Camilo Ortiz Cruz 201821615 [c.ortizc@uniandes.edu.co](mailto:c.ortizc@uniandes.edu.co)

Kevin Fernando Gómez Camargo 202015120 [k.gomezc@uniandes.edu.co](mailto:k.gomezc@uniandes.edu.co)

**Maquinas**

|  |  |  |
| --- | --- | --- |
|  | **Kevin Fernando** | **Camilo** |
| **Procesador** | 2,5 GHz Intel Core i5 de dos  núcleos | 11th Gen Intel® Core™ i7-1165G7 @ 2.80Ghz |
| **Memoria RAM** | 4 GB | 16 GB |
| **Sistema Operativo** | macOS Catalina 10.15.7 | Windows 10 Home 64-bits |

**Pruebas de tiempo**

Tabla pruebas de tiempo de Camilo.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Columna1** | **artists** | **artworks** | **req1p** | **req2p** | **req5p** | **Req1** | **Req2** | **Req4** | **Req5** |
| small | 1948 | 768 | 862 | 316 | 394 | 0 | 0 | 0 | 9,38 |
| 10pct | 6656 | 15008 | 3231 | 6504 | 8133 | 9,38 | 15,625 | 0 | 135,24 |
| 30pct | 10063 | 43704 | 5031 | 19251 | 23709 | 15,635 | 34,52 | 0 | 430,53 |
| 50pct | 12137 | 71432 | 6117 | 32009 | 38888 | 15,625 | 46,875 | 0 | 750,67 |
| large | 15223 | 138150 | 7664 | 61484 | 76117 | 15,625 | 65,55 | 0 | 1470,25 |

Los gráficos están al final del documento.

**Requerimiento 1:**

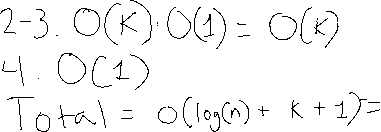
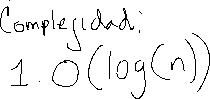
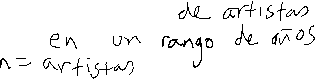
Para hacer el requerimiento 1 se consideró pasar todos los años de nacimiento de los artistas como índices en un mapa y los valores que fueran listas con la información de los artistas, sin embargo, esta idea se descartó pues se consideró que la complejidad temporal era similar a la de la solución planteada en el reto 1.

La solución propuesta es la siguiente:

1. Hacer búsqueda de ceil para encontrar la pos donde arranca la fecha límite inferior
2. Recorrer los artistas entre la posición del ceil hasta la última posición donde se tenga una fecha menor o igual al límite superior



1. Para los primeros tres elementos encontrados agregarlos a una lista
2. Basado en la última posición encontrada antes de que el año sea mayor al límite hacer una sublista de los últimos 3 elementos.



La razon por la cual hacerlo con el mapa resultaria similar se debe a que debido a que tenemos un rango y las llaves en el mapa no estan ordenadas, tendriamos que recorrer todo el mapa para encontrar las llaves que esten en el rango y adicional a esto tendriamos que buscar el floor de esas llaves, esto aunque no requiere de una implementacion muy dificil, implica ordenar todas las listas de todas las llaves y conlleva a un espacio adicional en memoria, adicionalmente en terminos de rendimiento en el reto 1, el algoritimo se ejecuto en un tiempo muy bajo de entre 10 y 30 ms con los datos large, por lo cual el trade off de reducir un poco esta velocidad pero duplicando los datos de los artistas no lo lo vale.

**Requerimiento 2:**

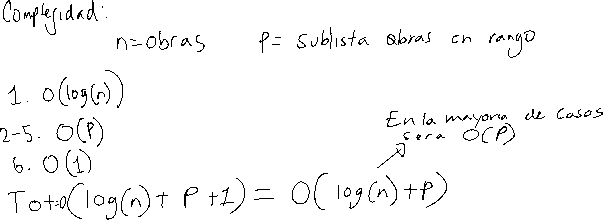
Para el requerimiento 2 utilizamos el algoritmo que habiamos planteado en el reto 1 pero en este caso aprovechamos los mapas y creamos uno que tiene como llave el ConstituentID del artista y como valor sus datos, lo cual nos permitio convertir nuestro algoritmo en lineal, lo cual consideramos que es eficiente en base a que en el reto 1 con un algoritmo linearitmico los tiempos con large estuvieron entre 74 y 500 ms, al pasar a lineal el algoritmo sera mucho mas rapido y dado que en el reto 1 el tiempo ya era bueno, con la nueva implementación este sera mucho mejor.

El algoritmo es el siguiente:

1. Buscar las pos del ceil de la fecha de compra del limite inferior
2. Recorrer desde pos del ceil hasta la pos del máximo valor dentro del límite superior
3. Para cada obra sumar en una variable si fue comprada y sumar al número de obras en el rango y sumar al número de artistas y tener una variable que calcule la máxima posición de las obras dentro del rango



1. Por cada id de las primeras 3 obras hacer un get del mapa creado que tiene como índice el id y agregar los nombres de los artistas a la obra
2. Agregar las primeras 3 obras a una lista
3. Crear una sublista de las ultimas 3 obras y por cada id de cada obra hacer get y agregar los nombres a la obra



Consideramos que el algoritmo es optimo ya que para poder cumplir con el requerimiento siempre se debera recorrer todos los elementos del subconjunto p, ya que se debe contar numero de artistas y obras compradas (Purchase), en un mundo ideal tendriamos un mapa con el rango que deseamos y este tiene la lista con las obras, sin embargo, implementar esto entra en las complejidad factorial o mas lo cual para las obras, el computador no podria hacer, por tanto llegar a una solucion O(p) es optima dado lo posible.

