorridos, uno para estudiar las obras de la nacionalidad top y otro para recorrer los ConstituentID relacionados con los artistas que participaron para dicha obra. Esta función entonces consideraría un O(n^2).

En conclusión, para determinar la complejidad asignada a este requerimiento, se consideraría O(n+n^2), siendo O(n^2) el término más representativo que pone evidencia el comportamiento del algoritmo.

Tabla 5. Resultado del tiempo de ejecución para el requerimiento 4, cambiando el tamaño del conjunto de datos.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Merge Sort [ms]	
20,00%	38213,00	930,00	**REQUERIMIENTO 4**
30,00%	53767,00	2660,00	
50,00%	83569,00	3550,00	
80,00%	125924,00	4110,00	
100,00%	153373,00	5330,00	

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 para un departamento dado por medio del uso de mapas, en particular: catalog['Department'].

También se realizan 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, se tendría que 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.

Este requerimiento fue el que tomó más tiempo para este reto, a diferencia del reto 1 en el cual el requerimiento 4 era el que más recursos computacionales gastaba. El comportamiento temporal de este requerimiento (así como se observa posteriormente en la Figura 1) es linearítmico debido a los ordenamientos realizados.

Tabla 6. Resultado del tiempo de ejecución para el requerimiento 5, cambiando el tamaño del conjunto de datos.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Merge Sort [ms]	
20,00%	38213,00	2110,00	**REQUERIMIENTO 5**
30,00%	53767,00	5680,00	
50,00%	83569,00	8350,00	
80,00%	125924,00	10710,00	
100,00%	153373,00	12340,00	

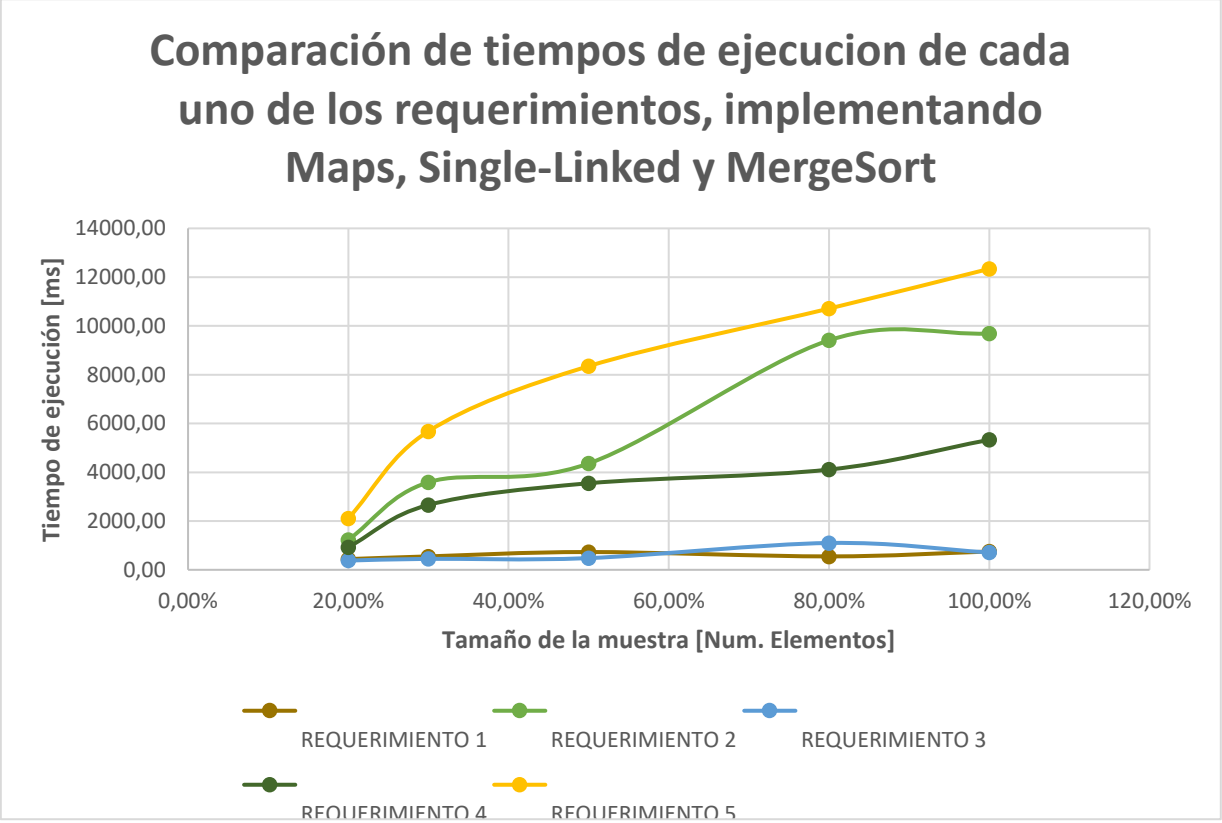

Figura 1. Comportamiento del tiempo de ejecución frente al porcentaje de datos cargado para cada uno de los requerimientos del reto 2.

Esto indica que las gráficas encontradas de manera experimental concuerdan en gran medida con los análisis de complejidad. Además, el haber implementado los mapas influyó en gran medida sobre los tiempos de respuesta para buscar la información en la memoria del computador. Así bien, se logra concluir que los mapas son estructuras de datos más versátiles que las listas debido a que permiten que la implementación pareja key:value sea aprovechada de manera óptima por el programador para acceder más rápidamente a la información, dada una serie de inputs por parte del usuario.

También se observa que el uso de mapas permitió la correcta implementación de requerimientos bastante óptimos, dado que como muestra la figura. 1, estos crecen o linealmente, o linearítmicamente, lo que va acorde a la discusión previa.