Reto 1

Req. 3 - Carlos Arturo Holguín Cardenás, 202012385, c.holguinc@uniandes.edu.co

Req. 4 - Daniel Hernández Pineda, 202013995, d.hernandez24@uniandes.edu.co

IMPORTANTE: Con la autorización del profesor, se hizo uso de listas nativas de Python en el archivo view.py para poder hacer uso de la librería "tabulate". Para que esta librería funcione adecuadamente, se recomienda usar la versión 3.7 de Python en el intérprete, además de instalarla con el comando "pip install tabulate" en caso de no tenerla instalada

Requerimiento 1

Preámbulo

Entiéndase: m = total de artistas; **x** = número de fechas de nacimiento. Para este requerimiento se cargó en el catálogo un mapa llamado MapReq1, cuyas llaves son fechas de nacimiento de artistas y sus valores son listas de los artistas nacidos en aquellas fechas. Además de esto, se cargó una lista de las llaves del mapa, la cual se ordenó de menor a mayor año.

Análisis de complejidad

Para desarrollar este requerimiento se hace uso de la función REQ1(). Inicialmente, se implementa una búsqueda binaria con el fin de encontrar la posición en la que se encuentra el primer año que se encuentra en el rango ingresado, realizando **log(x)** recorridos. Posteriormente, se recorre la lista de fechas de nacimiento desde la posición ya encontrada. Dentro de este ciclo, para cada fecha, se accede a su lista de artistas correspondiente en el MapReq1* y se recorre cada artista para almacenarlo en la respuesta final. Si bien se trata de un recorrido doble, en realidad se están recorriendo los artistas; el estar clasificados por fechas no aumenta el número de ciclos realizados. Por esta razón, en el peor caso, se realizan **m** recorridos.

De esta manera, para el requerimiento 1 se realizan **log(x) + m** recorridos en el peor caso. Se reduce entonces la complejidad a **O(m)**.

*La búsqueda en un hashmap tiene orden de crecimiento que se asume constante, una buena aproximación según lo enseñado en clase para los métodos mp.get() y me.getValue() aunque se trate de un probehashmap.

Comparación con Reto 1

Complejidad en el Reto 1: **O(m)**

Si bien la complejidad en notación Big O es la misma por tener una implementación tan similar, los tiempos de ejecución son más pequeños para este reto pues existe diferencia en los factores adicionales. En otras palabras, los menores tiempos tienen sentido debido a que en el reto 1 se

realizaban **log(m) + (3/2)m** recorridos en el peor caso, mientras que en este reto se realizan **log(x) + m**.

Pruebas de tiempo

Se utilizaron las siguientes entradas para recrear el peor caso:

- Año Inicial = 0
- Año Final = 2022

A continuación, los resultados obtenidos:

		Tiempo [ms]	
Archivo	**# artistas (m)**	**Daniel**	**Carlos**
small	1948	9,38	0,00
5pct	4996	12,50	0,00
10pct	6656	15,63	0,00
20pct	8724	18,75	15,62
30pct	10063	23,44	15,62
50pct	12137	20,32	15,62
80pct	14143	25,00	18,00
large	15223	23,96	23,40

Tabla 1. Pruebas de rendimiento del primer requerimiento

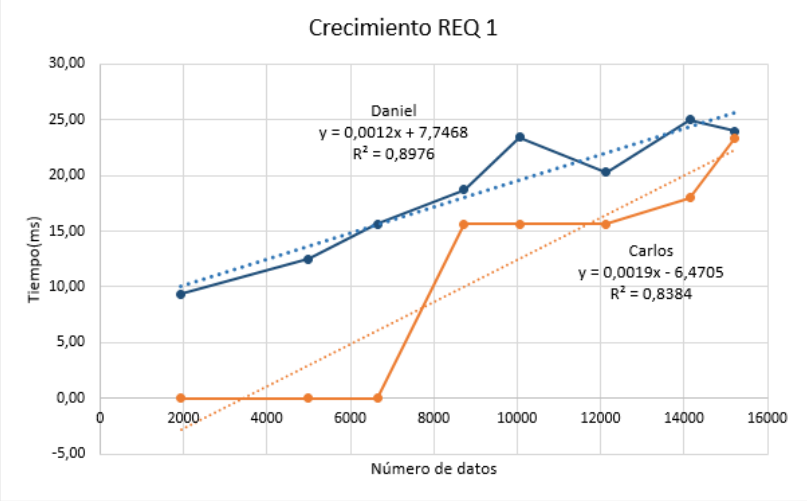

Figura 1. Gráfico del comportamiento del primer requerimiento (regresión lineal)

