# Análisis de los resultados: Reto 3

Estudiante A: Santiago Ballén Cod 202023544

Estudiante B: Paola Campiño Cod 202020785

Reto 3:

https://github.com/EDA2021-1-SEC07-G-2Grupo/Reto3-S07-G02.git

El documento continuación presenta los siguientes datos:

- Análisis de complejidad en notación O para cada uno de los requerimientos.
- Análisis de tiempo de ejecución y uso de memoria para cada uno de los requerimientos implementados.
- Gráfico con análisis de cada uno de los requerimientos implementados.

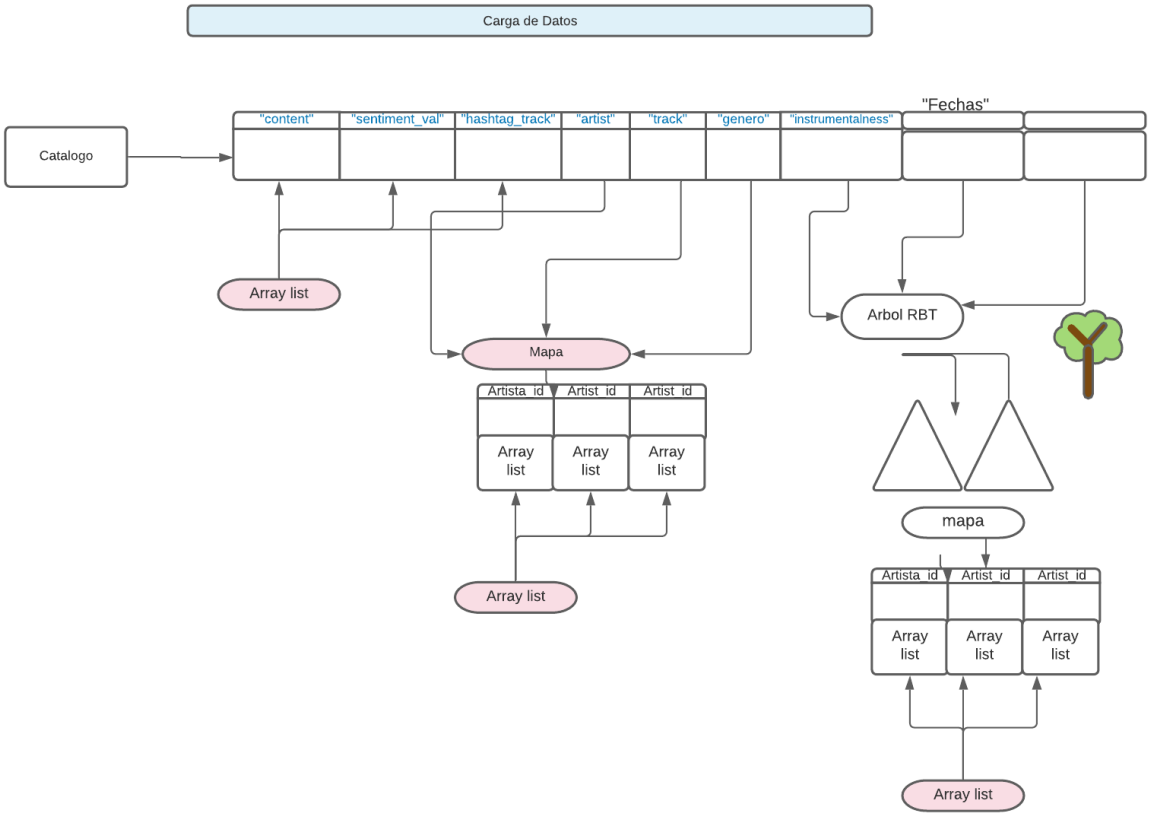
Cabe resaltar que los análisis de tiempo de ejecución y consumo de memoria se hicieron con el csv small y se ha creado un csv nuevo en el que se guardan los géneros musicales.

