Análisis de complejidad de requerimientos: Reto 1 EDA

Lukas Mateo Calderón (l.calderon2@uniandes.edu.co) – 201923480 – Requerimiento 3
Daniel Santiago Rueda Villalba (d.ruedav@uniandes.edu.co) –202112934–Requerimiento2

Especificaciones de la máquina donde fueron tomados los datos de este informe:

Procesador	Intel Core i5 2.30GHz
Memoria RAM (GB)	8.0 GB
Sistema Operativo	Windows 10 Pro

Metodología para la toma de datos de tiempo:

Para la toma de datos de tiempo de ejecución de cada requerimiento, se implementaron las funciones getTime() y deltaTime() así como se hizo durante los laboratorios, y se tomó el tiempo de la **ejecución del algoritmo del *model.py,*** no se tomó en cuenta el tiempo que demoró imprimir los datos en pantalla.

Los inputs que siempre se usaron para cada tamaño de archivo fueron:

- Req. 1: Año inicial: 1980. Año final: 2010.
- Req. 2: Buscar el Top 50 artistas.
- Req. 3: Buscar el Top 50 canciones.
- Req. 4: Buscar la canción más popular de “Ozuna” en Colombia.
- Req. 5: Buscar la discografía de “Ozuna”.
- Req. 6 (BONO): Buscar el Top 50 canciones con mayor distribución entre 1980 y 2010.

Los tiempos fueron redondeados a la unidad o media unidad más cercana (,0 o ,5)

Requerimiento 1 (Grupal): Consultar álbumes por periodo de tiempo.

La función dentro del *model.py* que realiza este requerimiento es la siguiente:

```
def getAlbumsByYear(catalog:Catalog, a_inicio, a_final):
    #Devuelve los albumes publicados en un periodo de tiempo
    lst = sortAlbums(catalog["albums"])
    result = lt.newList("ARRAY_LIST")
    for album in lt.iterator(lst):
        if a_inicio<=int(album["release_date"])<=a_final:
            lt.addLast(result, album)
        elif a_final<int(album["release_date"]):
            break
    return result
```

Análisis de complejidad: Se llama a la función sortAlums, la cual hace el ordenamiento merge de la lista de álbumes por año de publicación (complejidad O(n log n)), posteriormente se recorre dicha lista ordenada, elemento por elemento, identificando cuáles álbumes hacen parte del periodo de tiempo solicitado por el usuario (complejidad del peor caso O(n)). Para una complejidad total de O(n log n + n) = O(n log n).

Tabla de análisis de tiempo y gráfica:

Tamaño del archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo (ms)	27.5	433.0	880.5	1514	2128.5	3550	4826.5	5457

Se puede apreciar un comportamiento linearítmico de la función de tiempo para este requerimiento. Precisamente como se predijo en el análisis de complejidad.

Requerimiento 2 (Individual): Consultar los Top X artistas más populares **(Realizado por Daniel Santiago Rueda).**

La función dentro del *model.py* que realiza este requerimiento es la siguiente:

```python
def getTopArtists(catalog, size):
    #Devuelve el top X de artistas por popularidad
    lista_ordenada = sortArtists(catalog["artists"])
    top_artists = lt.newList("ARRAY_LIST")
    for i in range(1, size+1):
        artist = lt.getElement(lista_ordenada, i)
        lt.addLast(top_artists, artist)
    return top_artists
```

Análisis de complejidad: Se llama a la función sortArtists, que ordena con merge a los artistas principalmente por popularidad (complejidad O(n log n)), posteriormente se agregan los primeros X elementos de dicha lista ordenada a una nueva lista top_artists que es retornada al usuario (complejidad O(s), con s igual al tamaño de la lista que desea el usuario, con el peor caso siendo cuando se pide el top 100% de artistas: O(n)). Para una complejidad total de O(n log n + s) = O(n log n).

Tabla de análisis de tiempo y gráfica:

Tamaño del archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo (ms)	80	696.0	1003	1638	2304.5	2915	3631.5	4097

Se puede apreciar un comportamiento linearítmico del tiempo de ejecución del algoritmo, justo como se predijo en el análisis.

Requerimiento 3 (Individual): Consultar las Top X canciones más populares **(Realizado por Lukas Mateo Calderón).**

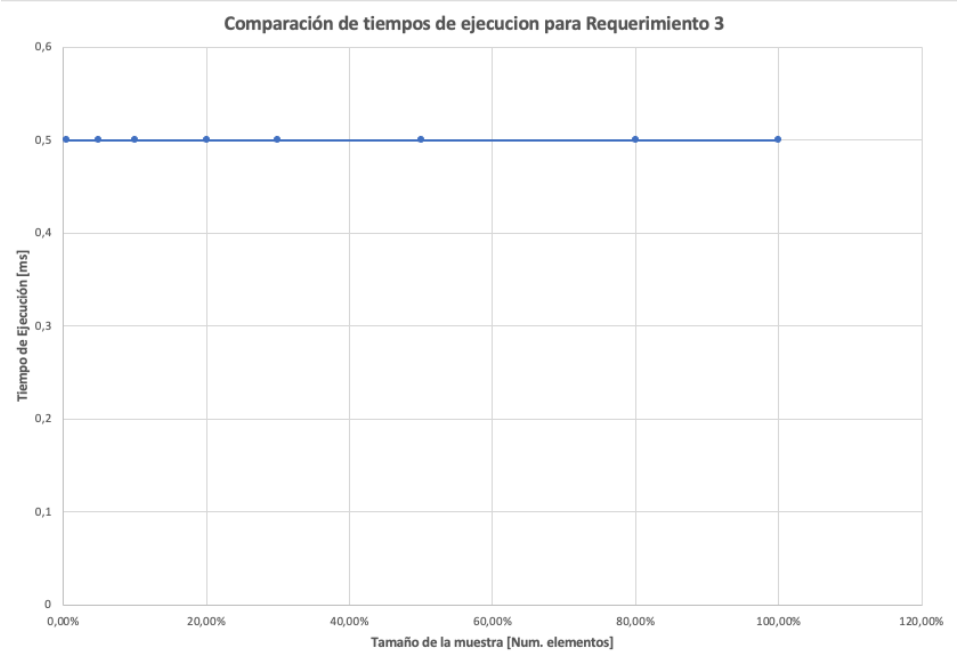
La función dentro del *model.py* que realiza este requerimiento es la siguiente:

```
def getTopTracks(catalog: Catalog, size: int):
    """
    Retorna los top x tracks (ordenadas por popularidad de mayor a menor)
    """
    # ya estan ordenadas por popularidad los tracks
    mejores_tracks = catalog["tracks"]  # sortTracks(catalog["tracks"])
    top_tracks = lt.newList("ARRAY_LIST")
    for i in range(1, size + 1):
        artist = lt.getElement(mejores_tracks, i)
        lt.addLast(top_tracks, artist)
    return top_tracks
```

Análisis de complejidad: Como ya la lista de tracks en el catálogo están ordenadas en orden decreciente de popularidad, dentro de la función simplemente se hace referencia a la lista de tracks (O(1)), se crea una nueva lista (O(1)), y para cada track en el rango pedido, se agregan los tracks en orden de popularidad decreciente a top_tracks (O(c), con c igual al top de canciones que pida el usuario). Por ultimo se retorna la nueva lista creada con los top tracks dentro del rango pedido. Esto significa que el orden de crecimiento de esta función es de O(c) (peor caso O(n)), ya que depende de la cantidad de canciones pedidas.

Tabla de análisis de tiempo y gráfica:

Tamaño del archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo (ms)	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5



La complejidad es constante, debido a que siempre se le pidió al programa el mismo top de canciones, justo como se predijo en el análisis.

Requerimiento 4 (Grupal): Consultar la mejor canción de un artista por país.

La función dentro del *model.py* que realiza este requerimiento es la siguiente:

```
def getBestTrack(catalog, name, country):
    #Devuelve la mejor canción de un artista
    pos = 1
    for art in lt.iterator(catalog["artists"]):
        if art["name"] == name:
            artist = lt.getElement(catalog["artists"], pos)
            break
        pos+=1
    for t in lt.iterator(artist["songs"]):
        if country in t["available_markets"]:
            best_song = t
            break
    return artist, best_song
```

Análisis de complejidad: Dado un nombre de artista, primero se hace una búsqueda lineal de este dentro del catalogo de artistas (complejidad del peor caso O(n)). Posteriormente, se examina canción por canción asociada a ese artista (artist[“songs”]) que durante la carga de datos ya fue organizada por popularidad, para así buscar solamente la primera canción que sea distribuida en el país dado (complejidad O(s), con s siendo el tamaño de la lista de canciones de un artista, el cual es pequeño comparado con el tamaño del archivo completo de tracks). Por lo tanto, la complejidad total de este algoritmo será O(s) (peor caso O(n)).

Tabla de análisis de tiempo y gráfica:

Tamaño del archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo (ms)	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Como se le pidió al programa devolver siempre la canción más popular del mismo artista, el algoritmo tardó un tiempo constante en dar respuesta, así como se predijo en la teoría.

Requerimiento 5 (Grupal): Consultar la discografía de un artista.

La función dentro del *model.py* que realiza este requerimiento es la siguiente:

```
def getDiscography(catalog, artist_name):
    pos = 1
    for art in lt.iterator(catalog["artists"]):
        if art["name"] == artist_name:
            artist = lt.getElement(catalog["artists"], pos)
            break
        pos+=1
    cont_single = 0
    cont_compilation = 0
    cont_album = 0
    for a in lt.iterator(artist["albums"]):
        if a["album_type"]=="single":
            cont_single+=1
        elif a["album_type"]=="compilation":
            cont_compilation+=1
        elif a["album_type"]=="album":
            cont_album+=1
    return cont_single, cont_compilation, cont_album, artist
```

Análisis de complejidad: Dado el nombre de un artista, al igual que en el requerimiento anterior, se busca linealmente dicho nombre dentro del catálogo de artistas (complejidad del peor caso O(n)). Posteriormente, se recorre la lista de álbumes asociados a dicho artista (artist["albums"]) y se identifica los que son tipo single, compilation o álbum (complejidad O(a), con a siendo el tamaño de la lista de álbumes relacionados al artista).
Para una complejidad total de O(a) (Peor caso O(n)).

NOTA: Como durante la carga de datos se ordenó a las canciones por popularidad para garantizar la eficiencia de esta carga, no es necesario ordenarlas en este requerimiento para identificar la canción más popular de cada álbum.

Tabla de análisis de tiempo y gráfica:

Tamaño del archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo (ms)	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

La función constante de este requerimiento para el caso particular de la entrada de datos que se utilizó esta de acuerdo con lo teorizado, debido a que la lista de álbumes de un artista es pequeña a comparación del tamaño total del archivo de álbumes.

Requerimiento 6 BONO (Grupal): Clasificar las canciones con mayor distribución.

La función dentro del *model.py* que realiza este requerimiento es la siguiente:

```python
def getTopTracksbyYear(catalog, a_inicio, a_final, top):
    lst = sortTracksbyCountry(catalog)
    result = lt.newList("ARRAY_LIST")
    for track in lt.iterator(lst):
        if int(lt.size(result))<top:
            if a_inicio<=int(track["release_date"])<=a_final:
                lt.addLast(result, track)
        else:
            break
    return result
```

Análisis de complejidad: Primero se ordena la lista de canciones con merge principalmente por número de mercados en los que la canción está disponible (complejidad O(n log n)). Luego, por cada track en esta lista ordenada se busca linealmente aquellas primeras X que tengan año de lanzamiento entre el periodo solicitado por el usuario (complejidad del peor caso O(n)). Para una complejidad total de O(n log n + n) = O(n log n).

Tabla de análisis de tiempo y gráfica:

Tamaño del archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo (ms)	44.5	459	961.5	1847	2810	4907.5	6443	8432.5

Se observa un comportamiento irregular durante las pruebas del 80 y 100% de los datos. Sin embargo, una buena aproximación a esta función es una función linearítmica que la acote superiormente