Aunque en las pruebas realizadas por Daniel Hernández parecería observarse un patrón de crecimiento (que se ajusta a una regresión lineal con ligera dificultad), en las pruebas realizadas por Carlos Holguín es más complicado evidenciar un patrón de crecimiento. Esto se puede atribuir, sin embargo, a la precisión del cronómetro de la librería "time" utilizada para estas mediciones, dado que el mínimo valor que arroja (distinto de cero) son 15.625 milisegundos, aproximando a este los valores que posiblemente habrían mostrado un comportamiento distinto en el gráfico de Carlos.

Requerimiento 2

Preámbulo

Entiéndase: **n** = total de obras; **y** = número de fechas de adquisición. Para este requerimiento se cargó en el catálogo un mapa llamado MapReq2, cuyas llaves son fechas de adquisición y sus valores son listas de las obras adquiridas en aquellas fechas. Además de esto, se cargó una lista de las llaves del mapa, la cual se ordenó de menor a mayor año.

Análisis de complejidad

Para desarrollar este requerimiento se hace uso de la función REQ2(). Inicialmente, se implementa una búsqueda binaria con el fin de encontrar la posición en la que se encuentra la primera fecha que se encuentra en el rango ingresado, realizando **log(y)** recorridos. Posteriormente, se recorre la lista de fechas de adquisición desde la posición ya encontrada. Dentro de este ciclo, para cada fecha, se accede a su lista de artistas correspondiente en el MapReq2* y se recorre cada obra para almacenarla en la respuesta final. Si bien se trata de un recorrido doble, en realidad se están recorriendo las obras; el estar clasificadas por fecha de adquisición no aumenta el número de ciclos realizados. Por esta razón, en el peor caso, se realizan **n** recorridos.

De esta manera, para el requerimiento 2 se realizan **log(y) + n** recorridos en el peor caso. Se reduce entonces la complejidad a **O(n)**.

Comparación con Reto 1

Complejidad en el Reto 1: **O(n)**

Si bien la complejidad en notación Big O es la misma por tener una implementación tan similar, los tiempos de ejecución son más pequeños para este reto pues existe diferencia en los factores adicionales. En otras palabras, los menores tiempos tienen sentido debido a que en el reto 1 se realizaban **log(n) + n** recorridos en el peor caso, mientras que en este reto se realizan **log(y) + n**.

Pruebas de tiempo

Se utilizaron las siguientes entradas para recrear el peor caso:

- Fecha Inicial = 1900-01-01
- Año Final = 2022-12-31

Los resultados obtenidos fueron:

		Tiempo [ms]	
Archivo	# obras (n)	Daniel	Carlos
small	768	6,25	15,62
5pct	7572	18,75	15,62
10pct	15008	31,25	31,25
20pct	29489	60,94	46,87
30pct	43704	93,75	78,12
50pct	71432	134,38	119,67
80pct	111781	223,96	185,50
large	138150	271,88	230,25

