OBSERVACIONES DEL RETO 1

Integrantes individuales:

Cristian Camilo Cortes Moreno Cod. 202011908 (cc.cortesm1@uniandes.edu.co)
Natalia Villegas Calderón Cod. 202113370 (n.villegasc@uniandes.edu.co)

Requerimientos individuales:

Requerimiento 2: Natalia Villegas Calderón

Requerimiento 3: Cristian Camilo Cortes Moreno

Analisis de complejidad → O() + Tiempo de ejecución:

Porcentaje de la muestra [pct]	Carga	Req 1	Req 2	Req 3	Req 4	Req 5	Req 6
0.50%	194,22	2,25	0,33	0,22	0,44	1,08	6,86
5.00%	1793,58	2,96	0,05	0,52	0,74	0,52	50,73
10.00%	3414,58	8,65	0,27	0,45	0,22	0,98	92,11
20.00%	6514,42	25,61	0,23	0,27	0,18	2,19	141,28
100.00%	24374,45	68,72	0,3	0,68	0,97	5,25	538,37

Tiempo de ejecución vs porcentaje muestra

Requerimiento 1:

O(nlog(n))

Range By date hace una busqueda lineal del primer dato y el segundo es binaria y hace una sublist del rango.

```
#Requerimiento 1
def albumsByReleaseYear(catalog, min, max):
    start_time = getTime()
    albumsByReleaseYear = rangeByDate(catalog['albumsByYear'], min, max, 'year')
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return albumsByReleaseYear, delta_time
```

Requerimiento 2:

O(n)

En caso de querer listar toda lista sería O(n), la complejidad varía según la cantidad de datos a listar.

```python
#Requerimiento 2
def artistByPopularity(catalog, top):
    start_time = getTime()
    topartistpopularity = lt.subList(catalog['artistByPopularity'], 1, top)
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return topartistpopularity, delta_time
```

Requerimiento 3:

O(n)

En caso de querer listar toda lista sería O(n), la complejidad varía según la cantidad de datos a listar.

```python
#Requerimiento 3
def tracksByPopularity(num, catalog):
    start_time = getTime()
    tracksByPopularity = getFirstNum(num, catalog['tracksByPopularity'])
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return tracksByPopularity, delta_time
```

Requerimiento 4:

N = Tamaño Lista de Artistas

O(log(n))

Se hace una busqueda binaria del artista y se recorren sus tracks, los tracks son un numero considerablemente pequeño para considerarlo.

```python
#Requerimiento 4
def popularTrackByArtist(catalog, artistName, countryCode):
    start_time = getTime()
    filteredTracks = lt.newList('ARRAY_LIST')
    artist = lt.getElement(catalog['artistByName'], binarySearch(catalog['artistByName'], artistName, 'name'))
    for track in lt.iterator(artist['tracks']):
        if countryCode in track['available_markets']:
            lt.addLast(filteredTracks,track)
    popularity = 0
    duration_ms = 0
    name = ''
    maxiumTrack = lt.getElement(filteredTracks, 1)
    for track in lt.iterator(filteredTracks):
        if float(track['popularity']) > float(popularity):
            popularity = float(track['popularity'])
            duration_ms = track['duration_ms']
            name = track['name']
            maxiumTrack = track
        elif float(track['popularity']) == float(popularity):
            if float(track['duration_ms']) > float(duration_ms):
                duration_ms = track['duration_ms']
                name = track['name']      You, 7 hours ago • Posible finished req 4 …
                maxiumTrack = track
            elif float(track['duration_ms']) == float(duration_ms):
                if track['name'] > name:
                    name = track['name']
                    maxiumTrack = track
    trackArray = lt.newList('ARRAY_LIST')
    lt.addLast(trackArray,maxiumTrack)
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return trackArray, delta_time
```

Requerimiento 5:

O(log(n))

Se hace una busqueda binaria del artista y se recorren sus tracks, los tracks son un numero considerablemente pequeño para considerarlo.

```python
#Bono
def tracksByDistributionInRange(catalog, min, max, top):
    start_time = getTime()
    listTracksByYear = rangeByDate(catalog['tracksByYear'], min, max, 'release_date')
    orderByDistributionTracks = sortLst(listTracksByYear,lt.size(listTracksByYear), cmpDistribution)
    topDistributionTracks = lt.subList(orderByDistributionTracks, 1, top)
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return topDistributionTracks, delta_time
```

Requerimiento 6:

O(nlog(n))

Range By date hace una busqueda lineal del primer dato y el segundo es binaria y hace una sublist del rango y despues organiza por distribution.

```python
#Bono
def tracksByDistributionInRange(catalog, min, max, top):
    start_time = getTime()
    listTracksByYear = rangeByDate(catalog['tracksByYear'], min, max, 'release_date')
    orderByDistributionTracks = sortLst(listTracksByYear,lt.size(listTracksByYear), cmpDistribution)
    topDistributionTracks = lt.subList(orderByDistributionTracks, 1, top)
    end_time = getTime()
    delta_time = deltaTime(start_time, end_time)
    return topDistributionTracks, delta_time
```

Gráfica: