Milton Andrés Mesa 202022496 ma.mesam1@uniandes.edu.co

Configuración óptima del catálogo: ARRAY_LIST

Algoritmo de ordenamiento: Mergesort, pues estamos trabajando con arreglos grandes

Algoritmos de búsqueda: Búsqueda binaria para arreglos que se puedan ordenar. Búsqueda lineal para arreglos que es mejor no ordenar.

I. ANÁLISIS DE COMPLEJIDAD:

Req.	Complejidad	Explicación. Ignoramos todas las operaciones de tiempo constante (asignación, comparación, lt.addLast(array), lt.getSize, lt.getElement(array), bucles constantes) Y resaltamos las operaciones que tengan tiempos de ejecución variables (bucles variables, lt.iterator(array), busqueda_binaria, for i in range(string), MS.sort, lt.sublist). Las complejidades menores se marcan con amarillo, las mayores se marcan con azul.
Carga	O(n^2 logn)	loadTracks: for track in input_file->O(n) loadArtists: for artists in input_file->O(n) loadAlbums: for album in input_file->O(n) agregar_datos_relacionados: for album in lt.iterator(catalog['albums']): ->O(n) crear_artist name: busqueda_binaria->O(logn) for track in lt.iterator(catalog['tracks']):->O(n) crear_artist: convertir_str_a_lista: for i in range(len(s))->O(n) for i in lt.iterator(track['artists_id']): ->O(n) busqueda_binaria->O(logn) convertir_lista_a_str: for I in lt.iterator(L)->O(n) crear_album: busqueda_binaria->O(logn) crear_territorio: convertir_str_a_lista: for i in range(len(s))->O(n) for artist in lt.iterator(catalog['artists']): crear_best_track: busqueda_binaria->O(logn)
1 Alb. Años	O(nlogn)	sort_list_by(catalog['albums'], cmpAlbumsByYear): MS.sort(lst, criterio) -> O(nlogn) mas_alto = busqueda_binaria ->O(logn) mas_bajo = busqueda_binaria ->O(logn) albums_en_rango=lt.subList(catalog['albums'],mas_bajo,mas_alto-mas_bajo)->O(n)

2 Top artistas	O(nlogn)	sort_list_by(catalog[‘artists’], cmpArtistsByPopularity): MS.sort(lst, criterio) -> O(nlogn) top_n_artistas = lt.subList(catalog['artists'], 1, top_n) -> O(n)
3 Top tracks	O(nlogn)	sort_list_by(catalog['tracks'], cmpTracksByPopularity): MS.sort(lst, criterio) -> O(nlogn) top_n_tracks = lt.subList(catalog['tracks'], 1, top_n) -> O(n)
4 Mejor track	O(n^2)	for track in lt.iterator(catalog['tracks']): -> O(n) for nombre in lt.iterator(track['artists_names']): -> O(n) for pais in lt.iterator(track['available_markets']): -> O(n) for album in lt.iterator(catalog['albums']): -> O(n) sort_list_by(canciones_en_territorio, cmpTracksByPopularity): MS.sort(lst, criterio) -> O(nlogn) *En este caso, el requerimiento tiene una complejidad para el peor caso, mayor que otros requerimientos. Sin embargo, los for secundarios iteran sobre las listas track[‘artists_names’] y track[‘available_markets’] que generalmente son números pequeños. Y la lista sobre la que hace merge_sort (canciones_en_territorio) también es pequeña relativamente. Por esta razón, cuando medimos el tiempo de ejecución real, este requerimiento dura bastante menos que los demás. Pero para otro tipo de archivos puede que dure más.*
5 Discografía	O(n^2)	sort_list_by(catalog['tracks'], cmpTracksByPopularity): MS.sort(lst, criterio) -> O(nlogn) for album in lt.iterator(catalog['albums']): -> O(n) for track in lt.iterator(catalog['tracks']): -> O(n) *En este caso el requerimiento sí se comporta para el peor caso, debido a que itera sobre las listas catalog[‘albums’] y catalog[‘tracks’], que son las más grandes.*
6 Distribucion	O(nlogn)	sort_list_by(catalog['tracks'], cmpAlbumsByYear) MS.sort(lst, criterio) -> O(nlogn) mas_alto = busqueda_binaria ->O(logn) mas_bajo = busqueda_binaria ->O(logn) tracks_en_rango = lt.subList(catalog['tracks'], mas_bajo, (mas_alto-mas_bajo)) ->O(n) sort_list_by(tracks_en_rango, cmpTracksByDistribution): MS.sort(lst, criterio) -> O(nlogn) top_n_tracks = lt.subList(tracks_en_rango, 1, top_n)->O(n)