Tabla 2. Pruebas de rendimiento para el segundo requerimiento

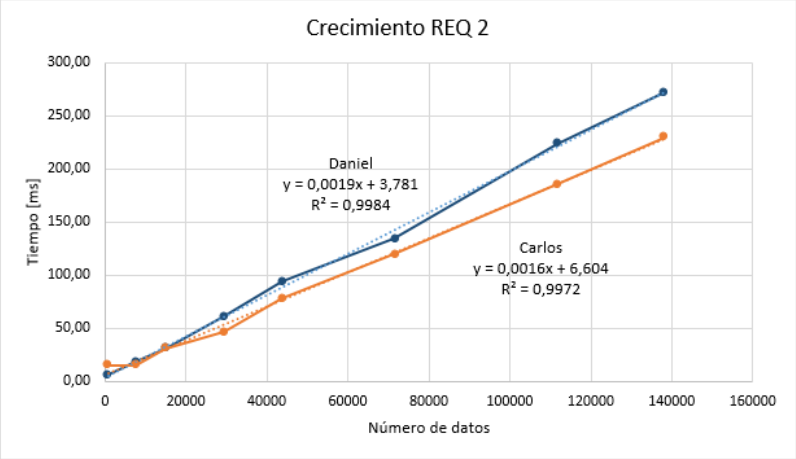

Figura 2. Gráfico del comportamiento del segundo requerimiento

Como se observó en las pruebas de rendimiento, el comportamiento del algoritmo es claramente lineal, confirmando el análisis de complejidad realizado previamente. La distinción entre las pruebas realizadas por cada miembro del grupo tiene que ver con la máquina de la que cada uno disponía, pero para ambos se tuvo un comportamiento acorde con el O(n) anunciado teóricamente.

Requerimiento 3 (implementado por Carlos Holguín)

Preámbulo

Entiéndase: **m** = size(artists); **n** = size(artworks); x=#de obras; z=#de medios de un artista; Para este requerimiento se cargó en el catálogo un mapa llamado NameidREQ3, cuyas llaves son los nombres de los artistas presentes en el archivo de artistas y sus valores son los constituentID de cada autor, por tanto, cuenta con **m** datos. Además, se cargó en el catálogo un mapa llamado IdWArtworkREQ3, cuyas llaves son el ID presentes en el archivo de artworks y sus valores son los títulos de la obras, por tanto, cuenta con **n** datos. Finalmente, se cargó en el catálogo un mapa llamado AddTitleAndDataREQ3,

cuyas llaves son los títulos de las obras presentes en el archivo de artworks y sus valores son una lista de cada título, fecha, medio y dimensiones, por tanto, cuenta con **n** datos.

Análisis de complejidad

Para desarrollar este requerimiento se hace uso de 2 funciones en *GetTechniquesReq3()* en primer lugar, se hace dos veces uso del método *get()* obteniendo el ID y una lista con los títulos, por tanto, se obtiene una complejidad constante de 5, posteriormente, se hace un recorrido en la lista de interés con los títulos de las obras, este representa x de complejidad temporal en el peor de los casos, ya que, en este ciclo se recorre el nombre de las obrar del artista de interés. Posteriormente, se hace uso de *get()* para obtener el valor de cada iteración en la lista esto tiene una complejidad de 1, también se usa un *mp.get()* con complejidad de 5, posteriormente, se llama a la función *CreateTableHashREQ3(),* en esta solo hay operaciones constantes de complejidad 5 con el *get()* y con el *put()* de 5.

Posteriormente, se sale de este ciclo y se hace uso del método *keyset()* para recorrer las llaves con los medios del mapa creado con la función *CreateTableHashREQ3(*), con una complejidad máxima de z, posteriormente se hace uso de *get()* con una complejidad de 5, finalmente se sale del ciclo para realizar el ultimo *get()* que nos dará los datos de interés, este posee una complejidad. Notemos que la complejidad en todos los casos fue lineal y cuando se realizó algún ciclo, este poseía una cantidad baja de elementos a recorrer, por tanto, estos recorridos eran muy rápidos.

Nota:

Obteniendo:

5+x (1+5+5+5) +z (5)+5

En notación Big O se obtiene **O(x)**

Comparación con Reto 1

Complejidad en el Reto 1: **O ($n^{\frac{3}{2}}$)**

En comparación con el reto 1 se observa que el reto 2 posee una menor complejidad temporal, por otro lado, en el reto 2 casi se obtiene un algoritmo constante, esto se puede evidenciar en los tiempos de búsqueda en comparación con los tiempos del reto 1.

Pruebas de Tiempo

Entradas del caso: **Louise Bourgeois**

Archivo	# obras (n)	tiempo [ms]
small	768	0
5pct	7572	0
10pct	15008	0
20pct	29489	15.62
30pct	43704	16
50pct	71432	25.25
80pct	111781	46.87
large	138150	63

Tabla 3. Pruebas de rendimiento del tercer requerimiento

Figura 3. Gráfico del comportamiento del tercer requerimiento

Por otro lado, en la Figura 3, se observa un comportamiento que casi lineal, observando que la complejidad temporal estimada es muy cercana a la que se estimó, por otro lado, el algoritmo posee unos tiempos bastante bajos

Requerimiento 4 (implementado por Daniel Hernández)

Preámbulo

Entiéndase: m = total de artistas; **n** = total de obras; **z** = #nacionalidades; **x** = máximo de autores en una obra; **y** = máximo de artistas pertenecientes a la misma nacionalidad. Si bien es teóricamente posible que **x,y,z** tengan el mismo tamaño que **m**, estos valores serán siempre de un orden significativamente menor que **m** por la naturaleza de los datos utilizados (aunque también aumentan a medida que **m** aumenta). Por esta razón, no tiene mucho sentido tratarlos como si fuesen **m**.

