RETO No 1 Grupo 4

Req. 2 – Andrés Serrano – 201731766 – a.serranoc@uniandes.edu.co
Req. 3 – Antonio Martínez – 201921009 – a.martinez2@uniandes.edu.co

Justificación de uso de ARRAY:

Para cargar los archivos dentro del catalago, se usaron lista de tipo ARRAY en los tres casos (canciones, artistas, albumes), pues la función getElement fue utilizada repetidas veces para los tres grupos de datos. Como esta función es O(1) en ARRAY list y O(n) en SINGLE_LINKED list, se optó por el primer tipo.

Requerimiento 1 (grupal):

Análisis de complejidad: O(n log(n)).

La función principal usada en este requerimiento, cuya función es buscar los álbumes en un periodo de tiempo ingresado por parámetro fue:

```
def albumsInTimePeriod(anoI, anoF, catalog):
    """
    Retorna la sublista de álbumes con años de publicación los años indicados por parametro
    Si el año superior del rango es más pequeño que el primer año de publicación, retorna 0
    Si el año inferior del rango es más grande que el ultimo año de publicación, retorna 0
    """
    #se ordenan los albumes por año de publicación
    albumsByYear, delta_time = mergeSortAlbumsbyYear(catalog)

    posI = 1
    posF = lt.size(albumsByYear)

    start_time = getTime()

    #condición para año superior del rango

    if getAlbumYear(lt.getElement(albumsByYear, posI)) > anoF:
        return 0

    #condición para año superior del rango
    if getAlbumYear(lt.getElement(albumsByYear, posF)) < anoI:
        return 0

    findI = False
    findF = False

    i = 1

    while (not findI) or (not findF):
```

```
        anoActual = getAlbumYear(lt.getElement(albumsByYear, i))

        if anoActual >= anoI and (not findI):
            posI = i
            findI = True

        if anoActual > anoF and (not findF):
            posF = i - 1
            findF = True

        i += 1

    albumsByPeriod = lt.subList(albumsByYear, posI, posF + 1 - posI)

    #se mide el tiempo de ejecución del algoritmo:
    end_time = getTime()
    delta_time1 = deltaTime(start_time, end_time)

    return albumsByPeriod, lt.size(albumsByPeriod), delta_time+delta_time1
```

La complejidad de la primera operación es O(n log n), pues usa el algoritmo de ordenamiento mergeSort, siendo n la longitud de la lista de álbumes.

Luego, en las operaciones que siguen se tiene complejidad O(1):

```
posI = 1
    posF = lt.size(albumsByYear)
```

- Obtener el tamaño de la lista es O(1) para una lista de tipo ARRAY, que fue la escogida para la lista de álbumes dentro del catalogo.

```
if getAlbumYear(lt.getElement(albumsByYear, posI)) > anoF:
        return 0

    #condición para año superior del rango
    if getAlbumYear(lt.getElement(albumsByYear, posF)) < anoI:
        return 0
```

- La función lt.getElement en un posición dada tiene coplejidad O(1) para una lista ARRAY
- que fue la escogida para la lista de álbumes dentro del catalogo.
- La función getAlbumYear también tiene complejidad O(1), pues simplemente transforma el formato de la fecha de publicación de un álbum en el archivo original al formato AAAA (entero).
-

Finalmente, el ciclo al final de la función tiene complejidad O(n) en el peor de los casos, pues se recorre toda la lista de álbumes para encontrar los que tengan un año de publicación dentro del rango especificado. En este caso es importante que la lista de los álbumes sea tipo ARRAY, pues se usa la función lt.getElement (O(1)) a cada repetición del ciclo, por lo que si se hubiera escogido SINGLE_LINKED la complejidad de este ciclo sería O(n^3).

En conclusión, la complejidad de este requerimiento es **O(n log n),** pues esta es la mayor complejidad observada dentro de la función.

Requerimiento 2 (individual) : ANDRÉS SERRANO 201731766

Análisis de complejidad: O(n log(n)).

La función principal de este requerimiento fue la siguiente:

```
#Req 2
def getTopXArtists(cantidad, catalog):
    #Se inicia la función de toma de tiempo
    start_time = getTime()
    #Se organizan los artistas por popularidad haciendo uso de Merge Sort
    artistasOrganizados, tiempo=mergeSortArtists(catalog,lt.size(catalog["artists"]))
    #Se detiene la toma de tiempo
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return artistasOrganizados, delta_time
```

En este algoritmo, se organizaron los artistas por popularidad haciendo uso de Merge Sort, el cual tiene una complejidad de O(nlogn).

Todo el resto de operaciones que se realizan tienen una complejidad constante.

Por tanto, la complejidad de este requerimiento es: O(nlogn)

Requerimiento 3 (individual) : ANTONIO MARTINEZ 201921009

Análisis de complejidad: O(n log(n)).

La función principal de este requerimiento fue la siguiente:

```
def getTopXsongs(top, catalog):
    '''
    Crea una sublista con los ultimos X elementos de la lista de canciones
    ordenas por popularidad.
    '''
    sortedSongs, delta_time = mergeSortSongs(catalog)
    #se mide el tiempo de ejecución después del algortimo de ordenamiento
    start_time = getTime()
    #creación de la sublista
    topSongs = lt.subList(sortedSongs, 1,top)
    end_time = getTime()
    #se suma el tiempo medido al tiempo del algoritmo
    delta_time += deltaTime(start_time, end_time)
```

```
    return topSongs, delta_time
```

La primera operación fue organizar las canciones del catalogo por orden de popularidad. Esta operación tiene complejidad O(n log n), pues se usó el algoritmo merge sort. (con n la cantidad de canciones cargadas al catalogo).

Luego, se creo una sublista para encontrar las canciones TOP X. Esta operación tiene una complejidad O(1) para un ARRAY list, pues consiste en acceder a los datos dada un posición, que es una operación O(1)

Por tanto, la complejidad de este requerimiento es: O(1) + O(n log n) = O(n log n)

Requerimiento 4 (grupal) :

Análisis de complejidad: O(N).

La función principal de este requerimiento fue la siguiente:

```
def getBestSongs(artistName, country, catalog):
    """
    Retorna una lista ordenada por popularidad de las canciones asociadas
    al artista en el pais correspondiente.
    Retorna 0 si no encuentra al artista.
    Args:
        artistaName: str con el nombre del artista de interés
        country: str con el nombre del país.
    """
    artistSongs, artist= findArtistSongs(artistName, catalog)

    if artistSongs == 0:
        return 0, 0, 0

    songsByCountry, delta_time = getSongsByCountry(artistSongs, country)

    return songsByCountry, artist,  delta_time
```

La primera operación de la función retorna la lista de canciones del artista. Para hacerlo, debe encontrar el artista por el nombre, recorriendo la lista de artistas una vez. Por tanto, está operación tiene una complejidad O(N) (N siendo el número de artistas del catalogo).

En esta parte es importante tener en cuenta que, al cargar los datos, se creo una lista de las canciones del artista y se asoció al diccionario correspondiente de este. Esta operación hace que la carga de datos sea de complejidad O(n^2), pues por cada canción cargada, se usa la función isPresent (O(n) para una lista ARRAY) para encontrar el artista y agregarle la canción.

Luego, la operación de usa la función getSongsByCountry tiene una complejidad O(n log n), pero en este caso **n es el número de canciones de solo un artista**, mucho más pequeño que el

número total de artistas recorrido anteriormente. En efecto, esta función, que retorna una lista de canciones ordenada por popularidad y que estén disponibles en el mercado indicado, usa merge sort para ordenar las canciones del artista ya identificado.:

```
def getSongsByCountry(songs, country):
    """
    Retorna una lista de canciones disponibles en el pais indicado, ordenadas por popularidad de mayor a menor
    Retorna el tiempo de ejecución del ordenamiento

    """
    sortedSongs, delta_time = mergeSortSongList(songs)

    start_time = getTime()
    countryCode = pycountry.countries.search_fuzzy(country)[0].alpha_2
    SongsByCountry = lt.newList()
    for song in lt.iterator(sortedSongs):
        codeList = song['available_markets'].replace("","").strip('][').replace("'","").replace(' ','').split(',')
        for code in codeList:
            if code == countryCode:
                lt.addLast(SongsByCountry, song)
    end_time = getTime()
    delta_time += deltaTime(start_time, end_time)

    return SongsByCountry, delta_time
```

Finalmente, por lo anterior, la complejidad de este requerimiento es O(N).