## Carga de Datos:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

| | Tiempo de Ejecución [ms] |
|---|---|
| | |

Para este reto fue necesaria la creación de 3 mapas




# Requerimiento 1: Caracterizar las reproducciones

## Tiempos de ejecución y consumo de Datos:

Datos con los que se evaluó el requerimiento

| Caracteriastica a consultar | Rango |
|---|---|
| “valence” | 0-1 |

Respuesta:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

| Consumo de Datos[kB] | Tiempo de Ejecución [ms] |
|---|---|
| 171.75 | 70.31 |

## Complejidad del Requerimiento:

La primera parte del requerimiento implica hacer una búsqueda dado un rango de valores con la función values() como se muestra a continuación:

```
total=controller.range_values(catalog[caracteristica],valor_min,valor_max)
```

### la complejidad hasta este punto es de: O(k)

Lo segundo que hay que hacer es hacer un recorrido por la lista que se obtuvo de la función anterior para sumar “size” con el fin de obtener el total de reproducciones, como se muestra a continuación:

```
controller.conteo_range_value(total)

def conteo_range_value(lista):

    total = 0

    for char in lt.iterator(lista):

        total += lt.size(char['song'])

    return total
```

Hasta este punto la complejidad de el requerimiento es de: O(N), N siendo la cantidad de mapas que se encontraron con la función anterior.

Por lo que hasta este punto la complejidad total es de: O(N), N siendo la cantidad de mapas encontrados en el rango pedido con la función value()

Lo ultimo que hay que hacer para este requerimiento es hacer el conteo de llaves, asociadas al valor dentro del rango solicitado. Para este punto se uso la función que aparece a continuación:

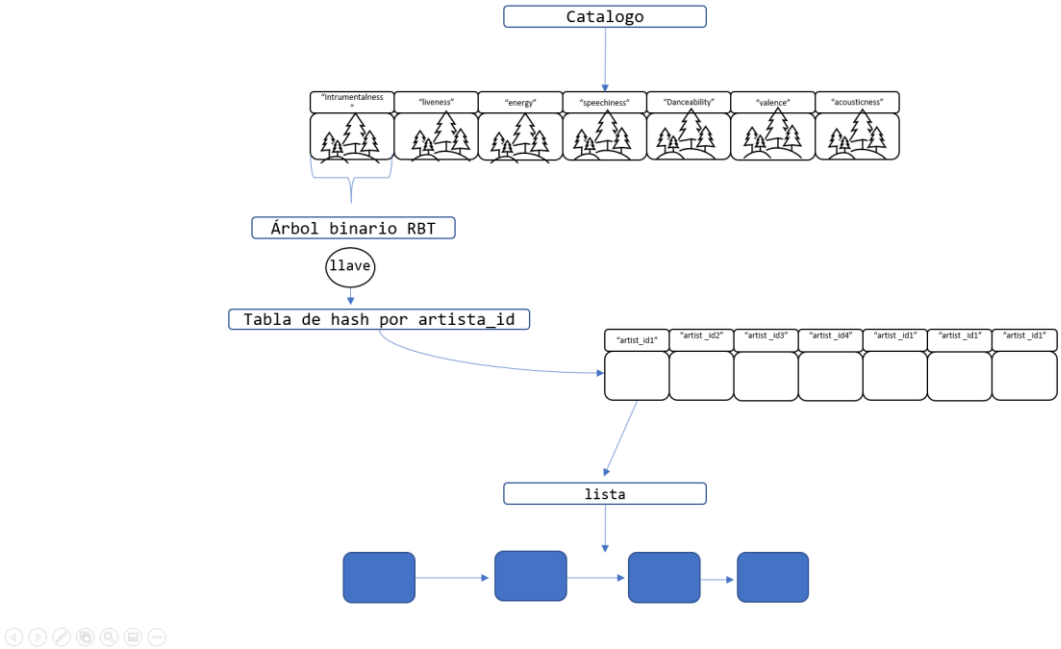
```
def conteo_llaves_unicas(lista):

    total = 0

    for char in lt.iterator(lista):

        total += 1

    return total
```

Esta función tiene una complejidad de O(N), N siendo la cantidad de llaves asociadas al valor dentro del rango solicitado.

Ya finalizado el requerimiento es correcto afirmar que la complejidad total es de:

O(2N), N siendo la cantidad de llaves asociadas al valor dentro del rango solicitado.

### Grafico:




## Requerimiento 2 (Estudiante A): Encontrar música para festejar

### Tiempos de ejecución y consumo de Datos:

Datos con los que se evaluó el requerimiento

| Rango Energy | Rango Danceabillity |
|---|---|
| 0.50-0.75 | 0.75-1 |

Respuesta:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

| Consumo de Datos[kB] | Tiempo de Ejecución [ms] |
|---|---|
| | |

### Complejidad del Requerimiento:

La primera parte de este requerimiento es bastante simple, unicamente se piden los datos necesarios para el funcionamiento de la aplicación (Energy y danceabillity) y se consultan los videos con estos valores por medio de la función range_values que va a ser uso de la función values () de la librería que se está usando:

```
def range_values(map,low,high):

    return om.values(map,low,high
```

Se hace uso de esta función 2 veces para buscar en el árbol de energy y el árbol de danceabillity las canciones que cumplan por separado las características exigidas. De manera que al final se obtendrá 2 listas.

Hasta este punto la complejidad es de O(2K)

Lo que sigue es transformar las listas adquiridas a 2 listas más simples en las que solo tengamos los IDs de cada una de las canciones con la respectiva característica. Para hacer esto se hace uso de la función:

```
def list_only_id(lista,coso):
```

Como este procedimiento se hace 2 veces, en este punto del programa se pude afirmar que la complejidad es de: O(N+G) y N siendo la cantidad de llaves en la lista con todos los elementos encontrados en el rango de la primera lista que esenergy, y G la segunda lista que es el danceabillity.

Lo siguiente es comparar las 2 listas de manera que podamos encontrar los tracks con las mismas características, para esto se usó la función:

```
def cmpare_two_list(list1,list2):

    new_list=lt.newList("ARRAY_LIST")

    if lt.size(list1)>lt.size(list2):

        lista_pequenia=list2

        lista_grande=list1

    else:

        lista_pequenia=list1

        lista_grande=list2

    for char in lt.iterator(lista_pequenia):

        if lt.isPresent(lista_grande,char)>0:

            lt.addLast(new_list,char)

    return new_list
```

En esta función lo que se está haciendo es que primero se elige la lista que se va a iterar, de preferencia la que tiene menos componentes, de manera el procedimiento se cumpla menos veces.

Esta función tiene una complejidad de: O(N) o O(G)

* N siendo la cantidad de llaves en la lista con todos los elementos encontrados en el rango de la primera lista que es energy, y G la segunda lista que es danceabillity.

La complejidad del requerimiento hasta este punto es de: O(2N+G) o O(N+2G)

Para finalizar se elige los datos que se va a presentar en la consola, para esto se creó una función que permitiera elegir los elementos de una forma random de manera que no se repita el número más de una vez.

```
def random_select(list,n):

    newlist=lt.newList(datastructure="ARRAY_LIST")
```

```
if lt.size(list)<n:
    n=lt.size(list)
else:
    n=n
i=0
while i<n:
    num1=random.randint(0,lt.size(list))
    elemento=lt.getElement(list,num1)



    while lt.isPresent(newlist,elemento)!=0:
        num1=random.randint(0,lt.size(list))
        elemento=lt.getElement(list,num1)
    lt.addLast(newlist,elemento)
    i+=1
return newlist
```

Ya finalizando es correcto afirmar que la complejidad de este programa es de: O(2N+G+H) o O(N+2G+H)

* N siendo la cantidad de llaves en la lista con todos los elementos encontrados en el rango de la primera lista que es energy, y G la segunda lista que es danceabillity. Y H siendo menor o igual a 5 elementos.

## Requerimiento 3 (Estudiante B): Encontrar música para estudiar

### Tiempos de ejecución y consumo de Datos:

Datos con los que se evaluó el requerimiento

| Rango Instrumentalness | Rango Tempo |
|---|---|
| 0-1 | 90-120 |

Respuesta:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

| Consumo de Datos[kB] | Tiempo de Ejecución [ms] |
|---|---|
| 3044 | 3062.5 |

## Complejidad del Requerimiento:

La primera parte de este requerimiento implica pedir los valores minimos y maximos de el tempo y el “intrumentalness” de la canción. Después de establecer estos valores se va a consultar los videos con estos valores por medio de la función range_values que va a ser uso de la función values () de la librería que se está usando:

```
def range_values(map,low,high):

    return om.values(map,low,high)
```

Se hace uso de la función 2 veces para buscar en el arbol de tempo y el arbol de instrumentalness, las caciones que cumplan por separado las caracteristicas exigidas. De manera que al final se obtendrá 2 listas.

Hasta este punto la complejidad es de O(2K)

Lo que sigue es transformar la lista adquirida a 2 lista más simple en la que solo tengamos los ids de cada una de las canciones con la característica respectiva. Para hacer esto se hace uso de la función:

```
def list_only_id(lista,coso):

     lista_nuea=lt.newList(datastructure="ARRAY_LIST")

     for char in lt.iterator(lista):

            elemento=lt.getElement(char["song"],1)

            lt.addLast(lista_nuea, elemento[coso])

     return lista_nuea
```

Como este procedimiento se hace 2 veces en este punto del programa se pude afirmar que la complejidad es de: O(N+G) y N siendo la cantidad de llaves en la lista con todos los elementos encontrados en el rango de la primera lista que es intrumentalness, y G la segunda lista que es el tempo.

Después lo que se hace es comparar las 2 listas de manera que podamos encontrar los tracks con las mismas características, para esto se usó la función:

```
def cmpare_two_list(list1,list2):

    new_list=lt.newList("ARRAY_LIST")

    if lt.size(list1)>lt.size(list2):

        lista_pequenia=list2

        lista_grande=list1

    else:
```

```
        lista_pequenia=list1
        lista_grande=list2
    for char in lt.iterator(lista_pequenia):
        if lt.isPresent(lista_grande,char)>0:
            lt.addLast(new_list,char)
    return new_list
```

En esta función lo que se está haciendo es que primero se elige la lista que se va a iterar, de preferencia la que tiene menos componentes, de manera el procedimiento se cumpla menos veces.

Esta función tiene una complejidad de: O(N) o O(G)

* N siendo la cantidad de llaves en la lista con todos los elementos encontrados en el rango de la primera lista que es intrumentalness, y G la segunda lista que es el tempo.

La complejidad del requerimiento hasta este punto es de: O(2N+G) o O(N+2G)

Para finalizar se elige los datos que se va a presentar en la consola, para esto se creó una función que permitiera elegir los elementos de una forma random de manera que no se repita el número más de una vez.