Para este requerimiento se cargó en el catálogo un mapa llamado MapReq4, cuyas llaves son las nacionalidades presentes en el archivo de artistas y sus valores son listas de las obras cuyos autores pertenecen a la nacionalidad respectiva. Cabe resaltar que, en caso de que una obra tenga más de un

artista de determinada nacionalidad, entonces la obra aparecerá más de una vez en la lista de obras de aquella nacionalidad en el MapReq4.

Análisis de complejidad

Para este requerimiento se crearon tres funciones: counterREQ4(), getCountListREQ4() y REQ4().

La función counterREQ4() recibe como parámetro una nacionalidad y, recurriendo al MapReq4, extrae la lista de obras correspondientes a aquella nacionalidad. Luego, calcula el tamaño de la lista. Por lo tanto, esta función tiene orden de crecimiento constante.

Por su parte, la función getCounListREQ4() invoca el método mp.keySet() para extraer la lista de nacionalidades. Al ser el MapReq4 de tipo linear probing con un factor de carga de 0.5, este método realiza aproximadamente **2z** ciclos, pues recorre todas las posiciones de la tabla de hash (incluyendo las vacías). Posteriormente, se recorre esta lista de nacionalidades, invocando la función counterREQ4() y almacenando cada nacionalidad con su conteo calculado en una lista. Este último recorrido corresponde a z ciclos. Finalmente, se ordena la última lista mencionada según el conteo obtenido para cada nacionalidad, añadiendo un **zlogz** adicional. Así, la función realiza alrededor de **3z + zlogz** ciclos en total.

Por último, la función REQ4() invoca a la función getCounListREQ4(), para luego extraer del MapReq4 la lista de obras correspondientes a la nacionalidad con más obras según la información retornada por getCounListREQ4(). Se puede ver, entonces, que para el requerimiento 4 se realizan aproximadamente **3z + zlogz** ciclos. Esto se puede resumir en una complejidad de **O(zlogz)**

Comparación con Reto 1

Complejidad en el Reto 1: **O(nxyz)**

Se puede observar una reducción importante en la complejidad del algoritmo, notando que el producto **nxy** tiene un valor mucho más grande que **log(z)** por lo explicado en el preámbulo. Esto se debió a que, en el reto anterior, era necesario recorrer el catálogo de obras completo por cada nacionalidad para identificar qué obras incluir en el conteo. Para este reto, esta información que relaciona las nacionalidades con las obras ya se encuentra cargada en el catálogo, volviendo el algoritmo mucho más eficiente; pasó de demorarse varios minutos en el reto 1 a demorarse milésimas de segundo en el reto 2.

Pruebas de tiempo

Se obtuvieron los siguientes resultados:

Archivo	# obras (n)	# nacionalidades (z)	tiempo [ms]
small	768	48	3,12
5pct	7572	84	6,25
10pct	15008	92	12,50
20pct	29489	102	9,38
30pct	43704	105	12,50
50pct	71432	112	15,63
80pct	111781	118	6,25
large	138150	118	9,38

Tabla 4. Pruebas de rendimiento para el cuarto requerimiento

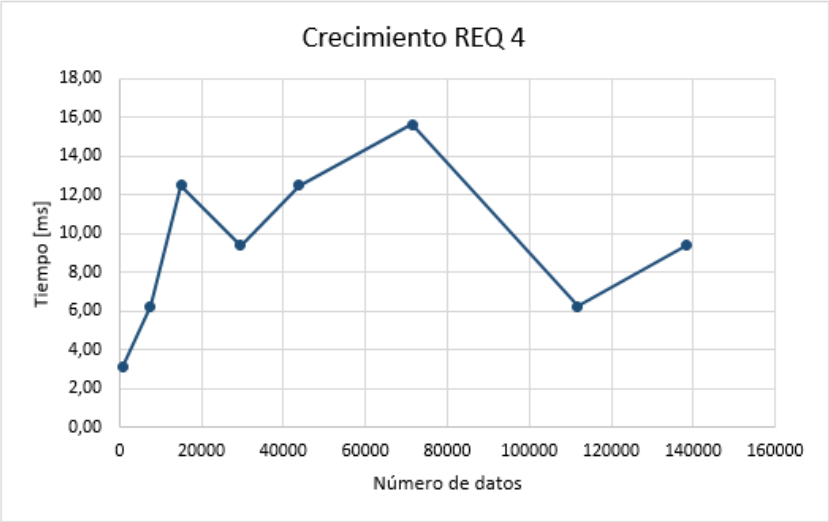

*Figura 4.1. Gráfico del comportamiento del cuarto requerimiento respecto a **n***

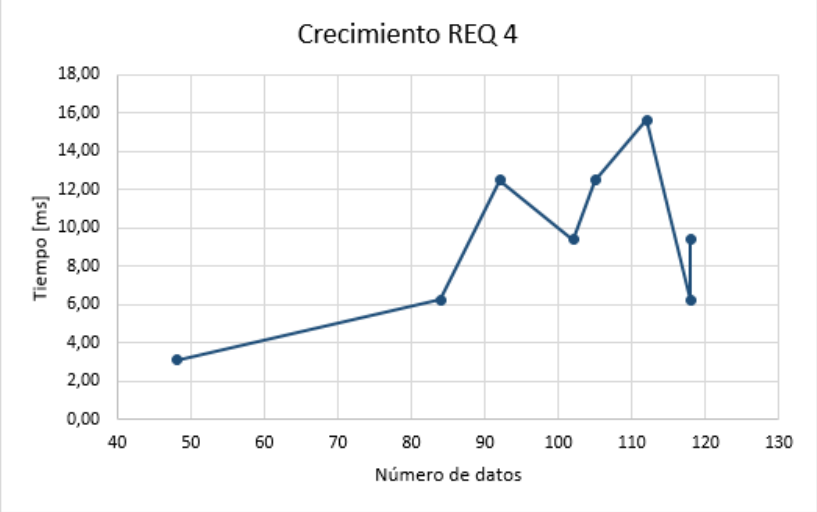

*Figura 4.2. Gráfico del comportamiento del cuarto requerimiento respecto a **z***

Respecto a las mediciones realizadas, en ninguno de los dos gráficos se observa ningún patrón claro en el comportamiento del tiempo de ejecución de este requerimiento; tanto respecto al número de obras (n) como respecto al número de nacionalidades (z) se muestra un comportamiento que no se aproxima a ningún modelo matemático.

Así como en el requerimiento 1, se identifica como causa más probable la precisión del cronómetro de la librería "time" utilizada para estas mediciones, dado que el mínimo valor que arroja (distinto de cero) son 15.625 milisegundos, aproximando a este los valores que posiblemente habrían mostrado comportamientos distintos en los gráficos.

Requerimiento 5

Preámbulo

Entiéndase: **n** = total de obras; **x =** número de obras del departamento. Aunque teóricamente **x** pudiese ser del mismo tamaño de **n**, los datos con los que se está trabajando hacen que **x** sea considerablemente menor que **n** para archivos de cualquier tamaño.

Para este requerimiento se cargó en el catálogo un mapa llamado MapReq5, cuyas llaves son los departamentos del museo y sus valores son listas de las obras correspondientes a cada departamento.

Análisis de complejidad

Para este requerimiento se hace uso de cuatro funciones: calculateDimensionsReq5(), calculateSingularCostReq5(), addTOP5Req5(), REQ5().

La función calculateDimensionsReq5() calcula las dimensiones físicas (área o volumen) de cada obra dependiendo la información que se brinde (calculará volumen siempre que se tenga información suficiente, y área cuando solo se tengan dos valores de longitud o el diámetro). Esta tiene orden de crecimiento constante, puesto que solo hay un ciclo y que itera únicamente 5 veces, por tanto, se aproxima. La función addTOP5Req5() también realiza 5 iteraciones en cualquier caso.

