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.

**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.

**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).

**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.

**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.