**OBSERVACIONES DEL RETO 2**

Juan Pablo Rodríguez Briceño Cod 202022764

Nicolas Pérez Terán Cod 202116903

**Ambientes de pruebas**

**Máquina 1** **Máquina 2**

|  |  |  |
| --- | --- | --- |
| **Procesadores** | Chip M1 | AMD Ryzen 5  3500U with Radeon  Vega Mobile Gfx  2.10 GHz |
| **Memoria RAM (GB)** | 8 GB | 12 GB (9,95  utilizables) |
| **Sistema Operativo** | MacOS BigSur | Windows 10 Home  64-bits |

**Maquina 1**

**Resultados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Porcentaje de la**  **muestra**  **[pct]** | **Req – 1** | **Req - 2** | **Req - 3** | **Req - 4** | **Req - 5** |
| Small (768) | ms | ms | ms | Ms | ms |
| 5%  (15008) | ms | ms | ms | Ms | ms |
| 10% | ms | ms | ms | Ms | ms |
| 20% | ms | ms | ms | Ms | ms |
| 30% | ms | ms | ms | ms | ms |
| 50% | ms | ms | ms | Ms | ms |
| 80% | ms | ms | ms | ms | ms |
| 100% | ms | ms | ms | ms | ms |

**Maquina 2**

**Resultados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Porcentaje de la**  **muestra**  **[pct]** | **Req – 1** | **Req - 2** | **Req - 3** | **Req - 4** | **Req - 5** |
| Small (768) | 15.625 ms | 1828.125 ms | 109.375 ms | 62.5 ms | 31.25 ms |
| 5%  (15008) | 93.75 ms | 2203.125 ms | 93.75 ms | 78.125 ms | 312.5 ms |
| 10% | 93.75 ms | 2609.375 ms | 93.75 ms | 140.625 ms | 625 ms |
| 20% | 140.625 ms | 2953.125 ms | 125 ms | 234.375 ms | 1296.875 ms |
| 30% | 171.875 ms | 3765.625 ms | 171.875 ms | 250 ms | 2125 ms |
| 50% | 203.125 ms | 4375.0 ms | 328.125 ms | 343.75 ms | 4109.375 ms |
| 80% | 234.375 ms | 5906.25 ms | 281.25 ms | 531.25 ms | 6046.875 ms |
| 100% | 343.75 ms | 6640.625 ms | 421.875 ms | 625 ms | 7593.75 ms |
|  |  | **Tabla de datos del Reto1.** | |  |  |
| **Porcentaje de la**  **muestra**  **[pct]** | **Req - 1** | **Req - 2** | **Req - 3** | **Req - 4** | **Req - 5** |
| Small (768) | 91.85699999 997 ms | 60.79100000 000004 ms | 22.35700000 000007 ms | 405.1800000  0000001 ms | 45.21120000  00000003 ms |
| 10.00%  (15008) | 222.4660000  00000072 ms | 571.2700000 000002 ms | 67.57099999  999999916 ms | 243805.655 | 489.1100000  0000001 ms |
| 20pct | 341.0570 | 956.53899 | 83.5310 | 587611.3 | 814.4880 |

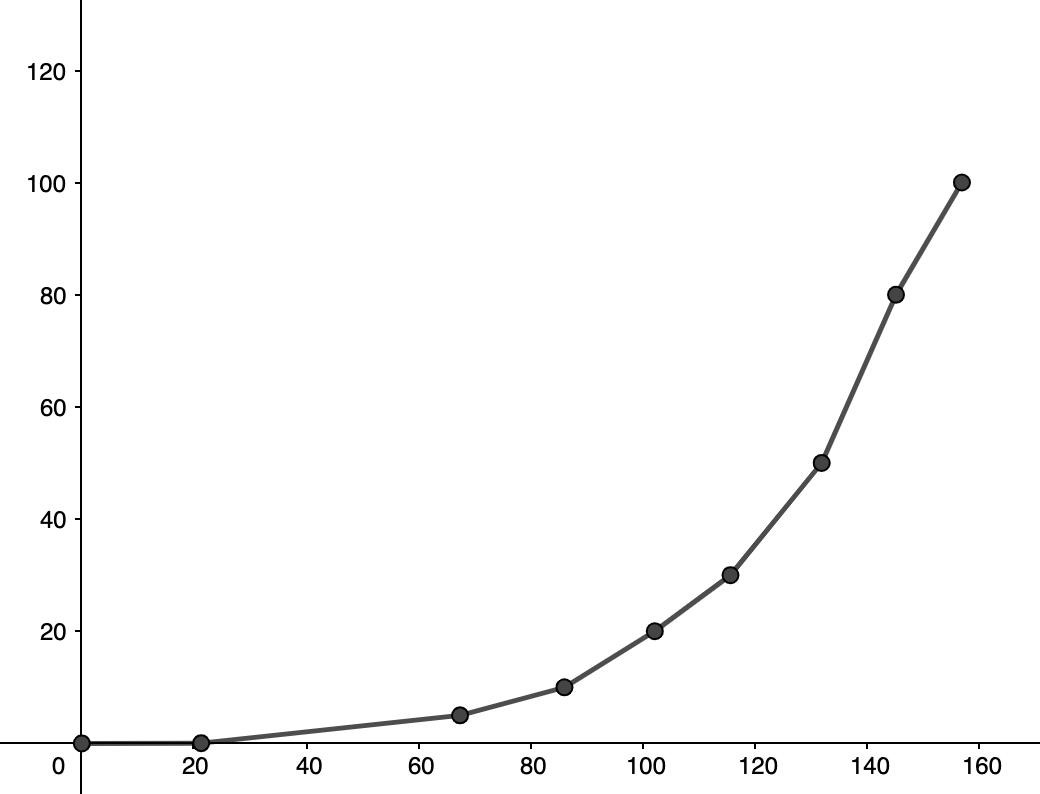
**Análisis de complejidad por cada requerimiento.**

**PD:** La extracción de nombres tiene una complejidad de O(1), porque al ser un diccionario basta con que se introduzca el id como llave.

**Req 1: Listar cronológicamente los artistas (Grupal)**. Se compone de:

* ArtistByDate: Que tiene una complejidad de O(1) por cada búsqueda que realiza, ya que, extrae los números de un map con unas llaves dadas. Al final, la complejidad vendría siendo la longitud del rango de fechas.

* GetSixArtworks: Tiene una complejidad de O(6), ya que busca 6 elementos de un ARRAY\_LIST.



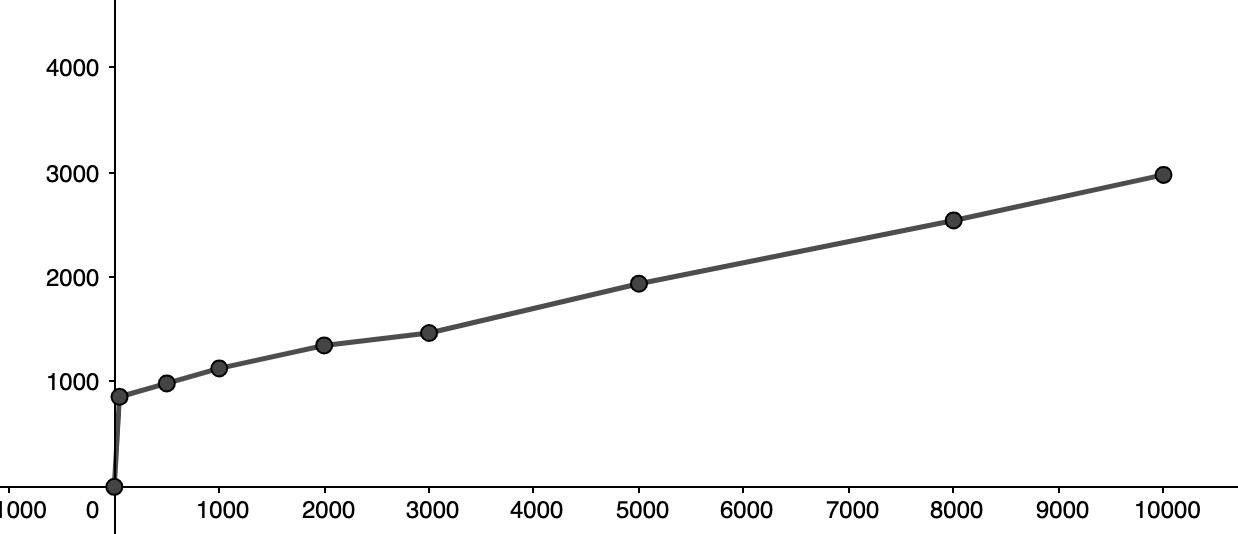
Si comparamos lo datos obtenidos en el reto 1 y 2.

|  |  |  |
| --- | --- | --- |
| **Pct** | **Reto 1** | **Reto 2** |
| Small | 91.85699999997 ms | 21.28 ms |
| 10pct | 222.466000000000072 ms | 86.0369 ms |
| 20pct | 241.0570 ms | 102.1549 ms |

Aquí directamente el segundo algoritmo es muchísimo más rápido y mantiene un incremento mucho menor al que maneja el primero. Es evidente que el crecimiento en promedio es de 20 en el segundo algoritmo y no aumenta demasiado, mientras que el primer algoritmo parece ser de mínimo 300. El algoritmo segundo resulta más efectivo sean pocos o muchos datos.

**Req 2: Listar cronológicamente las adquisiciones en un rango de fechas (Grupal)**. Se compone de:

* GetArtworksRange, que contiene:
  + Un algoritmo con complejidad O(n) que se encarga de buscar aquellas obras que estén en el rango.
  + Un algoritmo con complejidad de O(nLog(n)) que aplica MergeSort a los artworks extraídos y luego saca los primeros 10
* countUniqueArtists, el cual tiene una complejidad de O(n) que se encargara de contar los artistas solo una vez cada uno.
* getPurchasedArtworks, el cual tiene una complejidad de O(n) que se encarga de contar las obras que hayan sido compradas por el MoMA
* getSixArtworks, el cual tiene una complejidad de O(n) que se encarga de extraer las 3 primeras y 3 últimas obras en la lista del rango y generar un DataFrame



|  |  |  |
| --- | --- | --- |
| **Pct** | **Reto 1** | **Reto 2** |
| Small | 60.79100000000004 ms | 858.031 ms |
| 10pct | 571.2700000000002 ms | 1128.558 ms |
| 20pct | 956.53899 | 1347.742 ms |

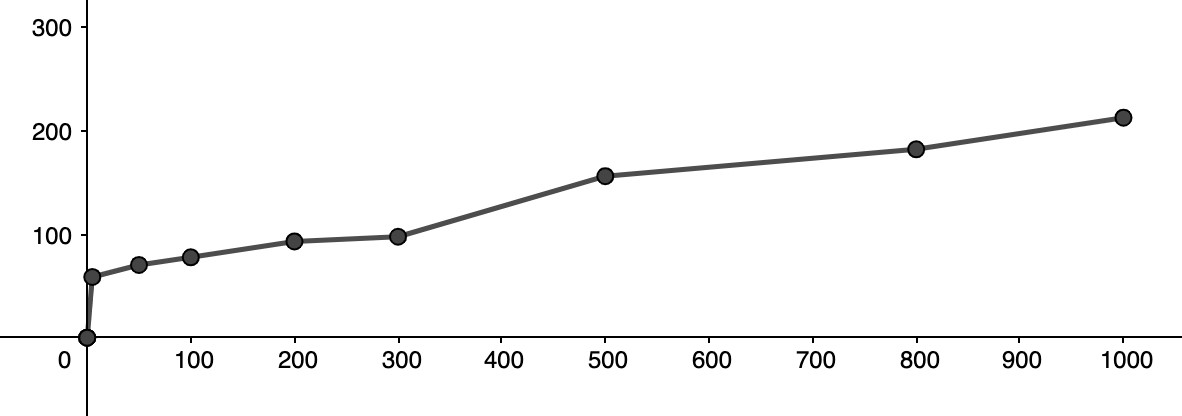
Podemos ver que el crecimiento es lo que ha cambiado, pues, aunque uno puede pensar que el primer algoritmo es más rápido, podemos decir que aumenta demasiado entre porcentajes. Mientras, que él según algoritmo incrementa, pero de manera constante y mucha menor medida. El segundo algoritmo será más rápido entre más datos sean.

**Req 3: Clasificar las obras de un artista por técnica (Individual - Nicolas Perez).** Se compone de:

* + ArtworksByArtist: Que, a su vez, se compone de dos partes. En la primera, llama a otra función llamada getArtWorksList, que crea un nuevo mapa donde las llaves serán los medios, y recorrerá toda la lista de obras para irlas ordenando según medio; su complejidad es O(n), siendo n el máximo de la

lista.

Luego, esta función con el mapa (Con las obras de un artista organizadas por medios) obtenido previamente, sacara sus llaves y empezara a extraer cada uno de los valores que hay, los cuales son listas con las obras según cada medio. Y en una nueva lista ira guardando parejas de (lista, tamaño), donde la lista serán las obras y el tamaño será la cantidad de elementos por medio. Esto tendrá una complejidad de O(n), porque dependerá de las obras que tenga un artista, lo cual podrían ser todas las obras en el peor de los casos.

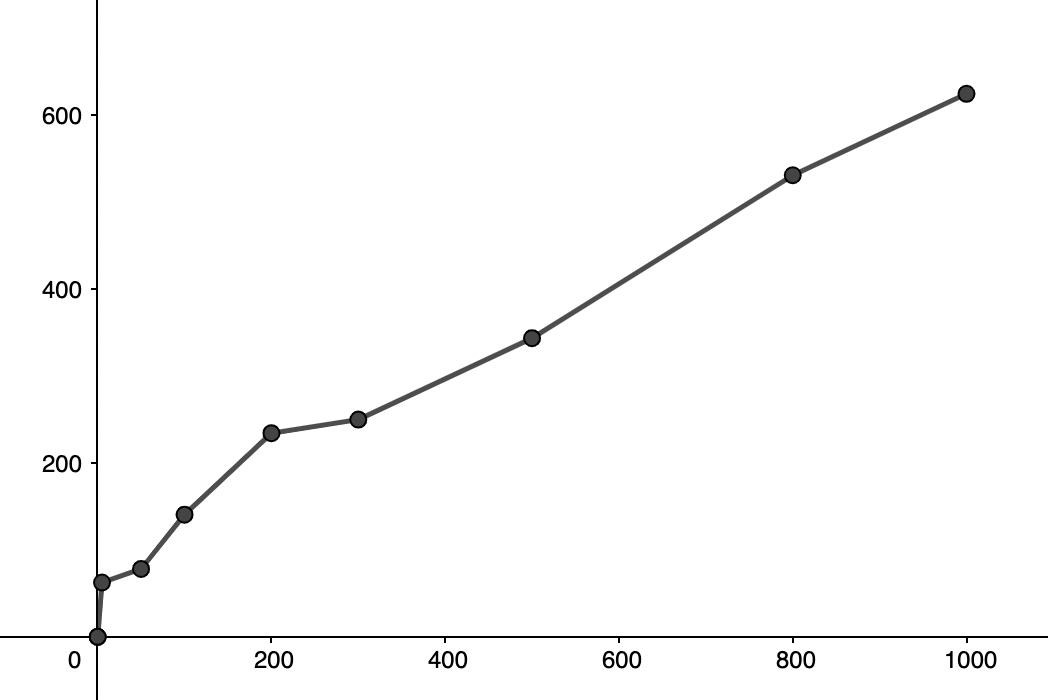


|  |  |  |
| --- | --- | --- |
| **Pct** | **Reto 1** | **Reto 2** |
| Small | 22.35700000000007 ms | 58.608 ms |
| 10pct | 67.57099999999999916 ms | 77.573 ms |
| 20pct | 83.5310 | 92.894 ms |

Como hemos visto en el primer requerimiento, el primer algoritmo se muestra más efectivo cuando se trata de muchos datos. Sin embargo, podemos evidenciar que cada vez la diferencia de tiempos es menos, debido a lo dicho anteriormente, el crecimiento del segundo algoritmo es mucho menor al que maneja el primero. Además, como se ve en la gráfica, el segundo algoritmo mantiene un ritmo constante, mientras que el primero va casi que aumentando el tiempo en (1/3).

**Req 4: Clasificar las obras por la nacionalidad de sus creadores (Individual - Juan Rodríguez).** Se compone de:

* top10lst, el cual se divide en tres partes:
  + Un algoritmo de complejidad O(n) que buscará las nacionalidades y las pondrá en una lista, la cual contendrá el número de obras
  + Un algoritmo de complejidad O(n(Log(n)) que organizara las nacionalidades según el número de obras que contentan la nacionalidad
  + Un algoritmo de complejidad O(n) que va a extraer las 10 primeras nacionalidades, las cuales se mostraran posteriormente al usuario
* top10DataFrame, el cual tiene una complejidad de o(n) que convertirá los datos de top10lst en un DataFrame
* getTopNationality, el cual tendrá una complejidad de o(n) que se encargara de sacar la lista con las obras, el número de obras y confirmara si los autores de la obra no tienen una nacionalidad que no sea la del top 1
* getSicArtWorks, el cual se encargará de extraer las 3 primeras y las 3 últimas obras en la lista de la nacionalidad y las convertirá en un DataFrame

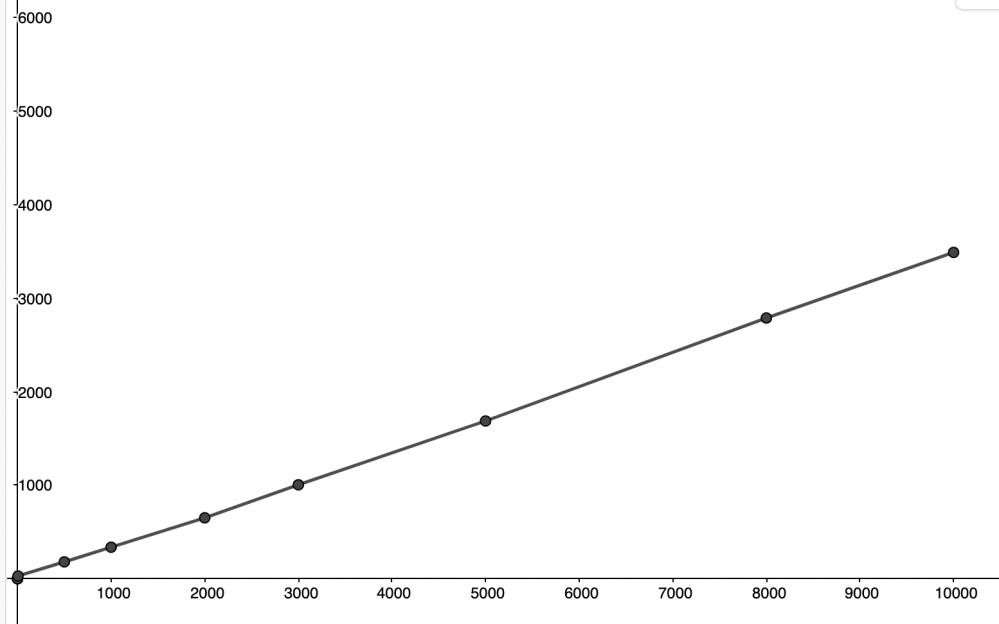


|  |  |  |
| --- | --- | --- |
| **Pct** | **Reto 1** | **Reto 2** |
| Small | 609.375 ms | 62.5 ms |
| 10pct | 413250 ms | 78.125 ms |
| 20pct | Tarda mucho más... | 140.625 ms |

Desde las pruebas con small, podemos notar que hay un cambio considerable en los tiempos de este requerimiento y, a medida que se empieza a trabajar con más datos, la diferencia se hace aún más grande tanto como si se compara entre las distintas versiones de los retos como los resultados de los mismos retos, pero con los otros archivos csv. También, el algoritmo del reto 2 no presenta diferencias tan drásticas cuando se cambia la cantidad de datos y suele mantener un ritmo de crecimiento constante a medida que se trabajan con más datos.

**Req 5: Transportar obras de un departamento (Grupal)**. Se compone de:

* GetArtworksByDep: El cual tiene una complejidad de O(1) por tipo de obra. Ya que busca en un mapa que tiene el departamento como índices. Así que obtendrá fácilmente todas las obras según un departamento.
* GetCost: Que tiene una complejidad de O(n), ya que debe recorrer todas las obras del departamento, recordemos que tiene que sacar cuentas de cada uno.



|  |  |  |
| --- | --- | --- |
| **Pct** | **Reto 1** | **Reto 2** |
| Small | 45.2112000000000003 ms | 29.586 ms |
| 10pct | 489.11000000000001 ms | 337.338 ms |
| 20pct | 814.4880 | 652.2709 ms |

Si bien los tiempos del reto 2 son menores a los del reto 1, puede que no tengan mucha diferencia cuando se tratan de pocos datos, pero esta diferencia si se va haciendo mas notoria a medida que el numero de datos aumenta. Por el otro lado, el crecimiento que poseen ambos algoritmos es de carácter lineal y parece mantenerse así sin importar el numero de datos con los que se vayan a trabajar.