La función calculateSingularCostReq5() se encarga de calcular el costo de la obra dadas sus dimensiones físicas y su peso. Esta función tiene orden de crecimiento constante, ya que solo hay condicionales y operaciones matemáticas que permiten determinar el costo.

La función REQ5() realiza varias operaciones. Primero, extrae la lista de obras correspondiente al departamento ingresado por parámetro a partir del MapReq5, con orden de crecimiento constante. Más adelante, recorre esta lista (**x** ciclos) y le calcula el costo de transporte a cada obra con las funciones antes descritas. Finalmente, ordena dos veces la lista que contiene la información extraída según distintos criterios, con el fin de obtener las obras más costosas de transportar y las más antiguas. Estos ordenamientos sumarían entonces **xlog(x)** recorridos cada uno.

De esta manera, el algoritmo realiza **x + 2xlog(x)** recorridos en el peor caso. Esto se puede resumir en una complejidad de **O(xlogx)**

Comparación con Reto 1

Complejidad en el Reto 1: **O(nlogn)**

Se puede observar una reducción significativa de la complejidad, dado que, como se explicó anteriormente, **x** es de tamaño evidentemente menor que **n**.

Este cambio en la complejidad se debe a que, en el reto anterior, se realizaba un ordenamiento del catálogo de obras completo con el fin de encontrar las obras pertenecientes al departamento dado a partir de una búsqueda binaria. Para este reto, en cambio, ya se tiene esta información cargada en el catálogo, por lo que solo hace falta realizar el cálculo del costo de transporte de cada obra y extraer los TOP 5 solicitados.

Pruebas de tiempo

Se utilizó la siguiente entrada (departamento con más obras):

- Departamento: Drawings & Prints

Los resultados obtenidos se muestran a continuación:

		Tiempo [ms]	
Archivo	**# obras (n)**	**Daniel**	**Carlos**
small	768	31,25	15,62
5pct	7572	402,08	343,75
10pct	15008	850,00	703,12
20pct	29489	1790,63	1390,62
30pct	43704	2676,57	2100,67
50pct	71432	4618,75	3750,56
80pct	111781	7551,62	6112,25
large	138150	9525,00	7450,25

Tabla 5. Pruebas de rendimiento para el quinto requerimiento

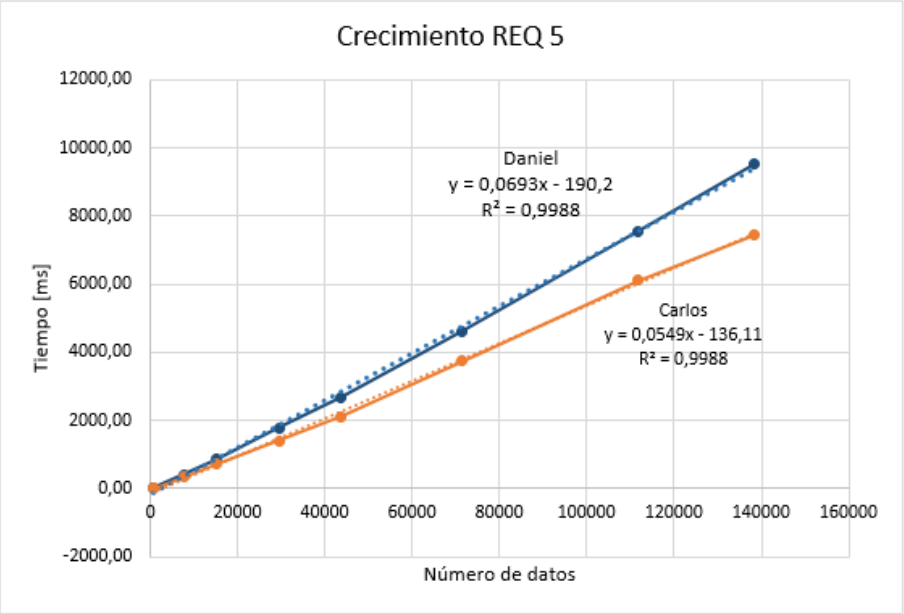

Figura 5. Gráfico del comportamiento del quinto requerimiento

Como era de esperarse, en las pruebas realizadas por ambos estudiantes se observó un orden de crecimiento compatible con el linearítmico propuesto en el análisis teórico, aunque se evaluó mediante regresiones lineales por el tipo de herramientas con las que se cuenta.