Juan Sebastián Sánchez – 202121498 – js.sanchezl1@uniandes.edu.co - Req. 3

Alejandro Jaramillo Castellanos – 202111794 - a.jaramilloc2@uniandes.edu.co - Req. 2

Tablas de tiempos Reto 1:

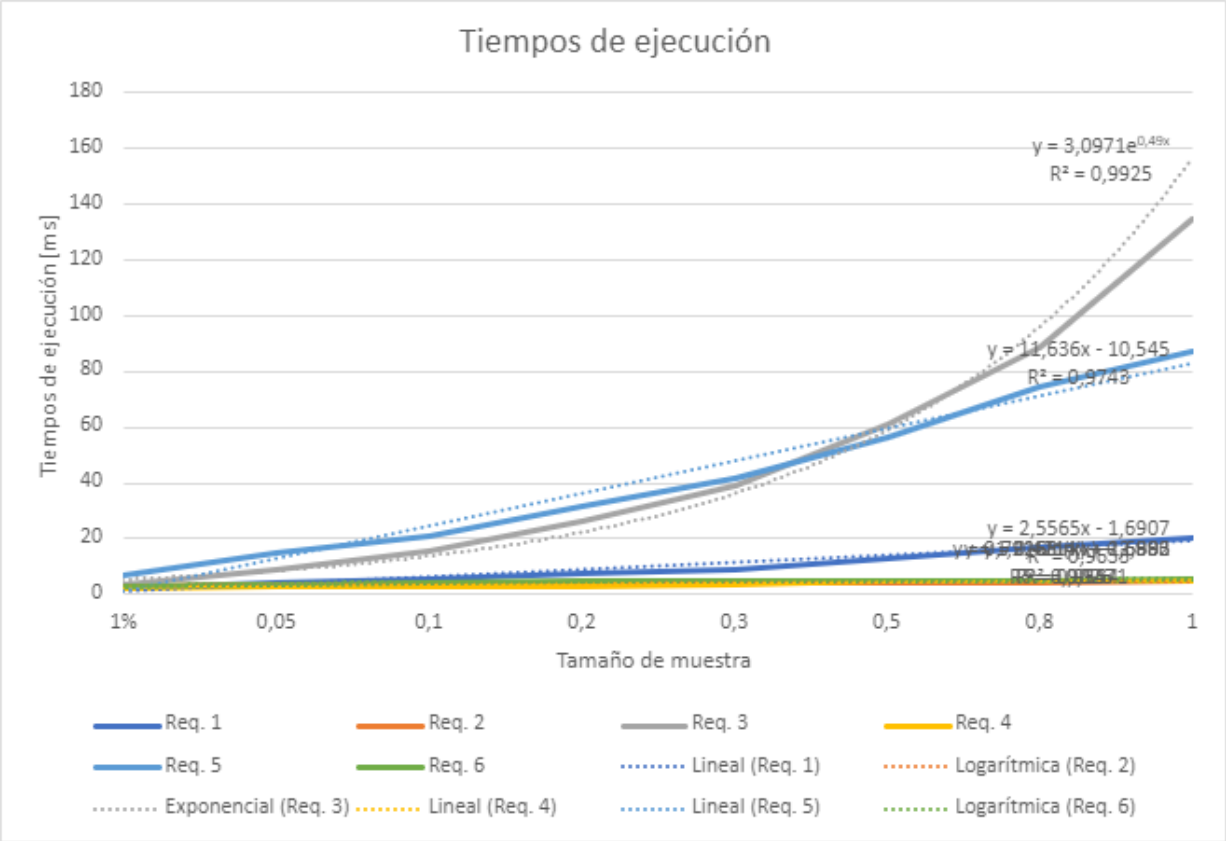
Análisis con ARRAY_LIST Y MERGE SORT

Cantidad Datos	Req. 1 Tiempo de Ejecución [ms]	Req. 2 Tiempo de Ejecución [ms]	Req. 3 Tiempo de Ejecución [ms]	Req. 4 Tiempo de Ejecución [ms]	Req. 5 Tiempo de Ejecución [ms]	Req. 6 Tiempo de Ejecución [ms]
0,50%	3650	0,0015	0,0014	6,2	0,59	5782
5%	9805	0,0018	0,0019	59,41	9,35	10167
10%	14950	0,0022	0,0021	115,09	19,96	
20%						
30%						
50%						
80%						
100%						

Tablas de tiempos Reto 2:

Análisis con PROBING, ARRAY_LIST Y MERGE SORT

Cantidad Datos	Req. 1 Tiempo de Ejecución [ms]	Req. 2 Tiempo de Ejecución [ms]	Req. 3 Tiempo de Ejecución [ms]	Req. 4 Tiempo de Ejecución [ms]	Req. 5 Tiempo de Ejecución [ms]	Req. 6 Tiempo de Ejecución [ms]
0,50%	2,13	2,58	3,63	209	7,05	210,22
5%	4	3,44	9,04	209,4	14,72	212,57
10%	5,74	3,7	15,76	92,4	21,11	214,44
20%	7,4	3,73	26,38	202,8	31,73	219,15
30%	9,11	3,91	39,12	205	41,6	211,87
50%	12,9	4	60,82	737,67	56,37	751,85
80%	17,07	4	88,37	211,5	74,52	208,9
100%	20,16	4,64	135	218	87,45	216,83



Memoria:

