**Integrantes:**

Req 2 - Esteban Leiva, 202021368, e.leivam@uniandes.edu.co

Req 3 - Michelle Vargas, 201914771, [bm.vargas@uniandes.edu.co](mailto:bm.vargas@uniandes.edu.co)

**Análisis de complejidad, tiempo y memoria:**

*REQ 1*:

def req1(*menor*,*mayor*,*feature*,*catalog*):

    delta\_time = -1.0

    delta\_memory = -1.0

    tracemalloc.start()

    start\_time = getTime()

    start\_memory = getMemory()

    events = om.values(catalog[feature],menor,mayor) #1 O(lgN)

    num\_events = model.numevents(events) #2 O(N)

    num\_artists = (model.artists(events,None))[0] #3 O(N)

    stop\_memory = getMemory()

    stop\_time = getTime()

    tracemalloc.stop()

    delta\_time = stop\_time - start\_time

    delta\_memory = deltaMemory(start\_memory, stop\_memory)

    print('\n'+(feature.capitalize())+' is between '+str(menor)+' and '+str(mayor)+'\nTotal of reproduction: '+str(num\_events)+'\nTotal of unique artists: '+str(num\_artists)+'\n')

*return* delta\_time, delta\_memory

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 1 es O(N).

Tiempo y consumo de memoria:

|  |  |
| --- | --- |
| Tiempo (ms) | Memoria (kB) |
| 155.305 | 80.898 |

*REQ 2*:

def req2(*catalog*,*min\_en*,*max\_en*,*min\_dan*,*max\_dan*):

    keys1 = om.values(catalog["energy"],min\_en,max\_en) #1 O(lgN)

    lista1 = model.listaeventos(catalog["energy"],keys1) #2 O(N)

    mapa = om.newMap(*omaptype*="RBT")

    i1 = it.newIterator(lista1)

*while* it.hasNext(i1): #3 O(N)

        event = it.next(i1)

        model.addtomap2(mapa, event,"danceability")

    keys2 = om.values(mapa,min\_dan,max\_dan) #4 O

    lista2 = model.listaeventos(mapa,keys2) #5 O

    mapafin = mp.newMap(*maptype*="PROBING",*loadfactor*=0.5)

    i2 = it.newIterator(lista2)

*while* it.hasNext(i2): #6 O(N)

        event = it.next(i2)

        mp.put(mapafin, event["track\_id"],event)

    print('\Energy is between '+str(min\_en)+' and '+str(max\_en)+'\Danceability is between '+str(min\_dan)+' and '+str(max\_dan)+'\nTotal of unique tracks in events: '+str(mp.size(mapafin))+'\n')

    n = 1

    listakeys = mp.keySet(mapafin)

*while* n<=5: #7 O(1)

        llave = lt.getElement(listakeys, n)

        event = mp.get(mapafin,llave)["value"]

        print('Track '+str(n)+': '+ event["track\_id"]+' with energy of '+str(event["energy"])+' and danceability of '+str(event["danceability"]))

        n +=1

    print('\n')

def req2(*catalog*,*min\_en*,*max\_en*,*min\_dan*,*max\_dan*):

    delta\_time = -1.0

    delta\_memory = -1.0

    tracemalloc.start()

    start\_time = getTime()

    start\_memory = getMemory()

    listadeentries1 = om.values(catalog["energy"],min\_en,max\_en) #1 O(lgN)

    mapa1 = model.mapaeventos(listadeentries1) #2 O(N)

    listadeentries2=om.values(catalog["danceability"],min\_dan,max\_dan) #3 O(lgN)

    tracksencomun=model.tracksencomun(mapa1,listadeentries2) #4 O(N)

    print('\nEnergy is between '+str(min\_en)+' and '+str(max\_en))

    print('Danceability is between '+str(min\_dan)+' and '+str(max\_dan))

    print('Total of unique tracks in events: '+str(tracksencomun[0]))

    print('\n--- Unique track\_id ---')

    lista=tracksencomun[1]

    i=it.newIterator(lista)

    x=1

*while* it.hasNext(i): #5 O(1)

        tupla=it.next(i)

        features=tupla[1]

        print('Track '+str(x)+': '+tupla[0]+' with energy of '+str(features[0])+' and danceability of '+str(features[1]))

        x+=1

    stop\_memory = getMemory()

    stop\_time = getTime()

    tracemalloc.stop()

    delta\_time = stop\_time - start\_time

    delta\_memory = deltaMemory(start\_memory, stop\_memory)

*return* delta\_time, delta\_memory

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 2 es O(N).