Requerimiento 5 (grupal) :

Análisis de complejidad: O(N).

La función principal de este requerimiento fue la siguiente:

```
def numeroAlbumesPorTipoPorArtista(catalog,artista):
    #Se empieza a medir el tiempo
    start_time = getTime()
    artistas=catalog["artists"]
    #Se busca el artista en el catálogo a partir del nombre
    posArtista=""
    for i in range(1, lt.size(artistas)+1):
        if lt.getElement(artistas,i)["name"]==artista:
            posArtista=i
    if posArtista=="":
        return "No se encontró el artista"
    artista = lt.getElement(artistas, posArtista)
    #Se inicializan los contadores de tipos de album
    albumesSingle=0
    albumesCompilation=0
    albumesAlbum=0
```

```
albumesArtista = artista['albums']
cancionesArtista= artista['songs']
#Se recorren los álbumes del artista y se van contando los tipos de álbum
for album in lt.iterator(albumesArtista):
    if album["album_type"] == "album":
        albumesAlbum += 1
    if album["album_type"] == "single":
        albumesSingle += 1
    if album["album_type"] == "compilation":
        albumesCompilation += 1
#Se detiene la función de conteo de tiempo
end_time = getTime()
delta_time = deltaTime(start_time,end_time)

return albumesSingle,albumesCompilation,albumesAlbum, albumesArtista, cancionesArtista, delta_time
```

En primer lugar, este algoritmo recorre la lista de artistas buscando aquel cuyo nombre coincida con el ingresado por parámetro. Dado que en el peor caso el artista buscado estará en la última posición, la complejidad de esta parte es O(n).

A continuación, se recorrerán todos los álbumes del artista para clasificarlos. Dado que en el peor de los casos todos los álbumes del catálogo pertenecerán al artista, la complejidad de esta parte es también O(n).

Por último, solo se realizan ciclos de entre 1 y 3 iteraciones, por lo que en el peor de los casos la complejidad del resto del procedimiento es constante.

De esta forma, dado que el único procedimiento cuya complejidad no es constante se comporta como O(n), esta es la complejidad del requerimiento.

PRUEBAS DE TIEMPOS DE EJECUCION.

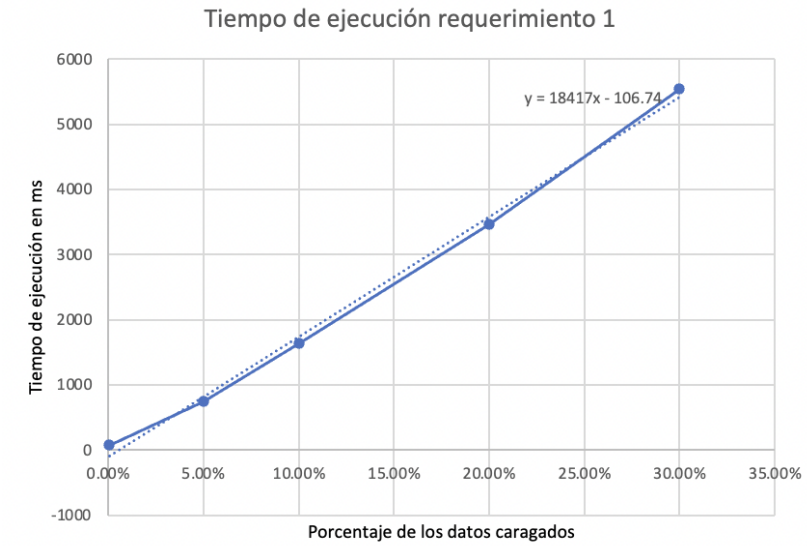
Especificaciones de la máquina utilizada para las pruebas.