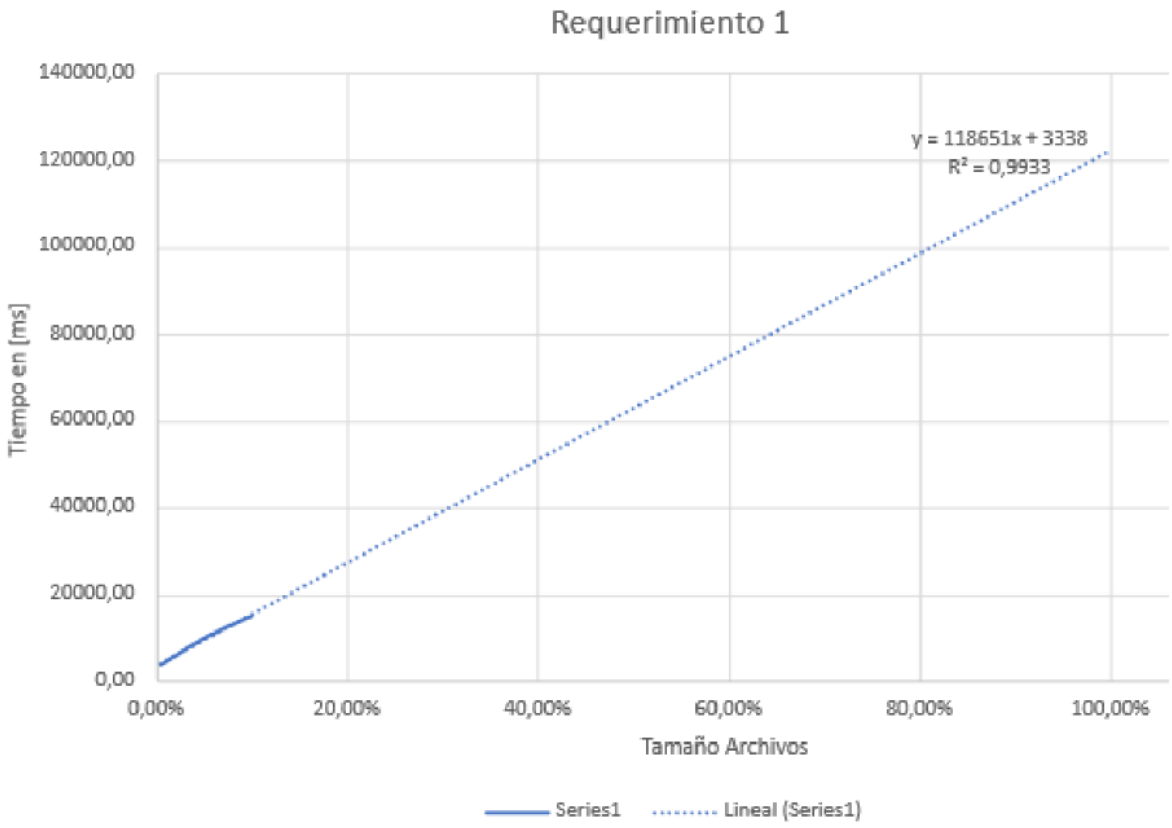
Análisis con PROBING, ARRAY_LIST Y MERGE SORT

Cantidad Datos	Req. 1 Memoria [kB]	Req. 2 Memoria [kB]	Req. 3 Memoria [kB]	Req. 4 Memoria [kB]	Req. 5 Memoria [kB]	Req. 6 Memoria [kB]	Carga Memoria [kB]
0,50%	7,6	1,2	2,08	6,5	3,57	0,52	236831,3
5%	8,2	1,35	2,61	6,96	3,91	2,5	337790,9
10%	8,6	2,875	3.26	8,81	4,51	4,25	438509
20%	9,2	3,11	3,8	9,68	4,68	4,85	618249
30%	9,1	3,74	3,97	10,11	6	5,02	780750,1
50%	9,96	4,72	4,21	10,95	6,41	6,21	1063019,4
80%	10,2	4,91	5,6	11,54	6,7	6,3	1450092,3
100%	10,93	5	5,7	11,83	7,26	7,38	1649636,6

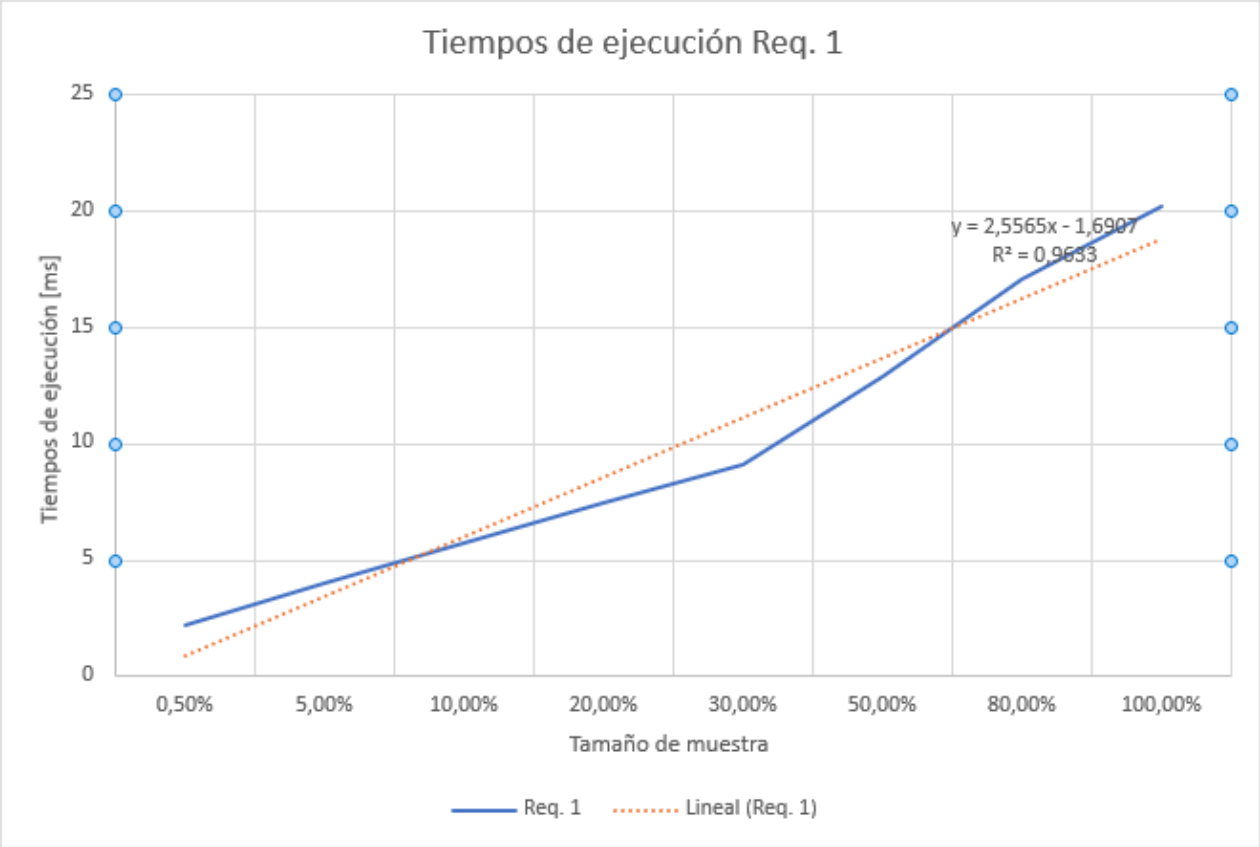
Requerimiento 1:

Pruebas de ejecución:

(RETO 1)



(RETO 2)



Análisis:

En la gráfica se obtiene una línea con pendiente, en otras palabras, complejidad O(N). En el código se usa mp.get() para obtener el álbum que esté en la llave del año, esto tiene complejidad O (1), luego se recorre cada álbum cambiando el id del artista por el nombre, lo cual tiene una complejidad de O(N), ya que se recorre toda la lista de álbumes de ese año, y luego se hace un merge sort por nombre, el cual tiene complejidad O(log[N]), por ende, la complejidad de este requerimiento es O(N).

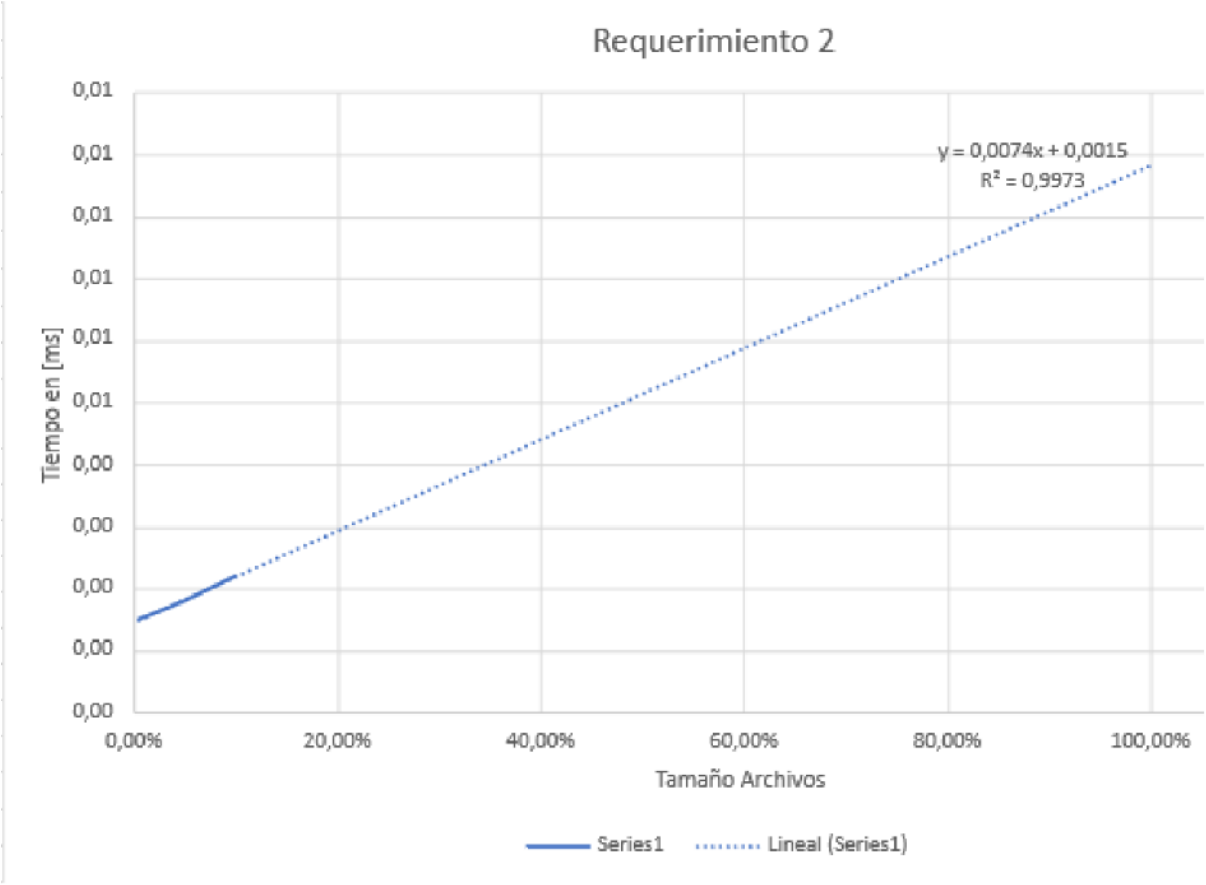
Reto 1 vs Reto 2:

En el reto 1 y 2 las gráficas de tiempos de ejecución nos dieron ambas lineales (O(N)), no obstante, los tiempos mejoraron notoriamente. En el reto 1, el archivo large duraba en cargar aproximadamente 120,000 ms, mientras que en el reto 2, el requerimiento cargaba en 20ms. Se puede ver claramente como las búsquedas se vuelven mucho más rápidas usando tablas de hash en vez de listas.

