NOMBRE	CÓDIGO	CORREO UNIANDES
Lindsay Vanessa Pinto Morato	202023138	l.pintom@uniandes.edu.co
Maicol Yojan Antonio Rincon	202027329	m.antonio@uniandes.edu.co

ANÁLISIS DE COMPLEJIDAD DE LOS REQUERIMIENTOS

Requerimiento 1:

Función:

```
def sortVideos(catalog, n, country, category):
```

Antes de utilizar esta función, previamente se ha creado un catálogo (catalog) mediante la función newCatalog en la cual se decidió utilizar un Array List debido a la mejor complejidad, y por tanto, menores tiempos de ejecución mostrados en las pruebas realizadas en laboratorios previos.

Ya teniendo esto, nuestra función sortVideos recibe dicho catálogo(catalog), el número de resultados que se desean (n), el país (country) y la categoría (category).

		COMPLEJIDAD
1	Posteriormente hace un **primer ciclo** para hacer un filtro por categoría buscada de acuerdo al id de la categoría	O(n)
2	Utilizando la función de comparación cmpVideosByCountry y usando el algoritmo Merge Sort se hace un ordenamiento por el nombre del país.	O(nlog(n))
3	Después de esto, realiza un **segundo y tercer ciclo** cuya función es dejar solamente el país de consulta. Se realiza entonces comparación determinando si el país buscado está o no en la lista.	O(2n)
3*	En el tercer ciclo, al tener un Break, la complejidad cambia abruptamente interrumpiendo el ciclo y saliendo del mismo.	O(log(n)).
4	Al final de esto se crea una sublista con la información de filtrado requerido hasta el momento. (Por país)	O(1)
5	Utilizando la función de comparación cmpVideosByCategory y usando el algoritmo Merge Sort se hace un ordenamiento por la categoría.	O(nlog(n))
6	Al igual que en el paso 3, se crean un **cuarto y quinto** ciclo utilizados para extraer la categoría buscada de acuerdo con el "category_id"	O(2n)

7	Al final de esto se crea una sublista con la información de filtrado requerido hasta el momento. (Por categoría)	O(1)
8	Utilizando la función de comparación cmpVideosByViews y usando el algoritmo Merge Sort se hace un ordenamiento por las vistas.	O(nlog(n))
9	Al final de esto se crea una sublista con la información de filtrado requerido hasta el momento. (Por vistas)	O(1)
10	Al final se retorna la sublista	O(1)

En esta función se utilizó el algoritmo MergeSort pues fue el que mostró mejor comportamiento para los ordenamientos en un ArrayList en las pruebas realizadas anteriormente. Aun así, las librerías de los demás ordenamientos fueron importadas para ser utilizadas a futuro si se desean cambiar los requerimientos del algoritmo.

La complejidad de este requerimiento es O(5n), lo cual quiere decir que su complejidad es del orden de o(n) en el peor de sus casos, esto debido a los ciclos que presenta para su implementación.

El algoritmo podría haberse realizado de una manera más eficiente, no obstante, se realizó dicha implementación debido a la facilidad de comprensión de la misma además de la necesidad de implementación de búsqueda a través de las llaves y su coincidencia con los valores de entrada del usuario reduciendo el error en utilización de mayúsculas y minúsculas con la función. Lower().

Requerimiento 2:

Función:

```
def getTrendingVideoByCountry(catalog, country):
```

Implementado por: Lindsay Pinto Morato

Este algoritmo presenta una estructura similar al inmediatamente anterior y también utiliza el catalog creado en la función newCatalog de la cual se habló en el requerimiento pasado. De igual manera, realiza un ordenamiento basado en el método MergeSort por las mismas razones enunciadas en el requerimiento 1.

		COMPLEJIDAD
1	Posteriormente hace un **primer y segundo ciclo** para hacer un filtro por el país buscado	O(n) -> primer ciclo
1*	En el segundo ciclo, al tener un Break, la complejidad cambia abruptamente interrumpiendo el ciclo y saliendo del mismo.	O(log(n)). -> segundo ciclo
2	Al final de esto se crea una sublista con la información de filtrado requerido hasta el momento. (Por país)	O(1)
3	El **tercer ciclo** extrae los nombres de los videos de acuerdo a la función de comparación. Esto permite saber si un video se ha repetido e ir actualizando la cuenta que se lleva por video. Al	O(n)

	final debe retornar el video que más veces apareció en el listado, lo que quiere decir que estuvo en tendencia	

La complejidad de este requerimiento es O(3n), lo cual quiere decir que su complejidad es del orden de o(n) en el peor de sus casos, esto debido a los ciclos que presenta para su implementación.