Máquina		
Procesadores	Apple M1 pro	
Memoria RAM (GB)	16	
Sistema Operativo	macOS Monterey	

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento

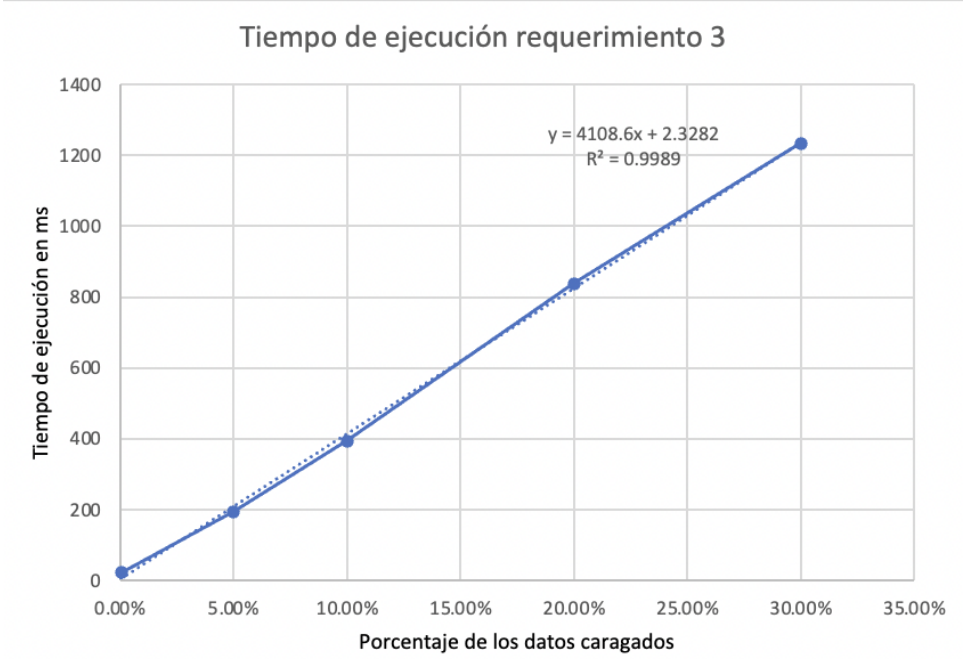
Tamaño de datos	Carga datos	Req 1	Req 2	Req 3	Req 4	Req 5
0.05%	534	70.45	36.52	22.9	69.3	1.67
5%	36427.52	741.42	245.18	195.3	71.5	10.64
10%	118822.94	1631.124	419.579	393.98	112.73	18.12
20%	364326.28	3462.87625	712.06	838.17	70.55	23.78
30%	719350.2	5540.602	903.26	1233.96	41.42	31.37

Tabla 2. Tiempos de ejecución medidos

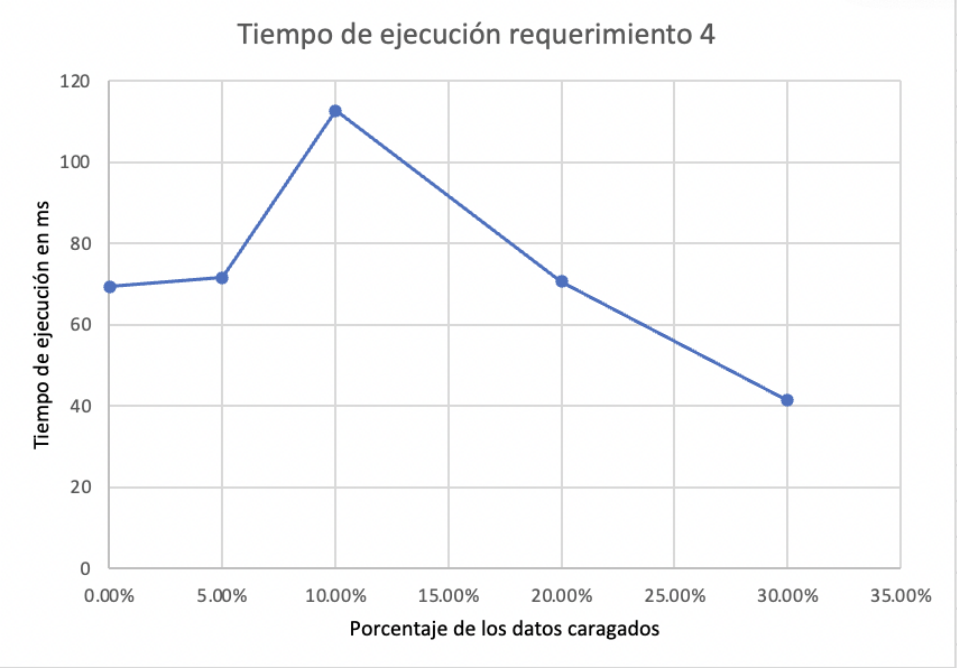


Gráfica 1. Tiempos de ejecución del requerimiento 1 en función de tamaño de datos.

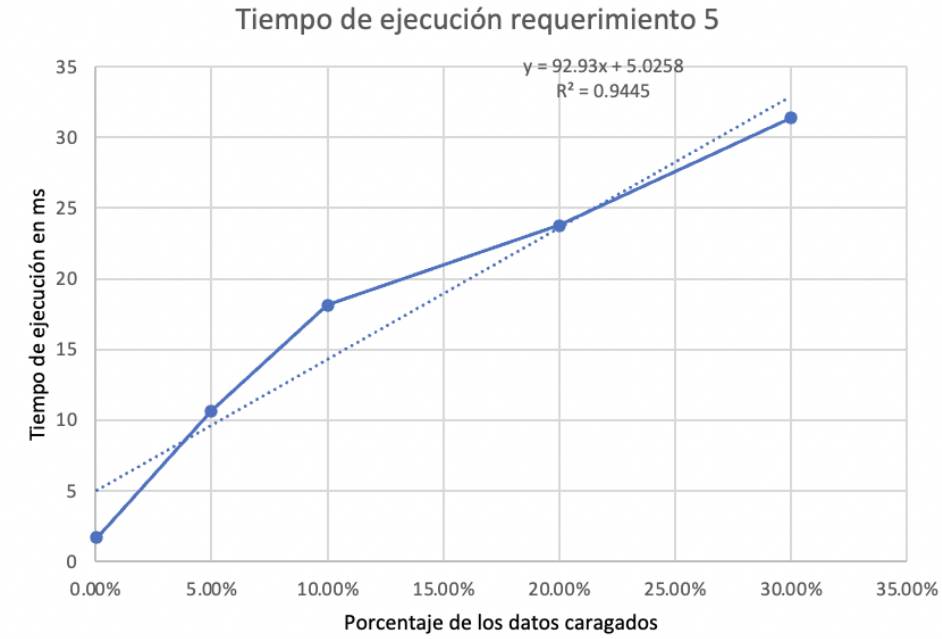
Gráfica 2. Tiempos de ejecución del requerimiento 2 en función de tamaño de datos.



Gráfica 3. Tiempos de ejecución del requerimiento 3 en función de tamaño de datos.



Gráfica 4. Tiempos de ejecución del requerimiento 4 en función de tamaño de datos.



Gráfica 5. Tiempos de ejecución del requerimiento 5 en función de tamaño de datos.

Como se puede observar en las ecuaciones presentadas en las gráficas, los resultados de complejidad temporal obtenidos experimentalmente coinciden con los análisis de complejidad realizados anteriormente. En efecto, para lso requerimientos 1, 2, 3 y 5, las complejidad observadas fueron menores a las esperadas. En los requerimientos 1, 2 y 3, se esperaba una complejidad O(n log n), y se encontró una relación lineal con el tamaño de los datos, lo que correspondería a una complejidad O(n) para ambos.

Sin embargo, esto no se cumple para la gráfica del requerimiento 4, en la cual se observa que los tiempos de ejecución decrecen a medida que aumenta el tamaño de los datos. Este decrecimiento se puede deber a algún cambio en los programas siendo ejecutados en la máquina en el momento de la prueba. Por ejemplo, algún programa pudo haber sido cerrado.
Finalmente, el requerimiento 5 parece seguir la complejidad temporal O(n) esperada, pues al llegar al máximo de datos que se pudieron cargar, se cumple una relación lineal entre el tiempo de ejecución y la cantidad de datos utilizados.