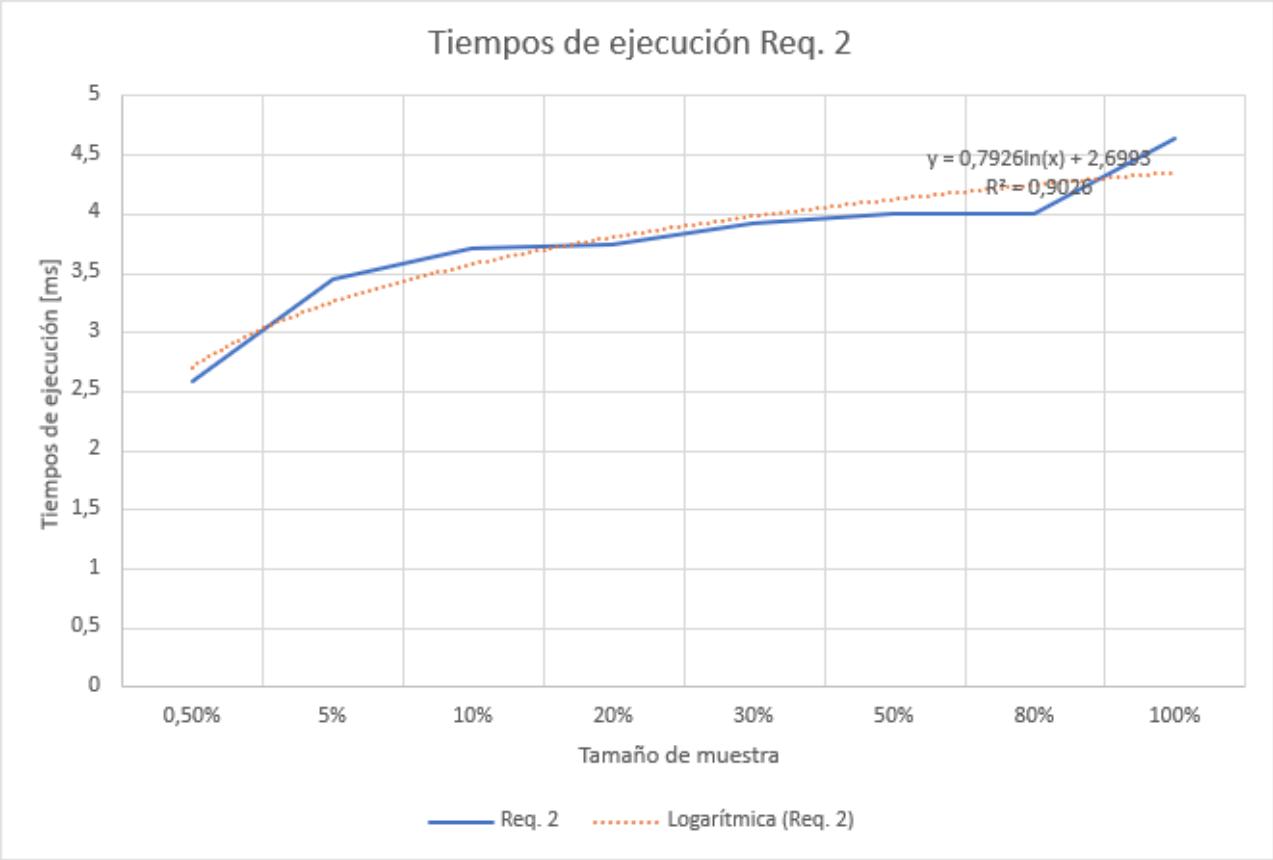
Requerimiento 2(Alejandro):

Pruebas de ejecución:

(RETO 1)



(RETO 2)



Análisis:

En la gráfica se obtiene un logaritmo natural, en otras palabras, complejidad O(log[N]). En el código se usa mp.get() para obtener el artista que esté en la llave de popularidad, esto tiene complejidad O (1), luego se recorre cada artista cambiando el id de la canción por el nombre, lo cual tiene una complejidad de O(C), ya que cambia solo las canciones de, los artistas en esa popularidad, por lo que son menos elementos, y luego se hace un merge sort por followers y nombre, el cual tiene complejidad O(log[N]), por ende, la complejidad de este requerimiento es O(log[N]).

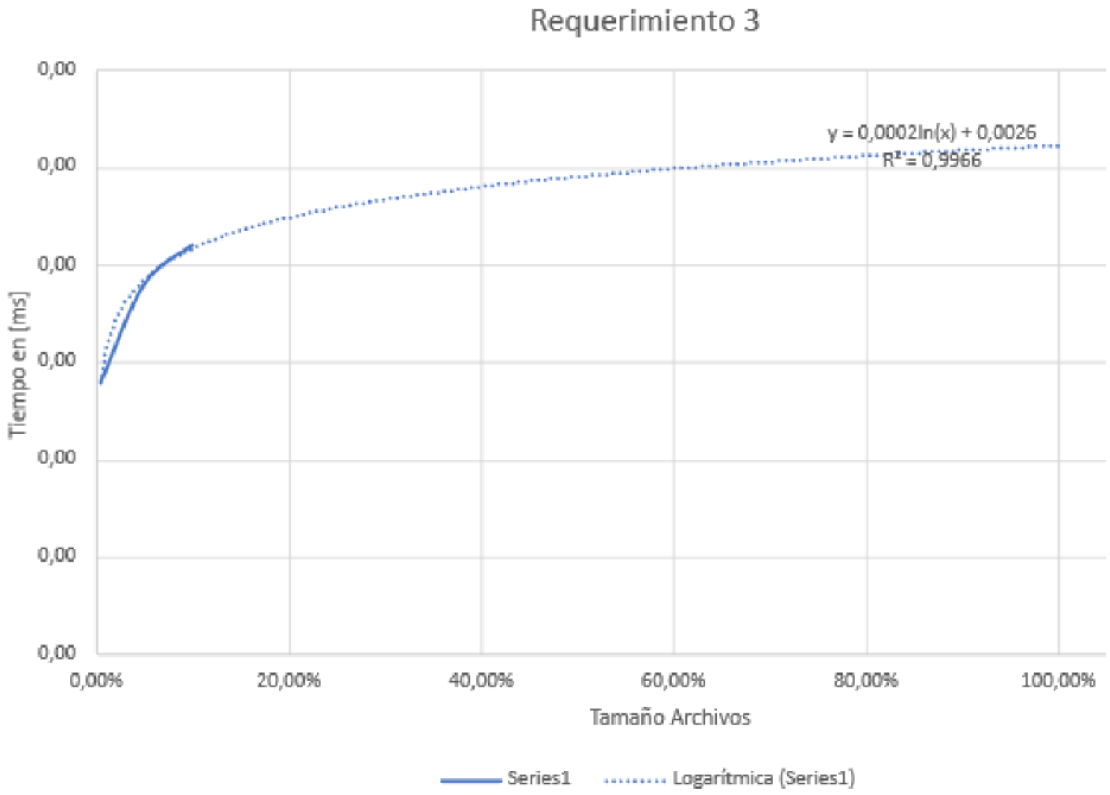
Reto 1 vs Reto 2:

En el primer reto tuvimos problemas para hacer la gráfica de este requerimiento ya que solo nos cargaba en el archivo small, 5% y 10%. La gráfica tenía una pequeña tendencia de ser lineal (O(N)) con una carga de menos de 1ms. Por otro lado, la gráfica del tiempo de ejecución del reto 2 nos da logarítmica (O(log(N)) y en el archivo más grande (large), el requerimiento se resuelve en menos de 5ms. Consideramos que no se puede hacer un análisis preciso entre ambos retos ya que necesitamos más información de los tiempos de carga del primer reto.