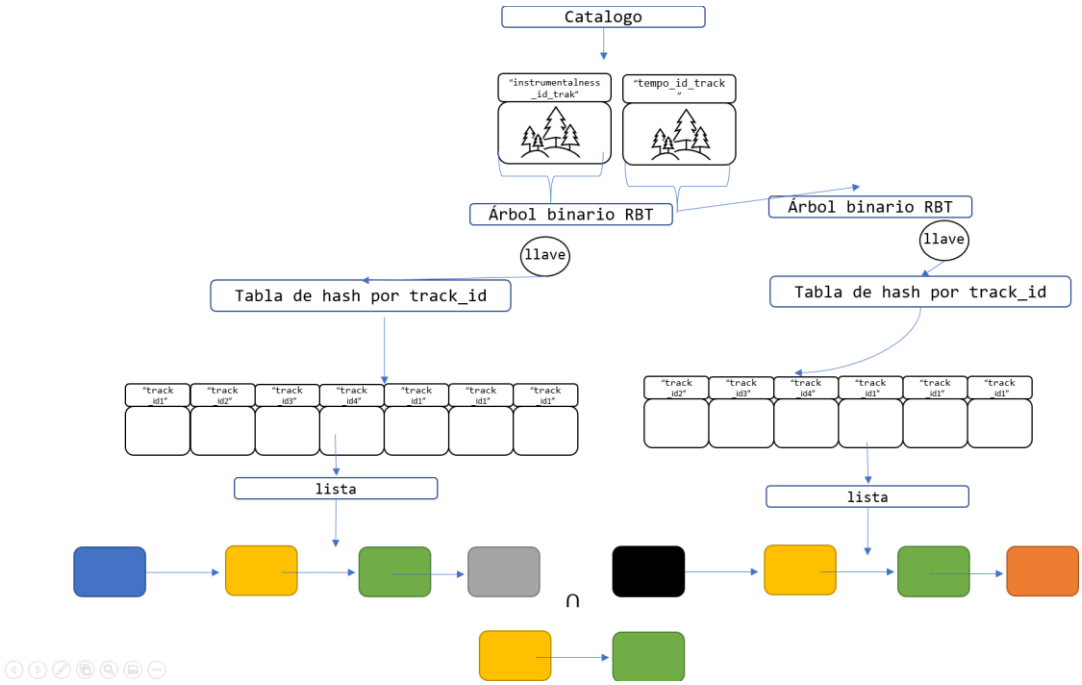
```
def random_select(list,n):
    newlist=lt.newList(datastructure="ARRAY_LIST")
    if lt.size(list)<n:
        n=lt.size(list)
    else:
        n=n
    i=0
    while i<n:
        num1=random.randint(0,lt.size(list))
        elemento=lt.getElement(list,num1)



        while lt.isPresent(newlist,elemento)!=0:
            num1=random.randint(0,lt.size(list))
            elemento=lt.getElement(list,num1)
        lt.addLast(newlist,elemento)
        i+=1
```

```
    return newlist
```

Ya finalizando es correcto afirmar que la complejidad de este programa es de: O(2N+G+H) o O(N+2G+H)

* N siendo la cantidad de llaves en la lista con todos los elementos encontrados en el rango de la primera lista que es intrumentalness, y G la segunda lista que es el tempo. Y H siendo menor o igual a 5 elementos.

Grafico:




## Requerimiento 4 (Grupal): Estudiar los géneros musicales

### Tiempos de ejecución y consumo de Datos:

Datos con los que se evaluó el requerimiento

| Generos Musicales a Buscar |
|---|
| Metal,Rock,Pop |

Respuesta:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

| Consumo de Datos[kB] | Tiempo de Ejecución [ms] |
|---|---|
| 28.26 | 453.125 |

## Complejidad del Requerimiento:

Para este requerimiento lo primero que se hiso fue pedir al usuario los géneros musicales que desea consultar. Para hacer la búsqueda lo primero que se hace es separar los nombres recibidos y empezar a hacer un recorrido sobre estos.

Dentro del ciclo se va a realizar una búsqueda en el mapa que contiene la información del género musical para finalmente, aplicar la información máxima y mínima en la búsqueda por rango.

```
lista_total=controller.lista_por_genero(catalog,char)
```

```
def lista_por_genero(catalog,n):
    minimo=float(get_someting_map(catalog["genero"],n,"BPM_minimo"))
    maximo=float(get_someting_map(catalog["genero"],n,"BPM_maximo"))
    tudo=(om.values(catalog["tempo"],minimo,maximo))
```

Como podemos ver en la función que se presenta se está haciendo una búsqueda en el mapa donde se guardan los géneros musicales. Esa primera parte tiene una complejidad de O(K).

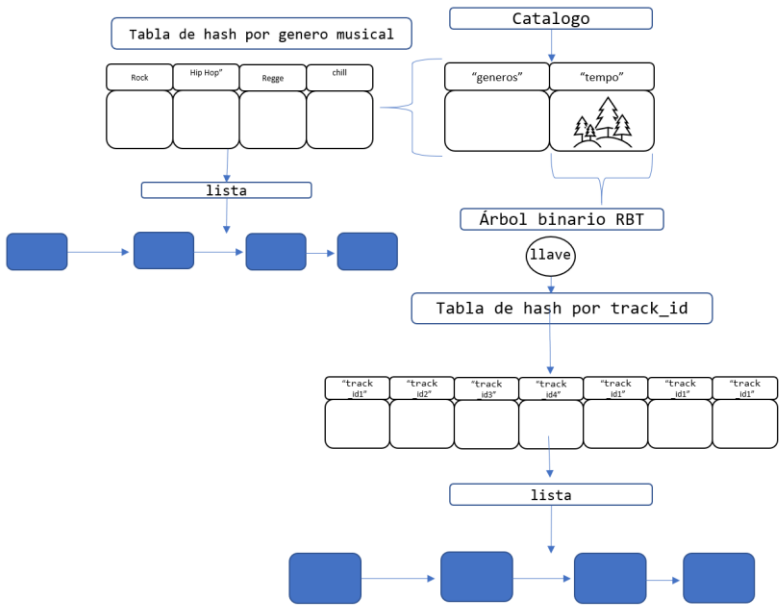
Después podemos ver que se está haciendo uso de la función om.values que recoge todas las listas dentro de un rango, está acción tiene una complejidad de O(K).

Lo que sigue después de adquirir las listas, es imprimir 10 elementos de manera aleatoria, este proceso tiene una complejidad de O(G), siendo G el número de artistas a imprimir.

La complejidad de este programa es de: O(N*G)

*G siendo el número de artistas a imprimir y N el número de géneros musicales a presentar.

## Grafico:



# Requerimiento 5 (Grupal): Indicar el género musical más escuchado en el tiempo

## Tiempos de ejecución y consumo de Datos:

Datos con los que se evaluó el requerimiento

| Hora minima | Hora maxima |
|---|---|
| 7:00 | 8:00 |

Respuesta:

La tabla a continuación presenta el tiempo que tardó el programa en cargar todos los datos en un catálogo:

| Consumo de Datos[kB] | Tiempo de Ejecución [ms] |
|---|---|
| 544 | 8250 |

## Complejidad del Requerimiento:

Para este requerimiento 3 lo primero que hay que hacer es recibir el rango de horas dadas por el usuario, si bien esta hora la hemos recibido desde un str es muy importante pasarla un formato con la hemos construido nuestro árbol.

```
def transform_hora(hora):
        time=hora.split(":")


        result = datetime.time(int(time[0]),int(time[1]))


        return result
```

Para esto vamos a usar la función datetime.time() esto tiene una complejidad de O(K).

### Hasta este punto la función tiene una complejidad de O(K)

Los segundo que va a pasar es que se va a usar la función om.values para hacer una búsqueda por un rango de horas. Este proceso nos va a devolver una tabla de hash separado por el nombre de cada uno del id_track. Esto tiene una complejidad de O(K)

```
def range_values(map,low,high):

    return om.values(map,low,high)
```

### Hasta este punto la complejidad del requerimiento es de O(K)

Lo siguiente que se va a presentar la cantidad de reproducciones por género musical, para hacerlo es importante pasar la lista encontrada anteriormente a una forma más sencilla de manera que usar la función lt.isPresent sea posible.

Para transformarla vamos a hacer uso de esta función list_only_id_repetition que va a recorrer toda lista encontrada anteriormente y se va conseguir la información especifica y se va a añadir al final de una lista nueva.

```
def list_only_id_repetición(lista,coso):
     lista_nuea=lt.newList(datastructure="ARRAY_LIST")
     for char in lt.iterator(lista):
            for elemento in lt.iterator(char["song"])
                lt.addLast(lista_nuea, elemento[coso])
     return lista_nuea
```

Este proceso tiene una complejidad de O(N)

*siendo N el número de elementos encontrados en la lista previamente consultada con la función om.values()

Complejidad cuando se buscan los géneros musicales en común:

Lo siguiente que va a pasar es que vamos a conseguir una lista con los géneros musicales en base a el tempo y separado por el track_id. Para esto hacemos una consulta en el diccionario y después usamos la función om.values, lo cual tiene una complejidad de O(K)

Y como más adelante vamos a comparar ambas listas entonces la lista en base al género musical será transformada a una lista más sencilla en donde solo usemos el id de los tracks. Esta parte del proceso tiene una complejidad O(T)

Al tener ambas listas vamos a hacer una comparación entre ambas listas de manera que solo sacaremos la intercepción entre ambas listas con la función a continuación:

```
def cmpare_two_list(list1,list2):
    new_list=lt.newList("ARRAY_LIST")
    if lt.size(list1)>lt.size(list2):
        lista_pequenia=list2
        lista_grande=list1
    else:
        lista_pequenia=list1
        lista_grande=list2
    for char in lt.iterator(lista_pequenia):
        if lt.isPresent(lista_grande,char)>0:
            lt.addLast(new_list,char)
    return new_list
```

Este proceso tiene una complejidad de O(F)

*F siendo la lista con menor cantidad de elementos (N o T)

Todo el proceso mencionado anteriormente va a ser repetido J veces siendo J el número de generos musicales a consultar.

La complejidad final va a ser de O(N+J*F*T)

*N es el número de elementos encontrados en el rango de horas, J la cantidad de géneros musicales a buscar, F la lista más pequeña entre N y T, y T la cantidad de elementos con cierto tempo.

Para finalizar cabe mencionar que la lista fue organizada por medio de la función merge sort que tiene una complejidad de (2log L)

La complejidad total es de: O(N+J*F*T+2log L)

*N es el número de elementos encontrados en el rango de horas, J la cantidad de géneros musicales a buscar, F la lista más pequeña entre N y T, y T la cantidad de elementos con cierto tempo. Y L la cantidad de elementos en común.

Complejidad cuando se busca y presentan 10 tracks:

```
def Top_tracks_hashtag(lista,catalog):
    i=1
    lista_retornar=lt.newList(datastructure="ARRAY_LIST" )
    while i<=10:
        h=random.randint(1,lt.size(lista))
        elemento=lt.getElement(lista,h)
        lis=mp.get(catalog["track_id_hashtrag"],str(elemento))
        num=0
        vader=0
        for char in lt.iterator(lis["value"]["song"]):
            lista_repeticiones=[]
            if char["hashtag"] not in lista_repeticiones:
                lista_repeticiones.append(char["hashtag"])
                if mp.contains(catalog["hashtag"],str(char["hashtag"]))==True:
                    info=mp.get(catalog["hashtag"],str(char["hashtag"]))
                    info=lt.getElement(info["value"]["song"],1)
                    if "."  in info["vader_avg"]:

                        vader+=float(info["vader_avg"])
                        num+=1

        if num==0:
```

```
            vader=vader/1
        else:
            vader=vader/num

        dato={"track_id":str(elemento),"num_hastags":str(num),"Vader":str(vader)}
        lt.addLast(lista_retornar,dato)
        i+=1
        lista_retornar=organizacion_req5(lista_retornar,lt.size(lista_retornar))

    return lista_retornar
```

Lo que está pasando en la función anterior es que se está eligiendo al azar un elemento de la lista con el género más referenciado el cual vamos a usar para hacer una búsqueda rápida en la tabla de hash "`track_id_hashtrag`", toda una lista con los elementos asociados al id.

Esta lista va a ser iterada se va a analizar los hashtags asociados al id, estos van a ser guardados de manera que no se contabilicen más de una vez. Al final se va a guardad cierta información seleccionada en una lista a aparte y al finalizar se va a hacer un merge sort de manera que podamos presentar la información de manera organizada.

Todo este proceso tiene una complejidad de O(10Q+2log(10Q))

*Q siendo la cantidad de elementos encontrados en la tabla de hash y 10 por que este proceso solo lo aremos10 veces.

Al final este requerimiento tiene una complejidad de: O(10Q+ N+J*F*T+2log L+2log(10Q))

*N es el número de elementos encontrados en el rango de horas, J la cantidad de géneros musicales a buscar, F la lista más pequeña entre N y T, y T la cantidad de elementos con cierto tempo. Y L la cantidad de elementos en común.

*Q siendo la cantidad de elementos encontrados en la tabla de hash y 10 por que este proceso solo lo aremos10 veces.