Reto 1

Estudiante A: Jose Vicente Vargas Panesso (201815601)

jv.vargas@uniandes.edu.co

Estudiante B: Daniel Reales (201822265)

da.reales@uniandes.edu.co

I. Análisis de Complejidad del Requerimiento 1

La primera función que se invoca al llamar la función del requerimiento 1 es la función *category_name_id*. Este recibe por parámetro el catalogo y el nombre de una categoría y retorna el id de la misma. La implementación de esta función en cuestión se realiza iterando sobre la lista de categorías que es pasada por parámetro. Por esta razón, esta sub-rutina tiene complejidad $O(n_c)$, donde n_c representa la cantidad de categorías. Es importante notar que si se asume que la cantidad de categorías es fija para los videos a analizar (sin importar cuantos se quieran estudiar), entonces hay independencia entre el número de videos que se analizan n y el número de categorías que se deben recorrer n_c.

Posteriormente, se realiza el filtro acorde al país y a la categoría seleccionadas por el usuario. Para esta tarea se utiliza la función *videosPorcategoriaPais*. Esta función retorna una lista nueva únicamente con los videos que satisfacen los filtros. Para realizar esta tarea, se implementa un recorrido por la lista. Por esta razón, la complejidad de esta segunda sub-rutina es $O(n)$.

Posteriormente, una vez realizado el filtro se procede a organizar los datos mediante el algoritmo de ordenamiento Merge Sort. Aunque este algoritmo en comparación con otros algoritmos recursivos como Quick Sort tiene mayor complejidad espacial, se seleccionó esta algoritmo debido a que la complejidad temporal es sustancialmente menor. Para Quick Sort, el peor caso tiene una complejidad temporal de $O(n^2_{categoria})$ mientras que Merge Sort tiene una complejidad temporal para este mismo caso de $O(n_{categoria} \log(n_{categoria}))$. En este sentido, esta tercera subrutina tendría esta última complejidad temporal. Respecto a esta última complejidad es importante notar que $n_{categoria}$ en este caso se refiere no a la longitud completa del catálogo, sino a la longitud de la lista una vez esta fue filtrada utilizando los criterios de país y categoría.

Finalmente, la última subrutina de este requerimiento es la creación de una lista que contiene el número de videos que el usuario solicitó. Dado que se realiza mediante un ciclo, este tiene complejidad $O(n_{usuario})$ donde $n_{usuario}$ se refiere a la longitud de elementos que el usuario quiere desplegar.

En conclusión, sumando las complejidades de las subrutinas vemos que el algoritmo que implementa el requerimiento 1 tiene una complejidad de:

$$O(n) + O(n_c) + O(n_{categoria} \log(n_{categoria})) + O(n_{usuario})$$

Por esta razón, tomando en cuenta que el término preponderante es $O(n_{categoria}\log(n_{categoria}))$, concluimos que el algoritmo tiene una complejidad temporal aproximada de $O(nlog(n))$

II. Análisis de Complejidad del Requerimiento 2 (Estudiante A):

La primera sub-rutina que se invoca al llamar la función del requerimiento 2 es *dividirPais.* Esta función retorna una nueva lista filtrando acorde al país ingresado por parámetro. La complejidad de esta es $O(n)$ debido a que la implementación se realiza mediante un ciclo sobre todos los videos del catalogo para identificar aquellos que coincidan con el país de interés. De esta manera, tras completar su llamado, se genera una lista de menor tamaño (igual a n_{pais}) que después es pasada a la subrutina *agregarTrending* cuya complejidad temporal se describe a continuación. Al interior de la función antes mencionada se realiza una organización utilizando el algoritmo *Merge Sort.* Este bloque de código tiene una complejidad $O(n\log(n))$ sobre n_{pais}. Sin embargo, posteriormente se procede a realizar un ciclo sobre la lista ingresada por parámetro. Esto agrega una complejidad temporal de $O(n_{pais})$. Por estas dos razones se concluye que la complejidad de la subrutina *agregarTrending* es de $O(\ n + nlog(n)) = O(n(1 + \log(n))$. Sin embargo, debido a que en el límite el término dominante corresponde a $nlog(n)$ se determina la complejidad de esta como $O(nlog(n))$.

Después de crear esta sublista filtrada y con la información del número de días trending para cada video, se procede a realizar el ordenamiento final de ella utilizando el algoritmo recursivo *Merge Sort*. Como se mencionó anteriormente, este paso tiene una complejidad $O(nlog(n))$ siendo $n = n_{trending}$. El tamaño de la lista en este caso es menor que n_{pais} debido a que al llamar a la subrutina *agregarTrending* sólo se encuentra una entrada por video. Finalmente, las instrucciones para crear la información a retornar todas tienen complejidad $O(1)$.

En conclusión, considerando todas las subrutinas que el Requerimiento 2 invoca, vemos que la complejidad es de:

$$O(n) + O(n_{pais}(1 + \log(n_{pais})) + O(n_{trending}\log(n_{trending})) + \ O(1)$$

Tomando los términos que dominan la expresión concluimos que el algoritmo tiene una complejidad aproximada de $O(nlog(n))$.

III. Análisis de Complejidad del Requerimiento 3 (Estudiante B):

La primera subrutina que invoca el requerimiento 3 es la función *category_name_id*. Como se analizó en el primer literal, esta tiene una complejidad temporal $O(n_c)$ donde n_c corresponde al número de elementos que tiene la lista que contiene la información de las categorías. Posteriormente, se invoca la subrutina *filter_by_category_ratio* cuya finalidad es retornar una lista filtrada acorde a un id de categoría ingresado por parámetro. La implementación de esta se realiza mediante un recorrido sobre todos los videos para identificar a los que cumplan con el filtro

indicado. Por esta razón su complejidad también será de $O(n)$, pero esta vez n corresponde al total de elementos de la lista de videos.

Después de realizar el filtro, se invoca a la función *agregarTrending* sobre la lista filtrada. Como se analizó en el literal anterior, esta subrutina tiene una complejidad temporal $O(n_{categoria}\log(n_{categoria}))$ donde $n_{categoria}$ corresponde al número de videos que se encuentran en la lista resultante al aplicar la función *filter_by_category_ratio.*

Finalmente, una vez se agrega la información del número de días trending, se realiza un ordenamiento basado en este criterio utilizando el algoritmo *Merge Sort*. Por consiguiente, la complejidad temporal se ve incrementada en $O(n_{categoria}\log(n_{categoria}))$. Debido a que los pasos restantes, concernientes a la construcción de la información que será retornada tienen una complejidad de $O(1)$ concluimos que la complejidad total del algoritmo que implementa el requerimiento 3 es de:

$$O(n_c) + O(n) + 2 \cdot O(n_{categoria}\log(n_{categoria})) + O(1)$$

Debido a que el término dominante corresponde a $O(n_{categoria}\log(n_{categoria}))$, concluimos que la complejidad temporal del algoritmo que implementa el requerimiento 3 es de $O(nlog(n))$.

I. <u>Análisis de Complejidad del Requerimiento 4 (Estudiante B):</u>

La primera subrutina que es llamada al ejecutar el requerimiento es *separarPaisTags.* Esta función retorna una nueva lista filtrada en la que únicamente se encuentran los videos cuyo país y tag son acordes a lo especificado por parámetro. La implementación de esta se realiza mediante un recorrido por la lista. Por esta razón, su complejidad temporal es de $O(n)$.

La segunda subrutina es el algoritmo de organización *Merge Sort*. Como se ha discutido anteriormente este tiene una complejidad temporan de $O(n_{pais/tags}\log(n_{pais/tags}))$, donde $n_{pais/tags}$ corresponde al número de elementos que contiene la lista una vez se aplica el filtro.

Posteriormente, se realiza un llamado a la subrutina *eliminarRepetidos* cuya implementación mediante un recorrido sobre la lista causa que tenga una complejidad temporal $O(n_{pais/tags})$. Por último, se construye una lista con los $n_{usuario}$ elementos que se van a desplegar. Para ello se utiliza un ciclo while. Esto agrega una complejidad temporal $O(n_{usuario})$ al algoritmo.

En conclusión, dadas las subrutinas utilizadas en el requerimiento, el algoritmo que implementa el requerimiento 4 tiene una complejidad dada por:

$$O(n) + O\left(n_{pais/tags} \log\left(n_{pais/tags}\right)\right) + O\left(n_{pais/tags}\right) + O(n_{usuario})$$

Debido a que el termino preponderante corresponde a $O(n_{pais/tags} \log\left(n_{pais/tags}\right))$ concluimos que la complejidad temporal del algoritmo que implementa el requerimiento 4 es $O\left(nlog(n)\right)$.

ANEXOS

```
def requerimiento1(catalog, category_name, country, n, tipo_orga
nizacion, prueba, size):
    opciones = {"0": selectionsort.sort, "1": insertionsort.sort
,
                "2": shellsort.sort, "3": quicksort.sort, "4": m
ergesort.sort}
    category_id = category_name_id(catalog,category_name)
    if prueba == False:
        nueva_lista = videosPorcategoriaPais(
            catalog["videos"], category_id, country)
    else:
        nueva_lista = lt.subList(catalog['videos'], 0, size)
    # organizacion
    t_start = time.time_ns()
    organizada = opciones[tipo_organizacion](nueva_lista, compar
acionLikes)
    t_end = time.time_ns()

    if prueba:
        organizada = videosPorcategoriaPais(organizada, category
_id, country)

    t_total = t_end - t_start
    lista_final = lt.newList("ARRAY_LIST")
    for j in range(1, n+1):
        if j <= lt.size(organizada) and j > 0:
            lt.addLast(lista_final, lt.getElement(organizada, j)
)
        else:
            pass
    return lista_final, t_total
```

```python
def requerimiento2(catalog, country):
    '''
    Retorna una lista que contiene los datos del video
    con más dias trending, cuya relación likes/dislikes es positi
va
    y acorde a un país determinado.
    '''
    lista_pais = dividirPais(catalog, country)
    lista_por_titulo = agregarTrending(lista_pais)
    nueva_lista = mergesort.sort(lista_por_titulo, comparacionDia
sRatioPais)
    elemento = lt.getElement(nueva_lista, 1)
    ratio = str(obtenerRatio(elemento))
    dias = elemento["dias_trending"]
    datos = lt.newList("ARRAY_LIST")
    lt.addLast(datos, elemento["title"])
    lt.addLast(datos, elemento["channel_title"])
    lt.addLast(datos, elemento["country"])
    lt.addLast(datos, ratio)
    lt.addLast(datos, dias)
    return datos
```

```python
def requerimiento3(catalog, category_name):
    '''
    Retorna una tupla con la información del video
    con más dias trending, cuya relación likes/dislikes es positi
va
    y acorde a una categoria determinada.
    '''
    id = category_name_id(catalog, category_name)
    filtered_list = filter_by_category_ratio(catalog, id)
    trending_list = agregarTrending(filtered_list)
    ordered_trending = mergesort.sort(trending_list, cmp_by_trend
ing)
    top_video = lt.getElement(ordered_trending, lt.size(ordered_t
rending))
    data = (top_video['title'], top_video['channel_title'],
            id, obtenerRatio(top_video), top_video['dias_trending
'])
    return data
```

```
def requerimiento4(catalog, country, tag, n):
    '''
    Retorna una lista con los n videos con más comentarios según
el país
    y el tag especificados por parámetro.
    '''
    lista_modificada = separarPaisTags(catalog, country, tag)
    organizada = mergesort.sort(lista_modificada, comparacionCome
ntarios)
    organizada = eliminarRepetidos(organizada)
    entregar = lt.newList("ARRAY_LIST")
    i = 1
    while i <= n:
        if lt.size(organizada) >= i:
            lt.addLast(entregar, lt.getElement(organizada, i))
            i += 1
        else:
            i = n+1
    return entregar
```