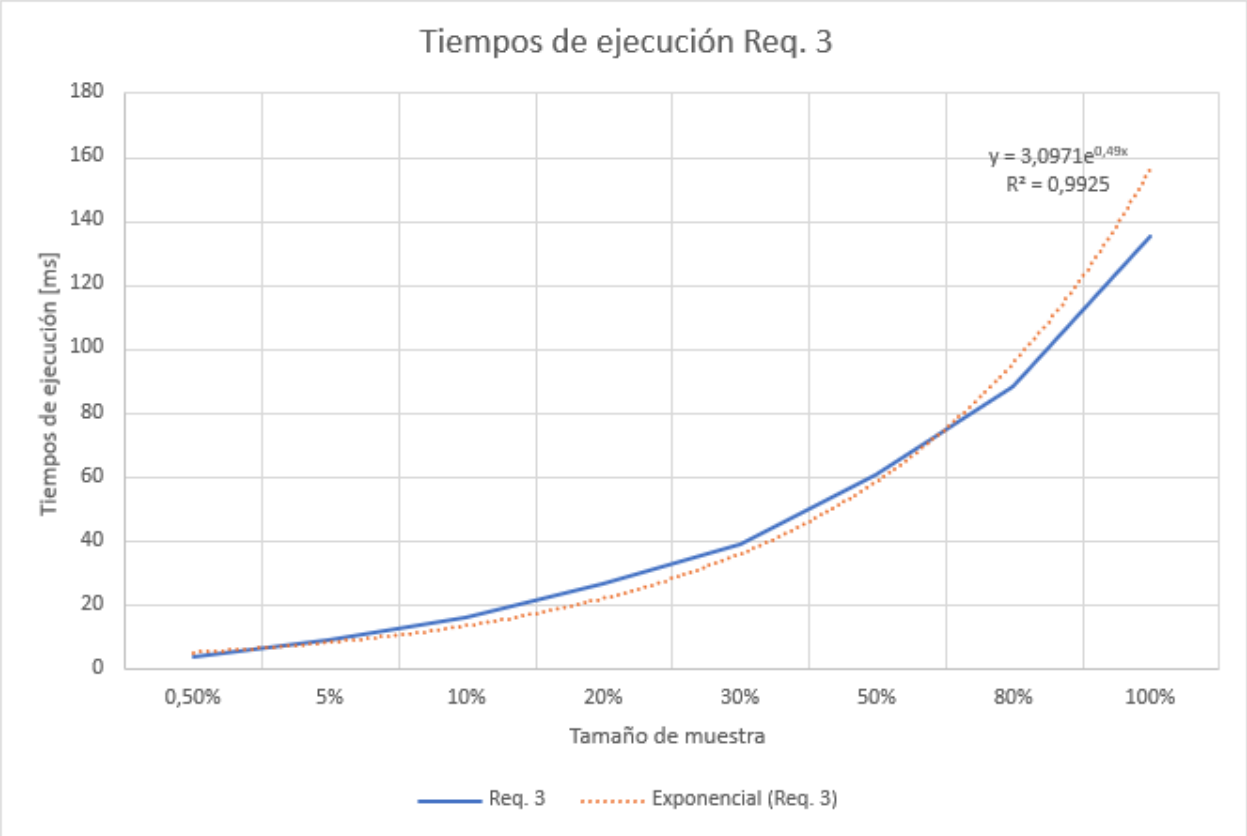
Requerimiento 3(Juan Sebastián):

Pruebas de ejecución:

(RETO 1)



(RETO 2)



Análisis:

En la gráfica se obtiene una línea con pendiente, en otras palabras, complejidad O(N). En el código se usa mp.get() para obtener la canción que esté en la llave de popularidad, esto tiene complejidad O (1), luego se recorre cada canción, luego recorre cada id del artista y lo cambia por su nombre, lo cual tiene una complejidad de O(N*n), ya que se recorre toda la lista de canciones con esa popularidad y luego se recorre cada artista de cada canción, pero máximo hay 10 artistas en una canción, por ende no es N^2, y luego se hace un merge sort por duración y nombre, el cual tiene complejidad O(log[N]), por ende, la complejidad de este requerimiento es O(N).

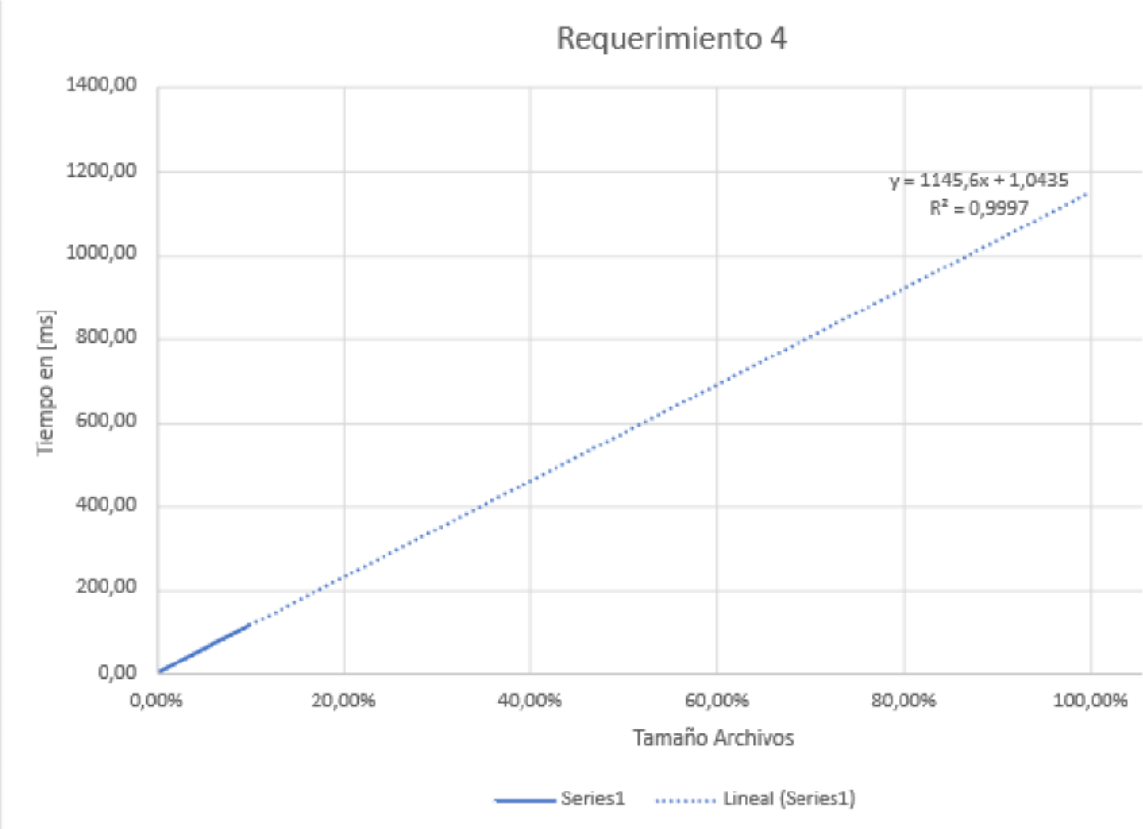
Reto 1 vs Reto 2:

En el primer reto tenemos un inconveniente similar al del requerimiento 2; solo nos cargaba en los archivos small, 5% y 10%. En este requerimiento, la gráfica de los tiempos de ejecución tiende a ser logarítmica (O(log(N)) y el tiempo de ejecución también es de menos de 1ms. Por otra parte, en el reto 2, a pesar de que se responde al requerimiento en el archivo más grande en tan solo 140ms, la gráfica tiene carácter exponencial entre 1 y 1.6, el cual sigue siendo relativamente bajo.

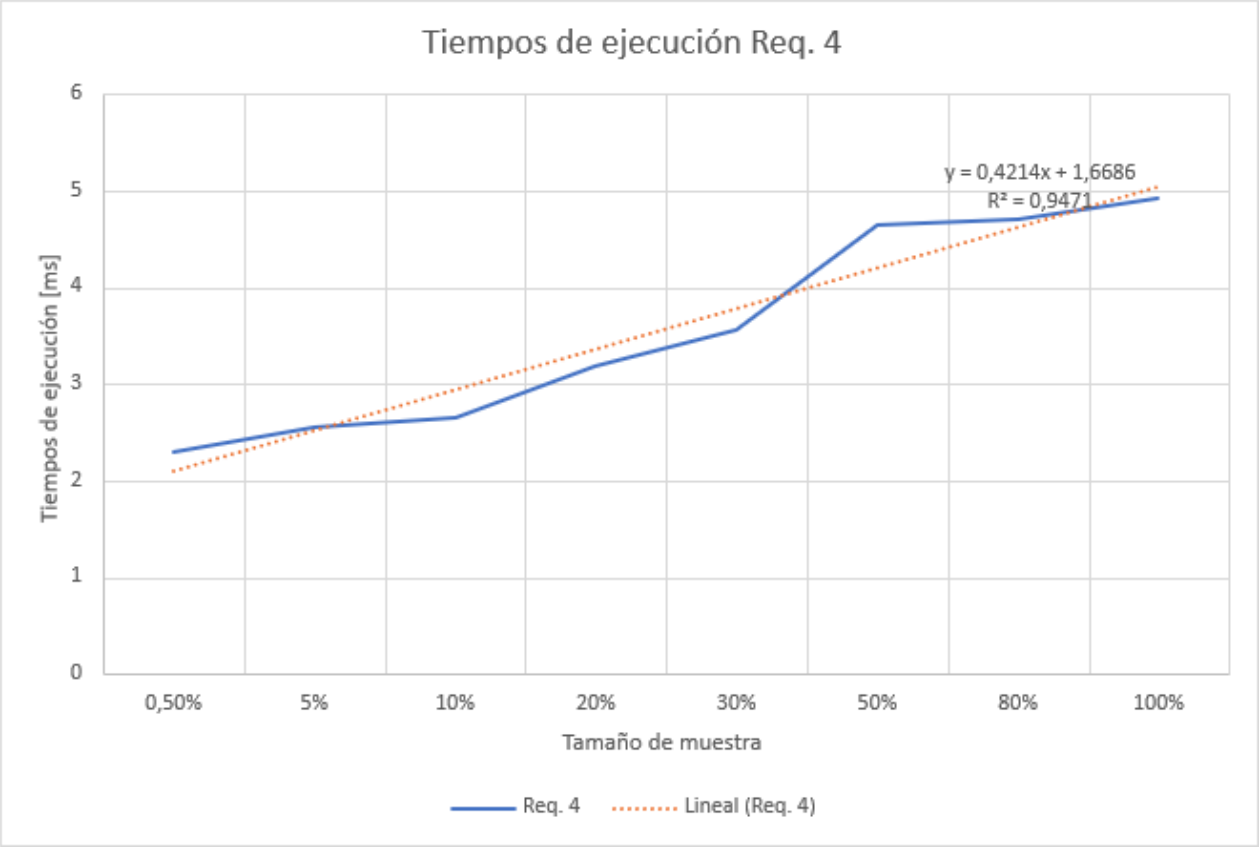
Requerimiento 4:

Pruebas de ejecución:

(RETO 1)



(RETO 2)



Análisis:

En la gráfica se obtiene una línea con pendiente, en otras palabras, complejidad O(N). En el código se usa mp.get() para obtener las canciones que estén en la llave del artista, esto tiene complejidad O (1), luego se recorre cada canción verificando que contenga el país ingresado en su mercado y también cambia cada id de artista por nombre, lo cual tiene una complejidad de O(N), ya que hace dos recorridos a parte con la misma complejidad, ya que se recorre toda la lista de canciones de ese artista, y luego se hace un merge sort por popularidad, el cual tiene complejidad O(log[N]), por ende, la complejidad de este requerimiento es O(N).