II. PRUEBAS DE TIEMPOS DE EJECUCIÓN:

5% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	3781.028	3839.428	4053.6666	3891.374
1 Alb. años	208.3351	210.1433	211.9068	210.1284
2 Top artistas	425.9702	455.5572	437.6718	439.7331
3 Top tracks	434.0518	396.0069	389.6237	406.5608
4 Mejor track	127.0294	132.1572	127.1104	128.7657
5 Discografía	389.8099	394.5781	332.9906	372.4595
6 Distribucion	234.775	257.4474	238.8146	243.679

10% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	7122.352	7842.112	7750.102	7571.522
1 Alb. años	437.3076	435.2397	416.3372	429.6282
2 Top artistas	722.7905	727.5335	748.1125	732.8122
3 Top tracks	796.9649	793.0265	771.8218	787.2711
4 Mejor track	240.2395	243.2047	242.2799	241.908
5 Discografía	708.8109	695.1268	676.8271	693.5883
6 Distribucion	508.7082	513.6358	506.2755	509.5398

20% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	11963.61	10869.49	11391.14	11408.08
1 Alb. años	887.7608	838.0129	813.5981	846.4573
2 Top artistas	1355.275	1327.105	1280.469	1320.95
3 Top tracks	1699.414	1658.522	1848.89	1735.609
4 Mejor track	454.5332	454.3994	448.8625	452.5984
5 Discografía	1579.347	1515.766	1482.874	1525.996
6 Distribucion	1015.853	1094.315	1042.125	1050.765

30% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	15719.45	14724.19	15305.59	15249.75
1 Alb. años	1323.866	1154.25	1164.015	1214.043
2 Top artistas	1740.37	1612.452	1605.96	1652.928
3 Top tracks	2426.256	2425.069	2357.991	2403.105
4 Mejor track	658.5095	653.6386	639.6559	650.6013
5 Discografía	2224.718	2256.192	2267.472	2249.46
6 Distribucion	1616.373	1589.195	1524.389	1576.652

50% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	21638.05	20766.71	21295.25	21233.34
1 Alb. años	1898.5	1749.312	1887.302	1845.038
2 Top artistas	2177.894	2148.505	2137.033	2154.477
3 Top tracks	4022.58	3967.382	3847.605	3945.856
4 Mejor track	1002.659	1011.123	1018.296	1010.693
5 Discografía	3673.326	3650.317	3937.549	3753.731
6 Distribucion	2584.552	2698.637	2609.363	2630.851

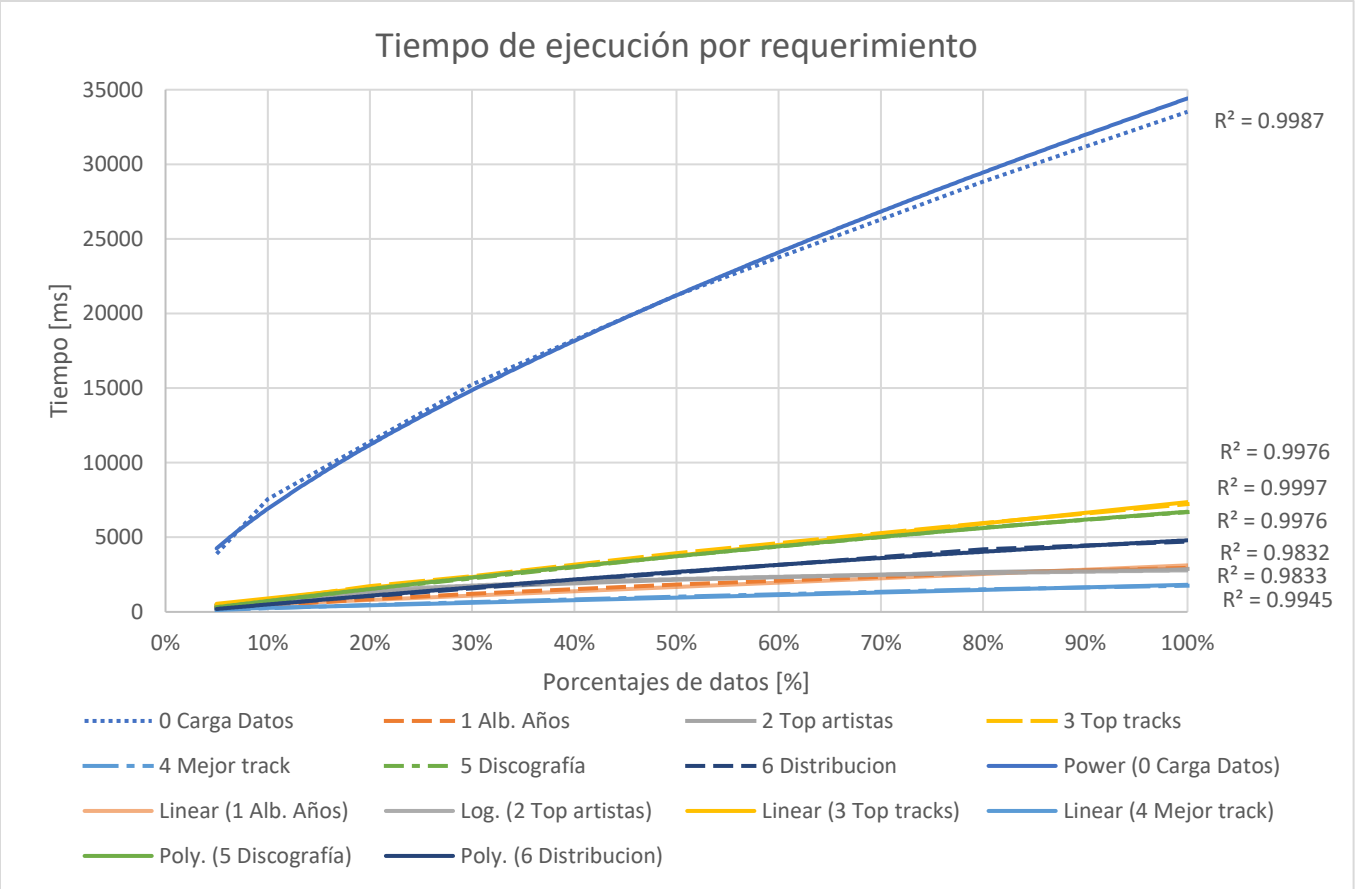
80% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	28816.82	28855.05	28835.85	28835.91
1 Alb. años	2586.331	2598.747	2627.986	2604.355
2 Top artistas	2652.976	2708.113	2676.347	2679.145
3 Top tracks	5931.62	5922.319	5995.408	5949.782
4 Mejor track	1521.569	1494.892	1499.509	1505.323
5 Discografía	5787.851	5588.98	5603.363	5660.064
6 Distribucion	4907.542	3889.159	3786.277	4194.326

100% de los datos

Req.	Toma 1 [ms]	Toma 2 [ms]	Toma 3 [ms]	Promedio [ms]
Carga	34166.58	33644.59	32753.89	33521.69
1 Alb. años	2927.007	2973.624	2943.794	2948.142
2 Top artistas	2860.986	2901.547	2877.053	2879.862
3 Top tracks	7623.99	7122.006	6924.828	7223.608
4 Mejor track	1779.861	1745.734	1740.842	1755.479
5 Discografía	6737.806	6826.565	6491.967	6685.446
6 Distribucion	4796.775	4582.338	4750.815	4709.976

III. Gráfica de comparación



Los resultados obtenidos concuerdan con el análisis de complejidad. La función que más tiempo tarda, la carga de datos CSV a arrays, tiene una complejidad teórica de O(n^2 logn) y tuvo una línea de mejor ajuste polínomica de grado 3. Esta es la función que más tiempo tardó debido a que, además de iterar los CSV y cargar los arrays, también debía hacer el preprocesamiento para los datos que pedía cada requerimiento.

La siguiente función es Top tracks, tuvo una complejidad de O(nlogn), es decir linearítmica, y su ajuste fue lineal. En seguida está la función Discografía, que se encarga de buscar la discografía completa para cierto artista. Esta función tiene una complejidad de O(n^2), debido a que debe iterar sobre las listas catalog['albums'] y catalog['tracks']. Sin embargo, gracias a que no debía organizar la lista esta función tardó ligeramente menos que la anterior.

La siguiente función fue la del requerimiento 6, de clasificar las canciones con mayor distribución. Esta tuvo una complejidad de O(nlogn), es decir, linearítmica, aunque el mejor ajuste que tuvo fue polinomial. Esto se debe a que a veces los algoritmos no se comportan como en el peor de los casos. Sin embargo, en perspectiva sí parece lineal. Le sigue la función Top artistas y Álbumes en rango de años, que ambas tuvieron una complejidad de O(nlogn), por lo que concuerda experimentalmente. Por último tenemos la función de buscar el mejor track para un artista dentro de un país. Esta tuvo una complejidad de O(n^2) y aun así fue la que más rápido se ejecutó. Esto es porque trabaja con arreglos de pocos elementos, y la búsqueda lineal y mergesort tardaron menos tiempo.