

# Estructura de datos y algoritmos: Reto 1

Estudiantes:

- Req 2. - Raul Santiago Rincon, r.rincon@uniandes.edu.co, 202120414.
- Req 3. - Benjamin Raisbeck Garcia, b.raisbeck@uniandes.edu.co, 202120398.

Análisis de complejidad de cada requerimiento:

Requerimiento 1:

```
#REQUERIMIENTO 1

def listaPeriodoTiempoAlbums(catalog, fecha1, fecha2):
    nuevalista = lt.newList()
    for i in range(lt.size(catalog)):
        albumData = lt.getElement(catalog, i)
        if int(fecha1) <= int(albumData['release_date']) <= int(fecha2):
            lt.addLast(nuevalista, albumData)
    return nuevalista
```

La complejidad de este requerimiento sería  $O(n)$  puesto que se itera sobre el size o el tamaño del catalogo por lo tanto depende de cuantos datos estemos cargando en catalogo para que se demore más o menos este algoritmo.

Requerimiento 2:

```
def getBestArtist(catalog, number):
    """
    Retorna los mejores artistas
    """
    bestartist = lt.newList()
    for cont in range(1, number+1):
        artist = lt.getElement(catalog, cont)
        lt.addLast(bestartist, artist)
    return bestartist
```

La complejidad de este requerimiento sería  $O(n)$  puesto que depende del TOP( $n$ ) que solicita el usuario para crear la nueva lista y retornarla con los artistas mas famosos.

Requerimiento 3:

```
def sortTracks(catalog):  
    tracks_ordenados = quick.sort(catalog, cmpSongsByPopularity)  
    return tracks_ordenados
```

```
def cmpSongsByPopularity(song1, song2):  
    if sortTracks(song1['popularity']) < sortTracks(song2['popularity']):  
        return sortTracks(song1['duration_ms']) > sortTracks(song2['duration_ms'])  
    elif sortTracks(song1['duration_ms']) < sortTracks(song2['duration_ms']):  
        return sortTracks(song1['name']) > sortTracks(song2['name'])  
    return sortTracks(song1['popularity']) > sortTracks(song2['popularity'])
```

Este requerimiento seria dependiendo del Quicksort que en promedio seria de una complejidad temporal de  $O(n \log n)$  y en el peor de los casos  $O(n^2)$ .

Requerimiento 4:

```
def findArtistDisco(artist_list, track_list, name):  
    artist_discography = []  
    for artist in artist_list['elements']:  
        if name == artist['name']:  
            artist_id = artist['id']  
            artist_id.rstrip()  
    for track in track_list['elements']:  
        track['artists_id'] = track['artists_id'].strip(" ").replace(" ", "").replace("'", "").split(",")  
        if len(track['artists_id']) > 1:  
            for element in track['artists_id']:  
                if artist_id == element:  
                    artist_discography.append(track)  
        elif len(track['artists_id']) == 1:  
            if artist_id == track['artists_id'][0].rstrip():  
                artist_discography.append(track)  
    if len(artist_discography) == 0:  
        print("No se encontraron canciones")  
    else:  
        return artist_discography
```

La complejidad de este algoritmo seria  $O(n*m)$  ya que se recorren dos lista diferentes por lo tanto se debe iterar en cada una y esto hace la complejidad de esta forma

Requerimiento 5:

PRUEBAS DE TIEMPO:

Cantidad (Datos)	REQ 1	REQ 2	REQ 3	REQ 4	REQ 5
100	311.02(ms)	265.14(ms)	125.54(ms)	454.01 (ms)	315.13(ms)
1000	413.13	351.56	413.13	413.13	413.13
10000	1031.13	1145.13	591.16	1350.04	763.45
50000	1454.14	1255.32	743.87	3145.54	6404.76