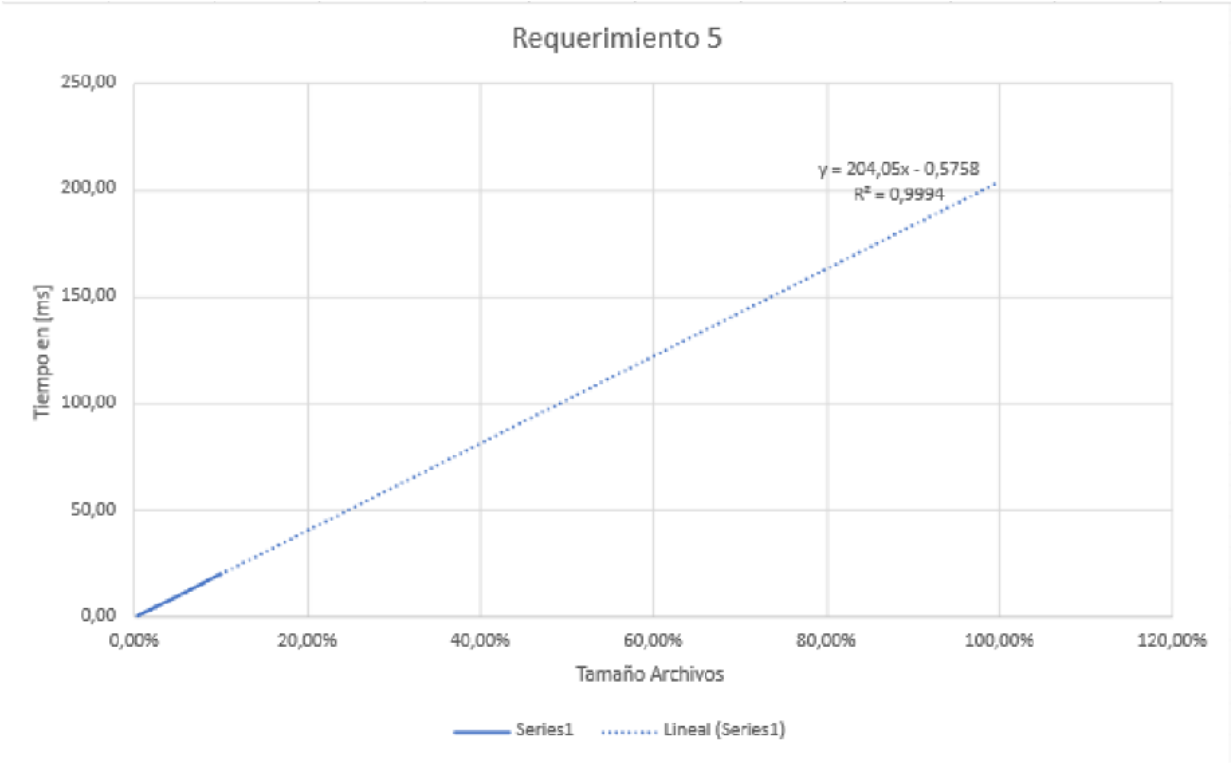
Reto 1 vs Reto 2:

En el reto 1 esta gráfica nos dio lineal (O(N)). En el archivo 10%, el requerimiento cargó en un poco más de 100 ms. La gráfica del reto 2 también nos dio lineal (O(N)) pero su tiempo de ejecución en el archivo large fue considerablemente menor, 5ms, teniendo en cuenta que este era un archivo con muchos más datos. La implementación que se usó en el segundo reto hace mucho más rápidas las búsquedas, por lo tanto, el tiempo de ejecución de este requerimiento es mucho menor.

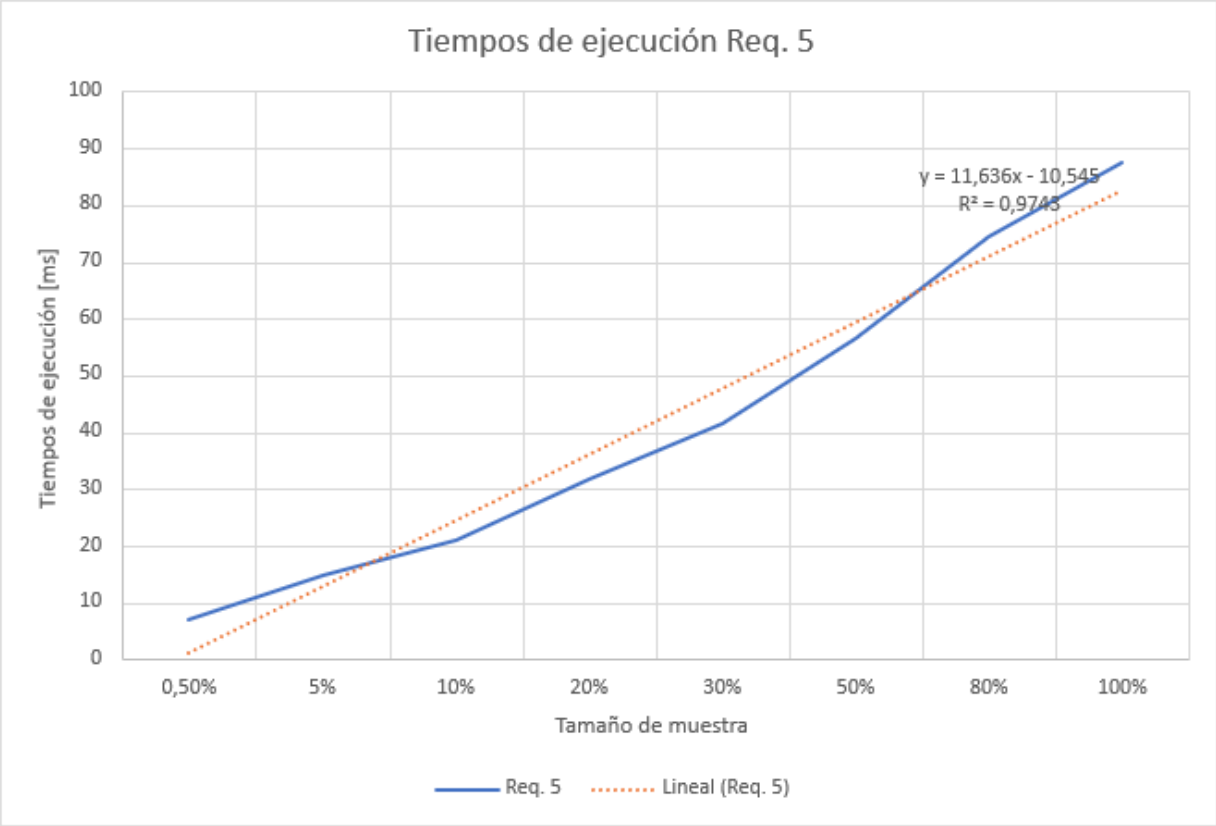
Requerimiento 5:

Pruebas de ejecución:

(RETO 1)



(RETO 2)



Análisis:

En la gráfica se obtiene una línea con pendiente, en otras palabras, complejidad O(N). En el código se usa mp.get() para obtener los álbumes que estén en la llave del artista, esto tiene complejidad O (1), y luego se hace un merge sort por fecha, el cual tiene complejidad O(log[N]). Luego se recorre cada álbum y se extrae su id [O(c)], luego se busca con mp.get() [O (1)], las canciones que tengan de llave el id del álbum. Luego se recorre cada canción, luego recorre cada id del artista y lo cambia por su nombre, lo cual tiene una complejidad de O(N*n), ya que se recorre toda la lista de canciones con esa popularidad y luego se recorre cada artista de cada canción, pero máximo hay 10 artistas en una canción, por ende, no es N^2, y luego se hace un merge sort por duración y nombre, el cual tiene complejidad O(log[N]), por ende, la complejidad de este requerimiento es O(N).

Reto 1 vs Reto 2:

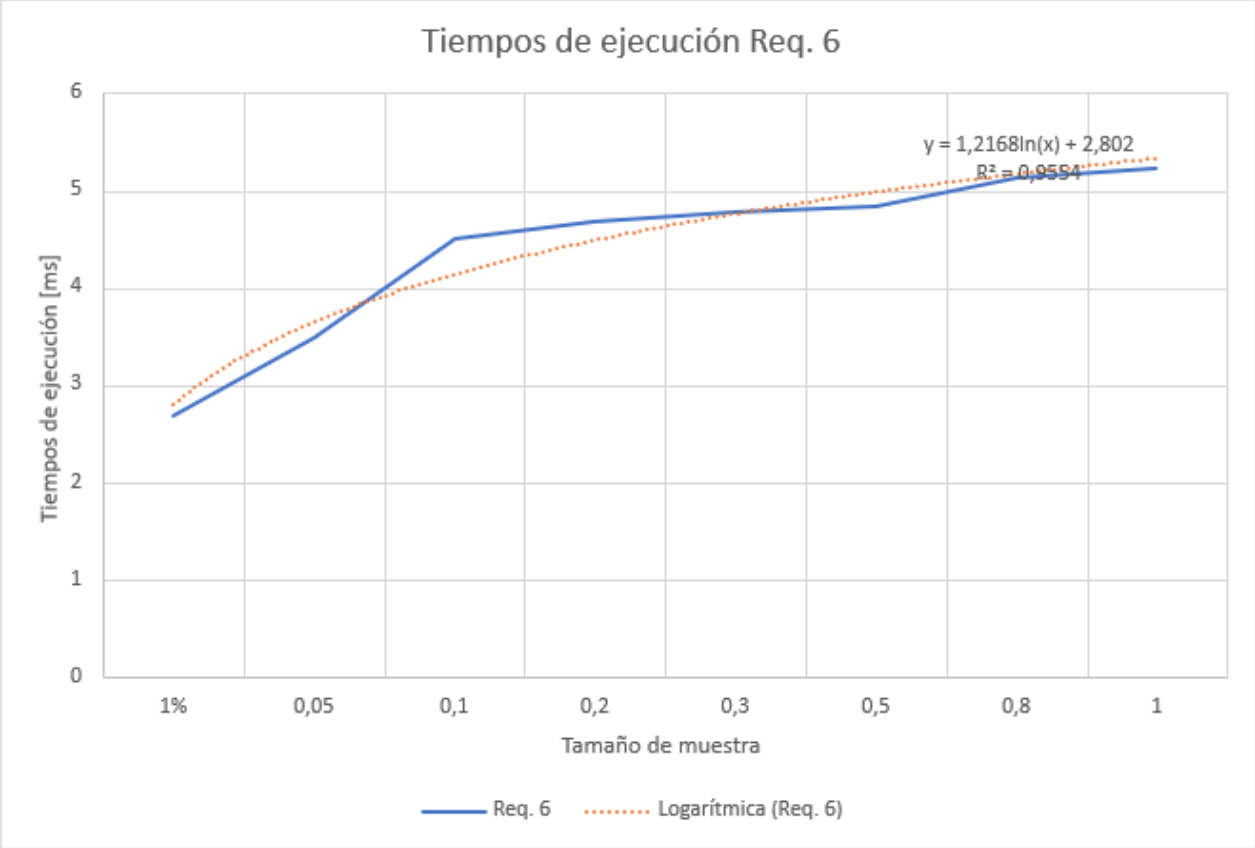
En ambos retos las gráficas del requerimiento 5 nos dieron lineales (O(N)). Sin embargo, para los archivos 10% el tiempo de carga en el reto 1 fue de más o menos 25ms y en el reto 2 de 20ms. La diferencia de carga en estos archivos es mínima, aunque si el comportamiento del tiempo de ejecución en el reto 1 fuera el que se predice con la gráfica, este requerimiento se respondería en 200ms en el archivo large, mientras que en el reto 2 se respondería en menos de 90ms en el mismo tamaño de archivo. Con una mayor cantidad de datos se

evidencia claramente la mejora en la eficiencia de la implementación que se usó en el segundo reto con respecto al primer reto.

Requerimiento 6 (Bono):

Pruebas de ejecución:

(RETO 2)



Análisis:

En la gráfica se obtiene un logaritmo natural, en otras palabras, complejidad O(log[N]). En el código se usa mp.get() para obtener las canciones que estén en la llave del id del artista, esto tiene complejidad O (1), luego se recorre cada canción cambiando el id del álbum por el nombre, lo cual tiene una complejidad de O(c), ya que cambia solo los albums de, las canciones para ese artista, por lo que son menos elementos, y luego se hace un merge sort por followers y nombre, el cual tiene complejidad O(log[N]), por ende, la complejidad de este requerimiento es O(log[N]).