Sin duda alguna, el algoritmo podría haberse realizado de una manera más eficiente, no obstante, teniendo en cuenta que debía buscarse el video que más apareciera en el archivo, y por ende con más días en tendencia para determinado país, nos pareció adecuada esta implementación.

Requerimiento 3:

Función:

```
def getTrendingByCategory(catalog, category):
```

La función recibe como parámetros el catalogo anteriormente creado, y el id de una categoría que se le pide al usuario.

		COMPLEJIDAD
1	Busca la categoría ingresada por el usuario, en el catálogo en la sección de categorías	O(n)
2	Utilizando la función de comparación cmpVideosByCategory y usando el algoritmo Merge Sort se hace un ordenamiento por categorías	O(log(n))
3	Con la lista ordenada, se busca el nombre de la categoría ingresada por el usuario, y arroja el índice donde se encuentra la primera de ellas	O(n)
3*	Con ese índice se realiza una segunda búsqueda, esta para saber donde termina esa categoría, dado que como la lista esta ordenada debe estar todas una detrás de la otra, con esto tenemos el índice donde termina dicha categoría.	O(n).
4	Se crea una sublista con los índices que encontramos, de esta manera solo queda filtrada la categoría que escogió el usuario	O(1)
5	Después con la sublista filtrada por la categoría se busca cual de ellos estuvo más días en tendencia	O(n)
6	Al final se retorna la sublista	O(1)

La complejidad del Algoritmo es O(5n) dado que entra 5 veces en un ciclo, pero todos ellos son independientes del otro

Requerimiento 3:

Función:

```
def getTrendingByCategory(catalog, category):
```

La función recibe como parámetros el catalogo anteriormente creado, y el id de una categoría que se le pide al usuario.

		COMPLEJIDAD
1	Busca la categoría ingresada por el usuario, en el catálogo en la sección de categorías	O(n)
2	Utilizando la función de comparación cmpVideosByCategory y usando el algoritmo Merge Sort se hace un ordenamiento por categorías	O(log(n))
3	Con la lista ordenada, se busca el nombre de la categoría ingresada por el usuario, y arroja el índice donde se encuentra la primera de ellas	O(n)
3*	Con ese índice se realiza una segunda búsqueda, esta para saber donde termina esa categoría, dado que como la lista esta ordenada debe estar todas una detrás de la otra, con esto tenemos el índice donde termina dicha categoría.	O(n).
4	Se crea una sublista con los índices que encontramos, de esta manera solo queda filtrada la categoría que escogió el usuario	O(1)
5	Después con la sublista filtrada por la categoría se busca cual de ellos estuvo más días en tendencia	O(n)
6	Al final se retorna la sublista	O(1)

La complejidad del Algoritmo es O(5n) dado que entra 5 veces en un ciclo, pero todos ellos son independientes del otro

Requerimiento 4:

Función:

```
def getTrendingByLikes(catalog,country,tag):
```

La función recibe como parámetros el catálogo anteriormente creado, el país y el tag que se le piden al usuario para realizar la búsqueda.

		COMPLEJIDAD
1	Utilizando la función de comparación cmpVideosByCountry y usando el algoritmo Merge Sort se hace un ordenamiento por países	O(log(n))

2	Con la lista ordenada, se busca el nombre del país ingresado por el usuario, y arroja el índice donde se encuentra el primero	O(n)
3	Con ese índice se realiza una segunda búsqueda, esta para saber dónde termina ese país, dado que como la lista esta ordenada debe estar todas una detrás de la otra, con esto tenemos el índice donde termina dicho país	O(n).
4	Se crea una sublista con los índices que encontramos, de esta manera solo queda filtrada los países que concuerdan con el ingresado por el usuario	O(1)
5	Después con esa sublista, se organizan por tags para realizar lo mismo en los pasos anteriores	O(1)
6	Una vez organizado se busca el índice menor donde aparece por primera el tag ingresado por el usuario	O(n)
7	Después se busca el índice mayor, donde terminan los tags	O(n)
8	Se crea una sublista filtrado con los 2 índices	O(1)
9	Se organiza la lista por la cantidad de likes	O(1)
10	Retorna los 3 videos con mas likes	O(1)

La complejidad del Algoritmo es O(4n) dado que entra 4 veces en un ciclo, pero todos ellos son independientes del otro