Tiempo y consumo de memoria:

|  |  |
| --- | --- |
| Tiempo (ms) | Memoria (kB) |
| 589.714 | 13045.141 |

*REQ 3*:

def req3(*catalog*,*min\_inst*,*max\_inst*,*min\_temp*,*max\_temp*):

    delta\_time = -1.0

    delta\_memory = -1.0

    tracemalloc.start()

    start\_time = getTime()

    start\_memory = getMemory()

    listadeentries1 = om.values(catalog['instrumentalness'],min\_inst,max\_inst) #1 O(lgN)

    mapa1 = model.mapaeventos(listadeentries1) #2 O(N)

    listadeentries2=om.values(catalog['tempo'],min\_temp,max\_temp) #3 O(lgN)

    tracksencomun=model.tracksencomun(mapa1,listadeentries2) #4 O(lgN)

    print('\Instrumentalness is between '+str(min\_inst)+' and '+str(max\_inst))

    print('Tempo is between '+str(min\_temp)+' and '+str(max\_temp))

    print('Total of unique tracks in events: '+str(tracksencomun[0]))

    print('\n--- Unique track\_id ---')

    lista=tracksencomun[1]

    i=it.newIterator(lista)

    x=1

*while* it.hasNext(i): #5 O(1)

        tupla=it.next(i)

        features=tupla[1]

        print('Track '+str(x)+': '+tupla[0]+' with instrumentalness of '+str(features[0])+' and tempo of '+str(features[1]))

        x+=1

    stop\_memory = getMemory()

    stop\_time = getTime()

    tracemalloc.stop()

    delta\_time = stop\_time - start\_time

    delta\_memory = deltaMemory(start\_memory, stop\_memory)

*return* delta\_time, delta\_memory

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 3 es O(N).

Tiempo y consumo de memoria:

|  |  |
| --- | --- |
| Tiempo (ms) | Memoria (kB) |
| 237.509 | 5415.328 |

*REQ 4*:

def req4(*catalog*,*genre*,*minimo*,*maximo*):

    mapa = catalog["tempo"]

*if* minimo==None and maximo==None:

        x=model.tempobygenre(genre) #1 O(1)

        menor=x[0]

        mayor=x[1]

*else*:

        menor=minimo

        mayor=maximo

    events = om.values(mapa,menor,mayor) #2 O(lgN)

    num\_events = model.numevents(events) #3 O(N)

    artists=(model.artists(events,10)) #4 O(N)

    numartists=artists[0]

    listartists=artists[1]

    print('\n======= '+genre.upper()+' ========'+'\nFor '+genre+' the tempo is between '+str(menor)+' and '+str(mayor)+'\n'+genre+' reproductions: '+str(num\_events)+' with '+str(numartists)+' different artists'+'\n\n---- Some artists for '+genre+' -----\n')

    i=it.newIterator(listartists)

    n=1

*while* it.hasNext(i): #5 O(1)

        artist=it.next(i)

        print('Artist '+str(n)+': '+artist)

        n+=1

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 4 es O(N).

Tiempo y consumo de memoria:

|  |  |
| --- | --- |
| Tiempo (ms) | Memoria (kB) |
| 15199.954 | 21.375 |

*REQ 5*:

def req5(*catalog*,*minim*,*maxim*):

    delta\_time = -1.0

    delta\_memory = -1.0

    tracemalloc.start()

    start\_time = getTime()

    start\_memory = getMemory()

    mapa=mp.newMap(*maptype*="PROBING",*loadfactor*=0.5)

    mapafinal=mp.newMap()

    total=0

    mayor=None

    mayorkey=0

    lista=om.values(catalog['time'],minim,maxim) #1 O(lgN)

    mapa=model.genresandtracks(lista) #2 O(N)

    genres=mp.keySet(mapa)

    i=it.newIterator(genres)

*while* it.hasNext(i): #3 O(N)

        genre=it.next(i)

        x=mp.get(mapa,genre)

        entry=me.getValue(x)

        eventos=mp.size(entry['events'])

        mp.put(mapafinal,eventos,genre)

        total+=eventos

    numevents=mp.keySet(mapafinal)

    ordered=mrge.sort(numevents,cmpnums) #4 O(NlgN)

    i=it.newIterator(ordered)

    n=1

*while* it.hasNext(i): #5 O(1)

        key=it.next(i)

        par=mp.get(mapafinal,key)

        value=me.getValue(par)

*if* n==1:

            mayor=value

            mayorkey=key

        print('TOP '+str(n)+': '+value.capitalize()+' with '+str(key)+' reps')

        n+=1

    pareja=mp.get(mapa,mayor)

    entry=me.getValue(pareja)

    tuplas=mp.keySet(entry['events'])

    x=model.orderednums(tuplas) #6 O(N)

    mapanums=x[0]

    print('\nThe TOP GENRE is '+mayor.capitalize()+' with '+str(mayorkey)+' reproductions')

    print('========== '+mayor.upper()+' SENTIMENT ANALYSIS ==========')

    print(mayor.capitalize()+' has '+str(mp.size(x[1]))+' unique tracks')

    listanums=lt.newList(*datastructure*='ARRAY\_LIST')

    llavesnums=(om.keySet(mapanums))

    m=1

*while* it.hasNext(n):

        num=it.next(n)

        lt.addLast(listanums,num)

    mergednums=mrge.sort(listanums,cmpnums) #7 O(NlgN)

    centinela=True

    d=it.newIterator(mergednums)

*while* it.hasNext(d) and centinela==True:

        num=it.next(d)

        par=om.get(mapafinal,num)

        listatuplas=me.getValue(par)

        t=it.newIterator(listatuplas)

*while* it.hasNext(t) and centinela==True:

            tupla=it.next(t)

            print('TOP '+str(m)+' track: '+tupla[0]+' with '+str(num)+' hashtags and VADER = '+str(tupla[1]))

            m+=1

*if* m>10:

                centinela=False

    print('\n')

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 5 es O(NlgN).

Tiempo y consumo de memoria:

|  |  |
| --- | --- |
| Tiempo (ms) | Memoria (kB) |
|  |  |