**Requerimiento 3 (Kevin Fernando):**

Para el requerimiento 3 me basé en el reto 1, pero con algunas modificaciones. Se utilizó :

**Requerimiento 4 (Camilo):**

Para el requerimiento 4 decidi reutilizar el codigo que hice para el Reto 1 pero utilizando el mapa de ConstituentID de tal forma que no hay que hacer una busqueda binaria por cada id de cada obra, pero esta vez decidí realizar todos los calculos en la carga de los datos dado que este requerimiento no necesita del input del usuario por lo cual no es logico volver a correrlo cada vez que el usuario lo pida, esto implica que la complejidad del algoritmo es de pues los datos ya estan en el catalogo y lo unico que debe hacerse es mostrarselos al usuario, el algoritmo en la carga de datos es el siguiente:



0.Crear un mapa de obrasUnicasxNacionalidad y un mapa de obrasTotalesxNacionalidad v

1.Recorrer todas las obras



2. Por cada obra recorrer los id y por cada id hacer un get del mapa de ids



3.Agregar el nombre del artista a la obra



4. Guardar la naciocionalidad del artista



5. En un mapa de nacionalidades de artistas de la obra, si esta ya tiene la nacionalidad (contains) en el valor sumar 1, si no tiene la nacionalidad hacer put de la nacionalidad con valor 1



7.Recorrer los keys de las nacionalidades de los artistas de la obra y si la nacionalidad se encuentra en los mapas creados en el paso (0), hacer addLast a obrasUnicas la obra y a obrasTotales sumar el valor (# artistas de esa nacionalidad en la obra), si no esta, a obrasUnicas hacer put de la nacionalidad y de valor una lista y hacer addlast a la lista con la obra, en obrasTotales hacer put con la nacionalidad y de valor el valor de la llave de la nacionalidad de los artistas de la obra.

8. crear una lista de tuplas que tenga el nombre del pais y el numero de obras Totales por nacionalidad

9. Hacerle sort a la lista de tuplas

10. Obtener el primer elemento de la lista de tuplas ordenada

11. Obtener la lista del pais con mas obras unicas de la llave obtenida en el paso 11.



Como se puede observar se logro reducir la complejidad a un tiempo lineal en el procesamiento lo cual es un buen tiempo que no afectara mucho la carga de datos y como tradeoff por esta demora en procesamiento, a la hora de ejecutura el requerimiento este sera en O(1).

**Requerimiento 5:**

En el caso de requerimiento 5 otra vez nos basamos en el codigo creado en el Reto 1 pero con 3 modificaciones importantes,la primera fue poner en un mapa con llave departamento la segunda fue utilizar el mapa de ConsituentID para no tener que hacer busqueda binaria por cada obra lo cual reduce en la primera parte la complejidad a tiempo lineal y la tercera modificación fue en la parte de ordenar las listas de antiguos y mas caros, en este caso se opto por no ordenar todo ya que esto tiene una complejidad de n\*log(n) (mergesort) lo cual para datos grandes toma un tiempo considerable, es por eso que se decidio hacer un selection sort modificado que solo ordene las primeras 5 posiciones, de tal modo la complejidad se vuelve n\*5, para saber si el cambio lo vale es necesario mirar en que punto log(n) es mayor que 5 y esto sucede si usamos log\_2 cuando n > 32, esto lo podemos ver en una gráfica:

Gráfico, Gráfico de líneas

Descripción generada automáticamenteGráfico, Gráfico de líneas

Descripción generada automáticamente

Donde se ve el punto donde log(n) sobrepasa a 5, es lógico que 5n se comporte mejor que n\*log(n) ya que el valor de log va a cambiar dependiente del numero de datos, y dado que todos los departamentos tienen mas de 32 obras entonces el cambio lo vale.

El algoritmo utilizado fue el siguiente:

1.Hacer get del departamento que dio el usuario



2.Recorrer el arreglo de las obras de ese departamento

3. Por cada obra sumar al total, sumar al peso, calcular costo y sumarlo al total de costo y al costo de la obra



4. Por cada obra recorrer los ids y hacer get en el mapa de constituentID y agregar el artista a la obra



5. agregar (addLast) la obra a 2 listas, una para antiguos solo agregar si el date es diferente de 0, y a la otra lista de precio agregar todos



6. Hacer selection sort modificado para encontrar el top5 de las 2 listas (antiguos y precio)



Como se puede ver en la complejidad logramos un tiempo lineal, esto consideramos que es optimo ya que para poder solucionar el requerimiento es necesario recorrer todas las obras del departamento para poder saber el precio de cada una, no obstante, una posible mejora es hacer todo en la carga de datos, ya que de este modo correr el requerimiento seria O(1) pero esto implicaría en el procesamiento correr el algoritmo para cada departamento y almacenar el resultado de cada departamento, lo cual consideramos que no lo valía dado el consumo de memoria y una mayor demora en el tiempo de carga.

Gráficos: