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.

Es importante resaltar que la función 4 solo se corre una vez, estos tiempos mostrados se corren en la carga de datos y después correr el requerimiento para el usuario es O(1), es decir 0ms.

Imagen que contiene texto, exterior, marcador, camino

Descripción generada automáticamente

Tabla pruebas de tiempo de Kevin Fernando.

Tabla

Descripción generada automáticamente con confianza media

Los gráficos están al final del documento.

**Requerimiento 1:**

Para resolver el requerimiento se creó un mapa que tiene como llave el año de nacimiento de los artistas y como valor una lista con los artistas que nacieron en ese año.

El algoritmo funciona de la siguiente manera:

1. Se obtienen las llaves del mapa de las fechas de nacimiento y se crea una lista vacía

Después de este paso el algoritmo tiene dos opciones.

La primera y mejor caso es:

1. Si la diferencia entre la fecha inicial y la final es menor o igual al numero de llaves en el mapa se inicia
2. Se recorre desde la fecha inicial hasta la fecha final (range)
3. Por cada fecha si esta contenida en el mapa se recorre la lista que contiene los artistas de esa fecha y se agregan a la lista creada

Segundo, peor caso:

1. Si la diferencia es mayor
2. Se recorren todas las llaves
3. Por cada llave si está dentro del rango se recorre la lista del valor de la llave y se agrega el artista a la lista creada
4. Una vez se recorre todo se ordena la lista creada

Para cualquiera de los dos casos

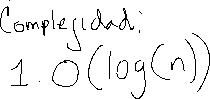
1. Se hace 2 sublist para los primeros 3 y últimos 3 elementos de la lista

Texto

Descripción generada automáticamente

Este algoritmo al necesitar usar mapas pero el algoritmo pedir elementos ordenados es menos eficiente que el algoritmo del reto 1, , esto se debe a que en el reto 1 el único elemento que había era O(k) pues al estar ordenada la lista solo se debía encontrar el ciel y después iterar sobre el numero de elementos, en este algoritmo con mapas hay mas valores que impactan el tiempo, es evidente que para temas de orden de todos los datos son mejores las listas que los mapas, o por lo menos en este caso. Aunque el tiempo no fue tan diferente en el caso de que el usuario ponga un rango de años muy grandes los resultados serán peores a los observados en el reto.

**Requerimiento 2:**



Para este requerimiento se creo un mapa con indice las fechas de adquisicion y como valor una lista con las obras que tuvieran esa fecha, adicionalmente se creo un mapa de los constituentid de los artistas y de valor el artista.

El algoritmo es el siguiente:

1. Obtener una lista con las llaves del map de fechas y crear una lista vacia

Este algoritmo tiene un mejor caso y un peor, el mejor caso es el siguiente:

1. La diferencia en dias de la fechas es menor que el numero de llaves en el map
2. Se recorre desde la fecha inicial hasta la fecha final
3. Si la fecha esta contenida en el mapa se reocrre la lista de obras de esa fecha y se agrega a la lista creada

Peor caso:

1. La diferencia es mayor
2. Se recorren todas las llaves
3. Por cada llave si está dentro del rango se recorre la lista del valor de la llave y se agrega el artista a la lista creada
4. Una vez se recorre todo se ordena la lista creada

Para cualquiera de los dos:

1. Sacar 2 sublista de los primeros 3 y ultimos 3
2. Para cada sublista y por cad obra de la sublista recorrer los ids y hacer un get del id en el mapa de ids y sacar el nombre del artistas, agregarlo a una lista y agregar la lsita con todos los nombres a la obra

Texto

Descripción generada automáticamente

Este algoritmo resulto eficiente pero no es lo mas eficiente posible, el algoritmo mas eficiente seria utilizar la implementacion del reto 1 pero con un mapa para encontrar los nombres de los artistas esto seria una complejidad de O(k) lo cual es lo minimo posible para este problema, sin embargo al no poder utilizar la lista como estructura principal fue necesario crear mapas con sub lista según la fecha de adquisicion, esto como se puede ver en la complejidad crea terminos estra en la complejidad del problema lo cual lo hace mas demorado en el tiempo de ejecucion, adicionalmente en este caso el mejor caso casi no se da ya que al ser diferencias de dias entre normlamente muchos años hay miles de datos en el rango de dias entre las dos fechas y el numero de fechas en el mapa no es tan grande razon por la cual la diferencia de tiempos entre el reto 1 y reto 2 fue tanta.

**Requerimiento 3 (Kevin Fernando):**

Para el requerimiento 3 me base en el reto 1, pero con algunas modificaciones. Se utilizaron dos indices, el primero tiene el nombre del artista como llave y el ConstituentID como valor, y el segundo tiene el ContituentID del artista como llave y como valor un arreglo con las obras asociadas:

1. Crear indices especificados
2. Encontrar el constituent ID del artista usando el primer indice
3. Encontrar el arreglo con las obras del artista usando el segundo indice
4. Crear un map que tenga los medios como llaves y como valor un arreglo con la información necesaria de cada obra con ese medio
5. Recorrer el arreglo
6. Si el map no contiene como llave el medio de la obra en que va la iteración, se crea el diccionario con la información, se crea un arreglo y se coloca como valor de la llave medio de la obra, y se agrega el diccionario al final del arreglo
7. Si el mapa ya contiene como llave el medio de la obra en que va la iteración, se crea el diccionario con la información y se añade al final del arreglo
8. Si ese arreglo es el más grande, se actualiza mayor y el medio más usado
9. Se hace MergeSort del arreglo más grande
10. Si el arreglo más grande tiene más de 3 obras, se crean dos sublistas del arreglo ya ordenado para las 3 primeras y las 3 ultimas

Complejidad:

Complejidad total:

La complejidad lograda en el reto 1 fue de orden O(z) siendo z el numero de obras. Por otro lado, la complejidad en el reto 2 es de O(n)+O(mlogm) siendo n el numero de obras de un artista en especifico y m el numero de obras del medio más usado por ese artista. Debido a que n << z porque hay 15220 artistas y a que m < n, entonces m << z, la implementación del requisito en el reto 2 debería ser mejor que la del reto 1.

**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).

Es importante resaltar que los tiempos de ejecucion para esta implementacion fueron mayores que los del reto 1, esto es posible que se deba a que en el reto 1 se utlizaron diccionarios nativos de Python, es muy posible que los diccionarios de python sean mucho mas rapidos que los mapas del ADT por lo cual el rendimiento se ve afectado.

**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



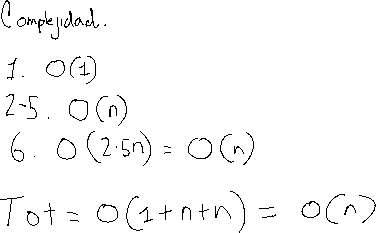
4. Por cada obra recorrer los ids y hacer get en el mapa de constituentID y agregar el artista a la obra



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.

En comparación al reto 1 este algoritmo fue mucho más rápido mejorando casi un 50% , esto se debió al uso de mapas para encontrar los nombres de los artistas que redujo a n la complejidad nlog(m) y el uso de selection que cambio nlog(n) a n, en este caso donde se necesita la información de un elemento en común de los datos los mapas fueron muy útiles pues permitieron encontrar de manera eficaz la información.

**Gráficos**:

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente