Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos Y algoritmos
ISIS1225 - 202119

Documento de Análisis

Ana Sofía Castellanos Mosquera
202114167
a.castellanosm@uniandes.edu.co

Martín Santiago Galván Castro
201911013
ms.galvan@uniandes.edu.co

Análisis General – Guardar datos:

Para la realización del reto 2, se guardó la información del archivo de Excel en un catálogo de la siguiente manera: se guardó las categorías de los videos en una lista tipo Array List, en donde cada elemento es un diccionario con el id de la categoría y su nombre.

Adicionalmente, se crearon 3 Maps. Uno ordenado por las categorías, otro por países y uno tercero ordenado a partir de los títulos de los videos cuya percepción es sumamente positiva realizado para cumplir el requerimiento 3. Para las primeras 2, se tienen parejas llave-valor en donde la llave es el id de una categoría para el caso del índice de categorías, o el nombre de un país para el índice de países. El valor de dichas parejas es un diccionario, para el caso de categorías de la forma{category: "Categoría", videos: (Lista de videos que pertenecen a dicha categoría)} y en el caso de países de la forma {country: "País", videos: (Lista de videos que pertenecen a dicho país)}.

Para el map por títulos de videos se tienen parejas llave-valor donde la llave es el nombre del video y el valor de dicha llave es un diccionario de la forma: {title: "Titulo video", videos: (Lista de videos cuyo título es title)}. De igual manera con el fin de almacenar los títulos que son llaves del map TitleMap se creó en el catálogo una lista con el nombre de 'titles', no se utiliza la función de KeySet(…) puesto que esta debe buscar nuevamente en el Map y esto aumentaría la complejidad del Requerimento 3.

Para guardar los Maps se utilizó el tipo de tabla de hash cuyo esquema de colisiones es Linear Probing con un factor de carga de 0.8. Hicimos uso de esta estructura de datos, ya que a partir de los análisis hechos en el laboratorio N7 se evidenció que este sistema de colisiones ocupa menor memoria que el Separate Chaining debido a que este solo almacena el map y no buckets adicionales para guardar la información. Así mismo Linear Probing con factor de carga de 0.8 fue el más eficiente en términos de tiempo.

Análisis de complejidades:

Requerimiento 1 (Equipo de trabajo):

El código desarrollado para lograr el primer requerimiento del reto se escribió teniendo en cuenta la siguiente estrategia:

1. Revisar que videos cumplen con los requisitos de búsqueda del primer requerimiento
2. Generar una lista con dichos videos
3. Organizar la lista con algún algoritmo de ordenamiento usando como criterio la cantidad de likes
4. Generar una sublista del tamaño del número de videos a listar
5. Imprimir la anterior lista.

Para lograr los pasos anteriores, al igual que en el reto anterior se tuvo que solucionar el problema de como acceder a un video a partir de un nombre de categoría, sabiendo que los videos se guardan con el id de su categoría. Para esto, se utilizó la misma función que en el reto anterior que accede a la lista de categorías y toma el id del nombre. Con esto, se pasó a realizar el primer paso de la estrategia.

Al realizar el algoritmo que realiza el paso 1 y 2, es decir, que mete en una lista los videos que cumplen con las condiciones, primero se accedió al índice de videos por país, y se usa como llave el país que se pone por entrada en la función. El valor de esta pareja es una lista de videos de dicho país. Se recorre una vez esta lista en donde se buscan los videos que son de la categoría que se busca. Los videos que cumplen los requisitos se ponen en una lista.

Luego, dado a que esta lista se tiene que organizar con algún algoritmo, se usó el algoritmo MergeSort para realizar dicha tarea. Esto dado a que no hay muchos límites sobre el uso de la memoria en el programa, y es de las menores complejidades para algoritmos de ordenamiento. Este algoritmo es de complejidad Big O O(n log(n)).

Dado a que se puede escribir la complejidad del requerimiento 1 como:

$$O(1) + O(n) + O(n \log(n))$$

En la anterior expresión, la complejidad O(1) expresa la complejidad que toma acceder al índice de país, O(n) expresa la complejidad que toma recorrer una vez la lista de videos de un país buscando los videos de una categoría. Y O(n log(n)) la complejidad del algoritmo MergeSort. Se pasa a describir la complejidad total del algoritmo la cual es:

$$O(n \log(n))$$

Dado a que la función linearitmica tiende a volverse más grande que la función lineal y Big O se delimita con el peor de los casos.

En la creación del anterior algoritmo, se podía ingresar al índice por país o al índice por categoría. Se optó por ingresar al índice por país dado a que la lista de videos por país se pueden encontrar menos videos que en la lista de videos de una categoría. Esto con el objetivo para que en los pasos subsiguientes de acceder al índice se recorran menos videos.

Requerimiento 2 (Martín Santiago Galván Castro):

En el caso del requerimiento 2, primero, al igual que en el requerimiento 1, se creó una lista solo con aquellos videos que cumplen con los requisitos. En este caso, similar al requerimiento anterior, se inicia accediendo al índice de videos por país. Posteriormente, se revisa cuales videos tienen una relación de likes /dislikes mayor a 10. Después de esto, se organiza haciendo uso del algoritmo MergeSort. Se organizaron por el id de video para poder realizar un algoritmo que cuenta el máximo de días de los videos con la menor complejidad posible.

Después de organizar por sus id de video, se procede a iterar sobre la lista de videos que cumplen los requisitos. Esta iteración se hace con el objetivo de calcular la moda de videos. Dado a que están organizados por id, un mismo video que aparece varias veces este agrupado otras fechas en donde el video aparece. Se inicia el conteo cuando se encuentra un nuevo video, si se encuentran más videos en las siguientes iteraciones, se aumenta el contador. Cuando se encuentra un video diferente al anterior, se reinicia la cuenta. Al completar el ciclo, se devuelve el video que más veces apareció en trending.

Se puede representar la complejidad del algoritmo comprendiendo la complejidad de cada parte del algoritmo. Primero, la creación de la lista con los videos que cumplen el requerimiento tiene una complejidad de O(1) + O(n). Luego, el algoritmo de MergeSort tiene complejidad de O(n log(n)) y el algoritmo que cuenta el máximo tiene complejidad de O(n). Se puede entender como:

$$O(1) + O(n) + O(n\log(n)) + O(n)$$

Por lo anterior, dado que la máxima complejidad es la del ordenamiento MergeSort, la complejidad del algoritmo para resolver el requerimiento 2 es de:

$$O(N \log N)$$

Requerimiento 3 (Ana Sofia Castellanos Mosquera):

El código desarrollado para lograr el tercer requerimiento del reto se escribió teniendo en cuenta la siguiente estrategia:

1. Se obtiene el id de la categoría a partir de su nombre
2. Se realiza un ciclo que itera sobre la lista de títulos presentes en el archivo csv y que son llaves en el map titleMap.
3. Se obtienen los videos que tienen ese título y se revisa si la categoría de dichos videos es la categoría ingresada por parámetro, de ser así se saca la cantidad de videos que existe con ese título y se revisa si el tamaño de esos valores de la llave supera el máximo, si lo hace se modifica el valor del máximo y el elemento que cuenta con esa mayor cantidad de días.
4. Se devuelve el máximo y el video que tiene dicho máximo

Considerando la estrategia utilizada al analizar el algoritmo en primer lugar, tiene una complejidad O(N) que resulta de encontrar en la lista de categorías el id que corresponda al

nombre de la categoría ingresada por parámetro. Tras ello el algoritmo cuenta con un Big O de O(N), ya que itera sobre la lista de los títulos de los videos presentes en el map titleMap para sacar el elemento con get(…) que tiene ese título, verifica si la categoría de los videos corresponde con el ingresado por parámetro y compara la cantidad de días de dicho video con el máximo para poder sacar el mayor elemento.

Considerando lo anterior, la ecuación que representa la complejidad del algoritmo es:

$$O(N) + O(N)$$

Debido a que la complejidad Big O considera solo el mayor nivel de complejidad. La ecuación Big O será al final el resultante del ciclo, es decir:

$$O(N)$$

Requerimiento 4 (Equipo de trabajo):

El código desarrollado para lograr el cuarto requerimiento del reto es similar al desarrollado en el requerimiento uno, sin embargo, este requerimiento ha de devolver videos todos diferentes en el top N y no ha de distinguir mayúsculas y minúsculas en el tag. Para ello este requerimiento se escribió teniendo en cuenta la siguiente estrategia:

1. Obtener del Map por países los videos cuyo país es el que ingresa por parámetro
2. Buscar dentro de estos videos encontrados en el Map aquellos cuyo tag (en minúscula) contenga el tag (en minúscula) ingresado por parámetro y agregar a una nueva lista
3. Ordenar esta nueva lista por comentarios, de manera que los videos que tengan mayor cantidad de comentarios queden en la parte superior
4. Crear una lista auxiliar que contendrá los nombres de los videos de la lista del paso 3
5. Realizar un ciclo para colocar una lista los videos ordenados sin que se repitan y hasta que se cumpla el top_n.

Considerando la estrategia utilizada al analizar el algoritmo en primer lugar, tiene una complejidad O(1) que resulta de buscar el país, ingresado por parámetro, en el Map de países. Tras ello el algoritmo cuenta con una complejidad de O(N), ya que se realiza un ciclo para encontrar los videos que cuenten con el tag ingresado por el usuario y guardarlos en una nueva lista.

A partir de esa lista se ordenan por la cantidad de comentarios con un algoritmo MergeSort, lo cual tiene una complejidad de O(N log N). Finalmente se realiza un nuevo ciclo sobre esa lista para revisar que no se repitan los nombres de los videos, lo cual tiene una complejidad de O(N).

A partir de lo anterior, la ecuación de complejidad queda como:

$$O(1) + O(N) + O(N\log(N)) + O(N)$$

Debido a que se toma el peor caso en Big O, la complejidad del algoritmo es de:

$$O(N \log N)$$

Comparación con el reto 1:

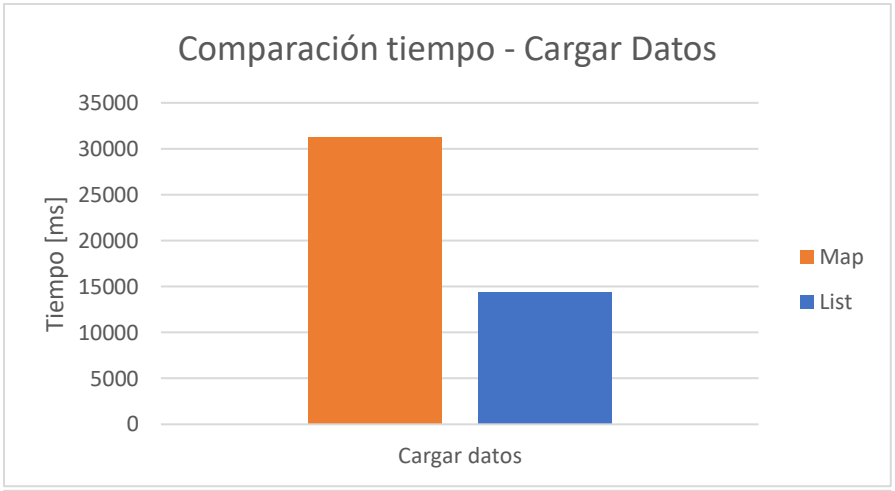
Como parte de estudio de las eficiencias entre el uso de listas y mapas, se busca comparar los tiempos y usos de memorias entre el reto 1 y el reto 2. Todas las siguientes pruebas se hicieron usando como parámetros de búsqueda los mismos mostrados en la guía del reto 1.

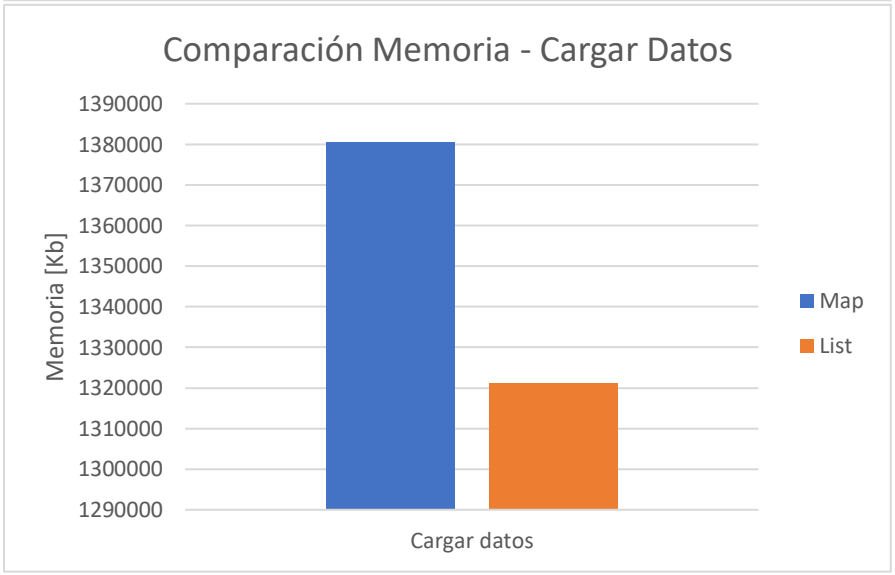
Cabe mencionar que, previo a la realización de estas pruebas, se realizaron cambios al código del reto 1. Más específicamente, a los requerimientos 2 y 3. Puesto que anteriormente, estos requerimientos no contaban con las complejidades óptimas. Ambos algoritmos compartían complejidad Big O de $O(n^2)$, por lo cual se modificó para que contara con complejidad Big O O(N log N). Así mismo, para el reto N2 se utilizó en el Req 2 un algoritmo de complejidad O(N log N) y para el Req 3 un algoritmo de complejidad O(N).

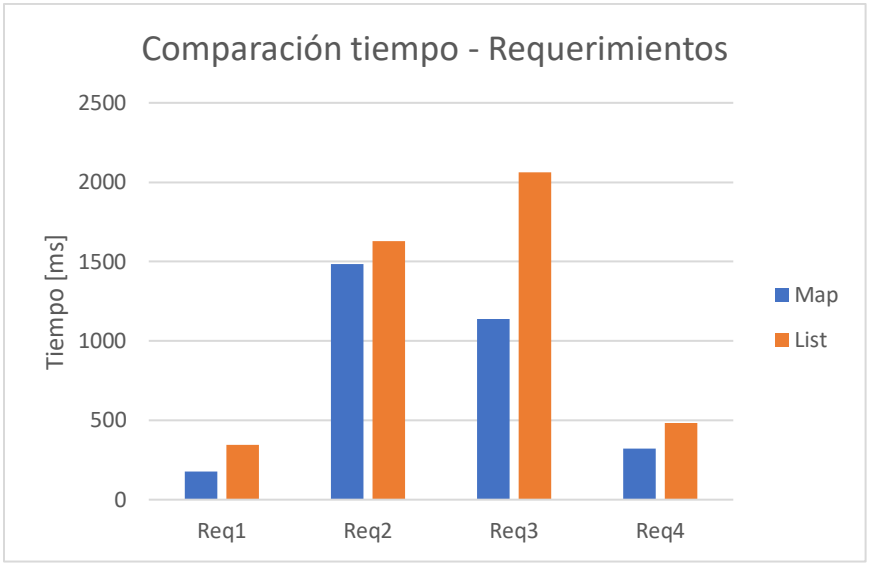
Comparaciones Maquina 1 (Ana Sofia Castellanos)

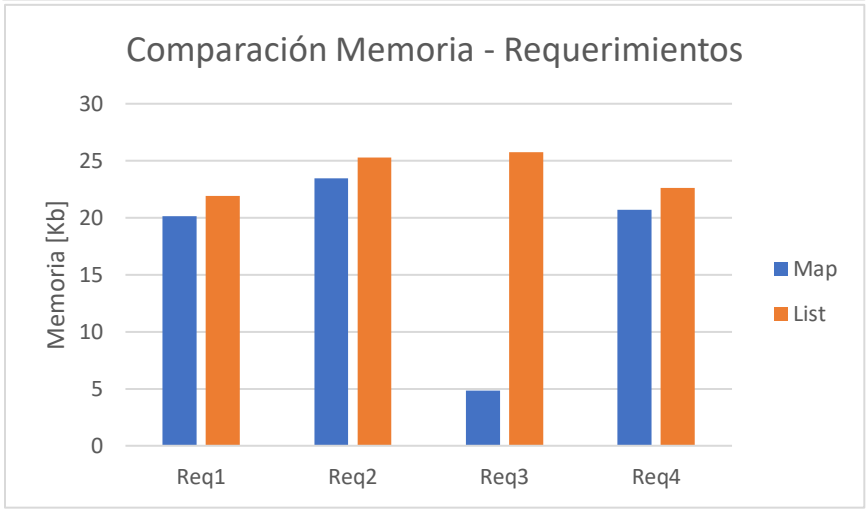
Datos Maps vs Listas:

	Map		List	
	Tiempo [ms]	Memoria [Kb]	Tiempo [ms]	Memoria [Kb]
Cargar datos	31182.6	1380694	14407.95	1321158
Req1	175.9676	20.128	345.893	21.909
Req2	1484.631	23.445	1628.071	25.28
Req3	1136.901	4.836	2061.428	25.74
Req4	321.5232	20.72	482.8094	22.62
Total	34301.62	1380763	18926.15	1321253


Comparación tiempo - Cargar Datos
35000
30000
25000
20000
15000
10000
5000
0
Tiempo [ms]
Cargar datos
Map
List


Comparación Memoria - Cargar Datos
1390000
1380000
1370000
1360000
1350000
1340000
1330000
1320000
1310000
1300000
1290000
Memoria [Kb]
Cargar datos
Map
List



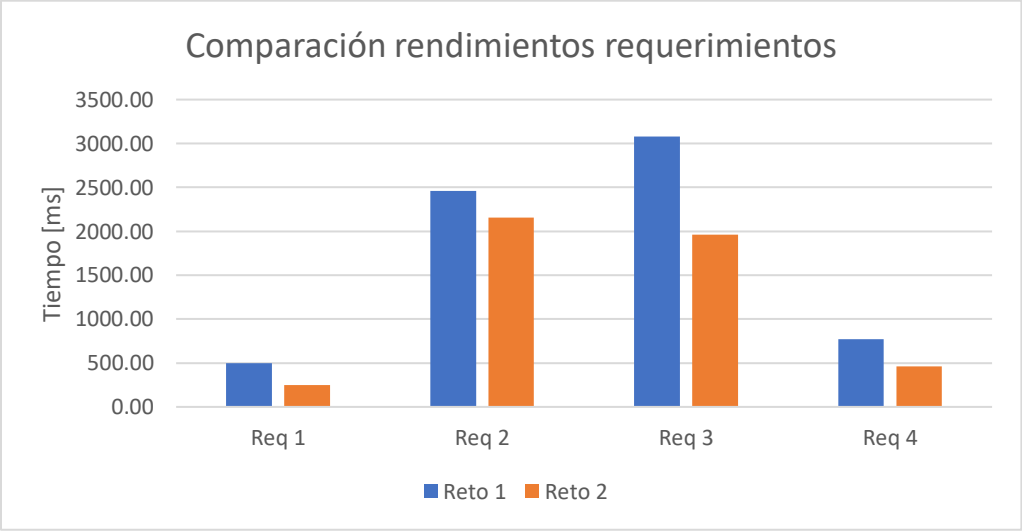


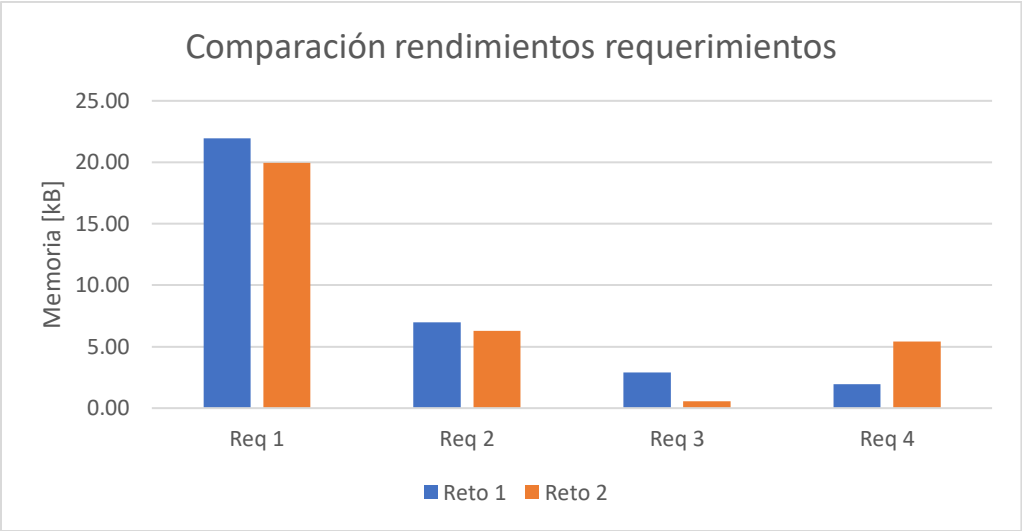
A partir de estas gráficas se puede observar que la carga de datos a nivel general en el Reto N2 es menos eficiente en términos de tiempo y memoria que en el Reto 1. Esto se debe a que la carga de datos del Reto 2 implementa una cantidad mayor de TADs tipo Map (por países, por categorías, por títulos de video) que el Reto 1 que solo implementa la lista de videos.

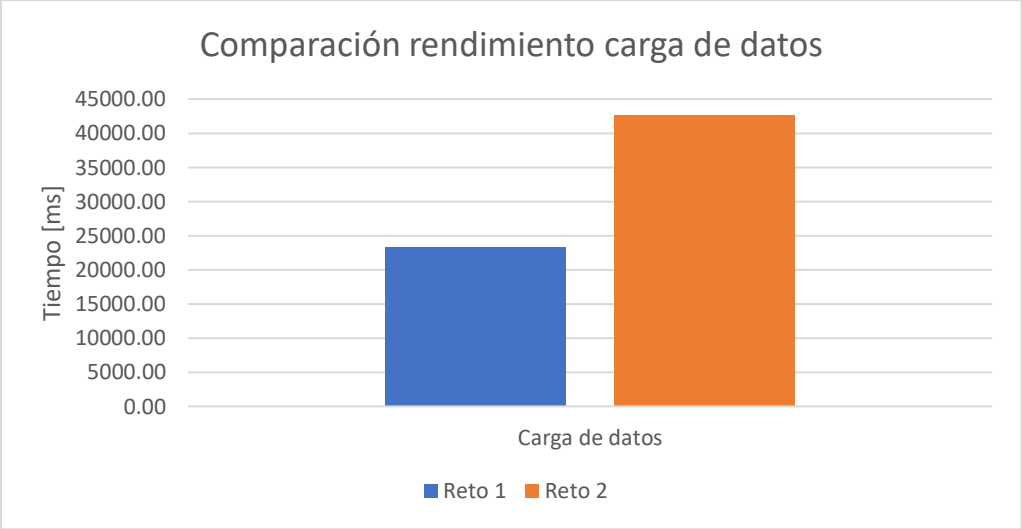
Ahora si bien la implementación con Maps es menos eficiente que en Listas cabe resaltar que al revisar los requerimientos se puede analizar que los implementados a través de Maps son más eficientes tanto en memoria y tiempo a comparación de los implementados con listas. Con lo cual es posible concluir que si bien al implementar Maps el tiempo de carga y la memoria utilizada es mayor que en las listas, el tiempo de respuesta y la memoria consumida es menor en los requerimientos.

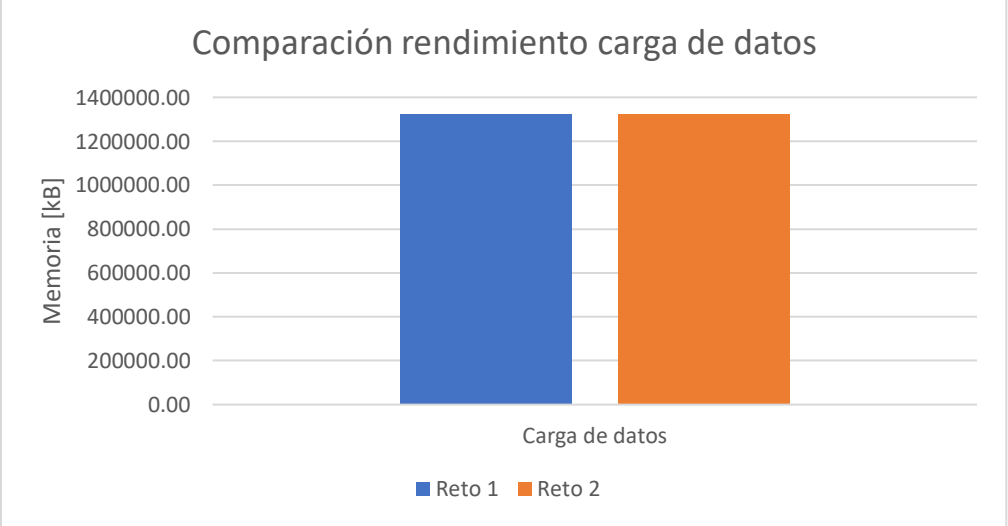
Comparaciones Maquina 2 (Martín Santiago Galván Castro)

	Reto 1		Reto 2	
	Tiempo [ms]	Memoria [kB]	Tiempo [ms]	Memoria [kB]
Carga de datos	23226,87	1321167,20	42646,71	1360213,67
Requerimiento 1	500,53	21,96	250,56	19,96
Requerimiento 2	2459,42	6,99	2153,09	6,30
Requerimiento 3	3078,46	2,91	1960,29	0,56
Requerimiento 4	772,16	1,94	464,35	5,43




Comparación rendimientos requerimientos
25.00
20.00
15.00
10.00
5.00
0.00
Memoria [kB]
Req 1
Req 2
Req 3
Req 4
Reto 1
Reto 2


Comparación rendimiento carga de datos
45000.00
40000.00
35000.00
30000.00
25000.00
20000.00
15000.00
10000.00
5000.00
0.00
Tiempo [ms]
Carga de datos
Reto 1
Reto 2


Comparación rendimiento carga de datos
1400000.00
1200000.00
1000000.00
800000.00
600000.00
400000.00
200000.00
0.00
Memoria [kB]
Carga de datos
Reto 1
Reto 2

Para la toma de datos de los resultados de la maquina 2, se realizaron 3 repeticiones y se promediaron esos resultados. Se evidencia que en general, los tiempos de ejecución para el código realizado en el reto 2 son más rápidos que para el reto 1. Adicionalmente, se puede evidenciar que se tiende a gastar igual o más memoria para llevar a cabo los requerimientos. Sin embargo, esta diferencia en el uso de la memoria depende más de cómo se implemente el requerimiento, puesto que el requerimiento 3, el cual no carga y organiza listas utiliza considerablemente menos memoria. Por parte de la carga de datos, se evidencia que el reto 2 requiere de más tiempo dado a la complejidad de como almacena datos. Mientras que, para el uso de memoria, es difícil ver mucha diferencia puesto que su uso de memoria es muy similar. Pero el reto 1 tiende a ocupar un poco más de memoria. Esta diferencia se da no por el código, si no más bien por los archivos de Excel que se leen.