Estudiante A: Andrés Felipe Gómez García, af.gomezg1@uniandes.edu.co, 202021189
Estudiante B: Tomás Bedoya Calixto, t.bedoya@uniandes.edu.co, 202020689

ANÁLISIS DE COMPLEJIDAD RETO 1

Requerimiento 1:

1. Crear mapas auxiliares (artistas, cancines) y contador: O(1)
2. Acceder al mapa ordenado de la característica deseada: O(1)
3. Realizar función keys() sobre el mapa ordenado de la característica deseada: O(log(n) + n elementos retornados)
4. Extraer los elementos dentro de cada llave, adicionarlos a los diccionarios auxiliares correspondientes (artistas, canciones) de modo que no se tomen en cuenta repeticiones, y sumarle al contador para cada elemento, de modo que también se conozcan los eventos de escucha totales. Como en el absolutamente peor de los casos cada evento de escucha tiene una llave diferente, y además keys() retorna todas las llaves, se infiere que el número de llaves es igual al número de elementos, pero que dentro de cada llave hay un elemento único. En consecuencia, en el peor caso, el ciclo interno solo se corre una vez, por lo que no afecta la complejidad, y la complejidad mayor es dada por la complejidad de extraer todos los elementos del mapa ordenado: O(Nlog(N)).
5. Extraer los números deseados de la tupla en la que se retornan e imprimirlos O(N)

Complejidad total: O(Nlog(N))

Requerimiento 2: Estudiante B

1. Crear mapa auxiliar (canciones): O(1)
2. Calcular rangos de bailabilidad y energía de acuerdo a los valores del intervalo proporcionados por el usuario: O(1)
3. Comparar intervalos para decidir cual es más pequeño, ergo, cual es más probable que tenga un menor número de llaves. De acuerdo con el resultado, extraer el mapa ordenado de la categoría con el menor intervalo del catálogo: O(1)
4. Realizar keys() sobre el mapa ordenado extraído: O(log(n) + n elementos retornados)
5. Para cada llave retornada, se extraen sus elementos y, para cada elemento, se analiza si se encuentra en el intervalo de la otra categoría. En caso de cumplirse la condición, se adiciona al mapa auxiliar de canciones de modo que no se

tomen en cuenta repeticiones. Como en el absolutamente peor de los casos cada evento de escucha tiene una llave diferente, y además keys() retorna todas las llaves, se infiere que el número de llaves es igual al número de elementos, pero que dentro de cada llave hay un elemento único. En consecuencia, en el peor caso, el ciclo interno solo se corre una vez, por lo que no afecta la complejidad, y la complejidad mayor es dada por la complejidad de extraer todos los elementos del mapa ordenado: O(Nlog(N)).

6. Realizar keySet() sobre el mapa auxiliar para extraer todas sus llaves: O(N)
7. Por medio de random.randint, se extraen posiciones aleatorias de la lista dada por keySet, luego se extrae la llave correspondiente del mapa, y se imprimen las propiedades deseadas. Se repite el ciclo 5 veces: O(1)

Complejidad Total: O(Nlog(N))

Requerimiento 3: Estudiante A

1. Crear mapa auxiliar (canciones): O(1)
2. Calcular rangos de bailabilidad y energía de acuerdo a los valores del intervalo proporcionados por el usuario: O(1)
3. Comparar intervalos para decidir cual es más pequeño, ergo, cual es más probable que tenga un menor número de llaves. De acuerdo con el resultado, extraer el mapa ordenado de la categoría con el menor intervalo del catálogo: O(1)
4. Realizar keys() sobre el mapa ordenado extraído: O(log(n) + n elementos retornados)
5. Para cada llave retornada, se extraen sus elementos y, para cada elemento, se analiza si se encuentra en el intervalo de la otra categoría. En caso de cumplirse la condición, se adiciona al mapa auxiliar de canciones de modo que no se tomen en cuenta repeticiones. Como en el absolutamente peor de los casos cada evento de escucha tiene una llave diferente, y además keys() retorna todas las llaves, se infiere que el número de llaves es igual al número de elementos, pero que dentro de cada llave hay un elemento único. En consecuencia, en el peor caso, el ciclo interno solo se corre una vez, por lo que no afecta la complejidad, y la complejidad mayor es dada por la complejidad de extraer todos los elementos del mapa ordenado: O(Nlog(N)).
6. Realizar keySet() sobre el mapa auxiliar para extraer todas sus llaves: O(N)
7. Por medio de random.randint, se extraen posiciones aleatorias de la lista dada por keySet, luego se extrae la llave correspondiente del mapa, y se imprimen las propiedades deseadas. Se repite el ciclo 5 veces: O(1)

Complejidad Total: O(Nlog(N))

Requerimiento 4:

0. (opcional) Se le permite al usuario ingresar un nuevo género personalizado para la consulta: O(1)
1. Input de la lista de géneros que se desean tener en cuenta en la consulta. Se realiza split() sobre el string ingresado: O(size del string)
2. Se crea una lista auxiliar para guardar la información requerida por cada género evaluado: O(1)

Se toma el número de géneros, que puede ser variable, como G

3. Como en el absolutamente peor de los casos cada evento de escucha tiene una llave diferente, y además keys() retorna todas las llaves, se infiere que el número de llaves es igual al número de elementos, pero que dentro de cada llave hay un elemento único. En consecuencia, en el peor caso, el ciclo interno solo se corre una vez, por lo que no afecta la complejidad, y la complejidad mayor es dada por la complejidad de extraer todos los elementos del mapa ordenado: O(Nlog(N)).
4. La complejidad anterior es sin tomar en cuenta el ciclo inicial con el propósito de simplificar su cálculo. Por lo tanto, toca añadirle la influencia que tiene el ciclo principal. Como el proceso se realiza por cada género, es decir G veces, y, adicionalmente, se realiza un keys() igualmente por cada género con base en sus rangos, la complejidad total del ciclo triple sería O(GNlog(N) + G[log(N) + n de elementos a retornar]). No obstante, considerando que G es una variable que, en términos prácticos, tendrá un valor despreciable a comparación de N, concluimos que, debido al marco de la posibilidad de que G tienda al número de N, a pesar de que esta posibilidad es altamente improbable, creemos prudente tomar G en cuenta como una variable, aunque sabemos que en la práctica se puede considerar que su aporte se asemejará al de una constante, lo que implica que será despreciable. Por consiguiente, la complejidad se toma O(GNlog(N)).
5. Se imprime la información requerida por cada género de parámetro para la consulta: O(G)

Complejidad General: O(GNlog(N))

Requerimiento 5:

1. Se realiza keys() sobre el diccionario de horas para obtener las llaves correspondientes al rango ingresado por el usuario: O(log(N) + n elementos retornados)
2. Se realiza keySet() sobre el diccionario de géneros tomando en cuenta todos aquellos que el usuario haya podido crear previamente: O(G)

3. Se crea un mapa auxiliar (mapaGeneros) y se introducen todos los géneros como llaves: O(G)
6. Se recorren las llaves obtenidas por medio de keys() en el paso 1. Como en el absolutamente peor de los casos cada evento de escucha tiene una llave diferente, y además keys() retorna todas las llaves, se infiere que el número de llaves es igual al número de elementos, pero que dentro de cada llave hay un elemento único. En consecuencia, en el peor caso, el ciclo interno solo se corre una vez, por lo que no afecta la complejidad, y la complejidad mayor es dada por la complejidad de extraer todos los elementos del mapa ordenado (Nlog(N)). No obstante, como se hace una comparación por genero para cada elemento, el cual se adjunta a los géneros que le corresponden en el diccionario de géneros, resulta entonces que: O(GNlog(N))
7. Se crea lista por arreglo auxiliar (listaTuplas): O(1)
8. Se ingresa a la lista auxiliar el genero y el tamaño de la lista correspondiente a cada genero (guardada en el diccionario de géneros) en forma de tupla: O(G)
9. Se realiza mergeSort sobre la lista auziliar: O(Glog(G))
10. A partir de el ordenamiento de la lista, se extrae el genero con mayor cantidad de eventos de escucha del diccionario de géneros: O(1)
11. Se crea otra lista por arreglo auxiliar (lst) y se transfieren los elementos de la lista del genero con más eventos de escucha: O(N)
12. Se realiza un mergeSort sobre lst: O(Nlog(N))
13. Se crean 2 listas enlazadas (repetidos, tracksVader), un contador y una variable posición: O(1)
14. Hay un while que tiene como condiciones que el contador, que se incrementa cada vez que se encuentra un track ID nuevo con un promedio de Vader averages diferente de 0, sea menor que 10, y que la variable pos sea menor o igual al tamaño de la lista lst. Por consiguiente, en el peor de los casos se considera que todos los track IDs de la lista son iguales, por lo que al revisar repeticiones por medio de isPresent() en la lista repetidos, la condición nunca es verdad, el contador nunca incrementa y se recorre toda la lista lst (solo se adjuntaría, en este peor caso, un valor a tracksvader). Cabe aclarar que el promedio que se calcula en la primera iteración (usualmente se calcula para cada nuevo track ID) consiste en sumar todos los valores vader averages únicos, es decir, uno por cada hashtag único que halla sido detectado para dicho track ID en la carga de datos, y dividir entre el número de dichos hashtags: O(N)
15. Se suman e imprimen el total de eventos de escucha sumando los valores numéricos correspondientes a cada género en listaTuplas: O(G)
16. Se recorre la listaTuplas y se imprimen los eventos de escucha por género: O(G)
17. Se recorre tracksVader y se extrae el promedio de los Vader averages para todos los elementos de la lista (10 max): O(1)

Complejidad General: O(GNlogN)