ANÁLISIS DEL RETO

Emmanuel Bolteada Manzo, 202226494, e.bolteada@uniandes.edu.co

Jesús Misael Reséndiz Cruz, 202226497, j.resendiz@uniandes.edu.co

María Fernanda De la Hoz, 202214512, m.delah@uniandes.edu.co

Requerimiento 1

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	-Lista ordenada de datos -Fecha de inicio del rango -Fecha final del rango
Salidas	El número total de películas presentes en el periodo. Las tres primeras y tres últimas películas de dicha lista, en donde cada elemento contendrá la siguiente información:La fecha de lanzamiento (release_year). El nombre de la película (title).La duración (duration). La plataforma de distribución en línea o streaming. El nombre del director(director). Los actores (cast).
Implementado (Sí/No)	Si. Grupal.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
La lista es ordenada con mergesort, por año de lanzamiento de los datos. (sortByReleaseYear)	O(n log n), dada que esta es la complejidad que posee este algoritmo de ordenamiento.
Recorrido de toda la lista de datos para realizar comparaciones entre los datos de entrada y el año de lanzamiento de cada elemento. Para así agregar a la lista de retorno, los datos que coincidan satisfactoriamente con las condiciones. (getMoviesByReleaseYear)	O(n), siendo n la cantidad de elementos que posee la lista de datos.
Devolución de la respuesta	O(1), ya que solo se itera una vez
TOTAL	***O(n log n) + O(n) = O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

Se estableció que para el ordenamiento de los datos el algoritmo utilizado para todas las pruebas es MergeSort, pues muestra un mejor rendimiento en términos de eficiencia. Se realizó una variación en los porcentajes de la muestra usados durante las pruebas.

Como variables de prueba, se utilizaron los valores de:

Año inicial=1920

Año final=1999

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	1,8 GHz Dual-Core Intel Core i5
Memoria RAM (GB)	8
Sistema Operativo	macOS

Entrada, porcentaje de la muestra	**Tiempo (ms)**
0.50%	20.864
5%	77.636
10%	145.737
20%	245.073
30%	375.932
50%	717.095
80%	1168.666
100%	1657.741

De la complejidad espacial de la función se puede establecer que, a partir de la complejidad que posee el algoritmo de ordenamiento MergeSorte de o(n), de la creación de variables a partir de la invocación a funciones del tipo lt.getElement(), tendríamos una complejidad de o(1). Por su parte, la función lt.newList(), al crear una nueva lista tiene un o(n).

Graficas

Análisis

A partir de los resultados obtenidos, se puede identificar que la función además de cumplir con los requerimientos pedidos, está modelado bajo un formato de lista Arraylist, el cual para el manejo grandes cantidades de datos presenta un mejor desempeño. De igual manera, fue identificado el algoritmo de ordenamiento MergeSort como el de mejor rendimiento en cuestión del manejo de tiempo.

Acerca de la complejidad tanto espacial como de tiempo, se presentan algoritmos que se mantienen en el estándar de rendimiento bueno. Al no llegarse a ver ordenes de complejidad mayores que linearítmicos.

Requerimiento 2

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Fecha inicial del periodo (con formato “%B %d, %Y"). Fecha final del periodo (con formato “%B %d, %Y").
Salidas	• El número total de películas presentes en el periodo. • Las tres primeras y tres últimas películas de dicha lista, en donde cada elemento contendrá la siguiente información: • El nombre de la película (title).

	• La fecha de adición a la plataforma (date_added). • La duración (duration). • La fecha de lanzamiento (release_year). • La plataforma de distribución en línea o streaming. • El nombre del director (director). • Los actores (cast).
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ordenando con merge sort por fecha de adición. (sortByDateAdded)	**O(n log n)**, esto por la literatura de la matemática en la que se describe el algoritmo.
Encontrar el contenido para el periodo dado (getShowsByPeriod)	**O(n),** donde n es el tamaño de la lista a recorrer para buscar las fechas en ese periodo de tiempo.
Formato para imprimir los datos.	**O(1)**, cuanto máximo la cantidad de registros a mostrar siempre va a ser de 6 muestras.
TOTAL	***O(n log n) + O(n) = O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

El ordenamiento de la lista siempre se realizó con Merge Sort ya que este fue con el que se obtuvo mejores resultados en el laboratorio, así como con la estructura de datos Array List. La variable a tomar en cuenta para las pruebas entonces fue el tamaño de los datos con el que se prueba el requerimiento.

Para seguir una misma tendencia de congruencia en la entrada de los datos, el periodo de fechas en los que se buscará es: 2018-02-11 a 2020-05-23

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

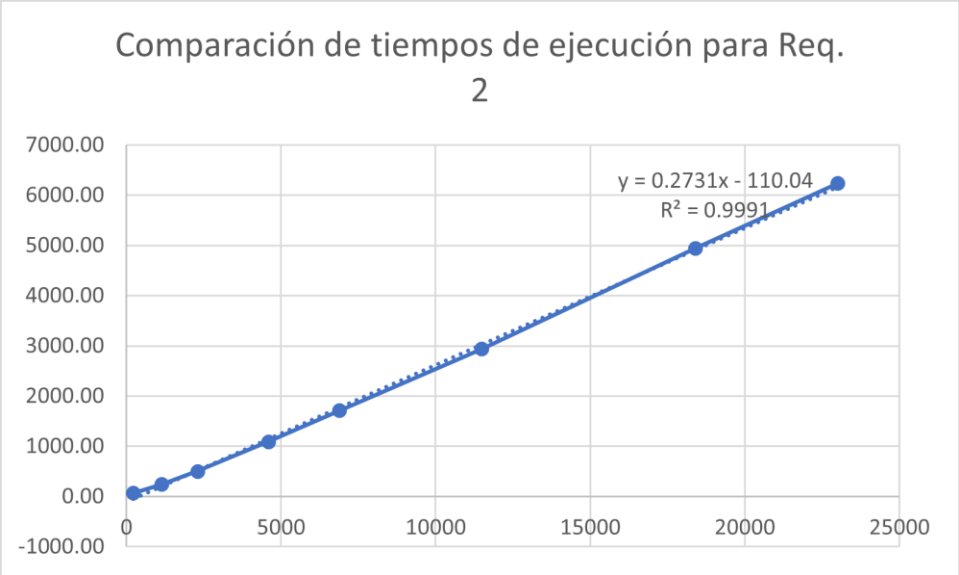
Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Memoria RAM (GB)	16
Sistema Operativo	Windows 10

Entrada, porcentaje de la muestra	Tiempo (ms)
0.50%	64.858
5%	239.495
10%	500.904
20%	1091.198
30%	1712.620
50%	2933.916
80%	4933.651
100%	6231.510

Con respecto a la complejidad espacial de todo el proceso del requerimiento, tenemos que lo considerable es el algoritmo de ordenamiento merge sort que da una complejidad de O(n). Para el resto de los pasos son variables temporales que no dependen de la cantidad de datos y siempre son fijas, por lo que no se toman a la hora de dar el total. Siendo así **O(n)**.

Graficas



Análisis

Los resultados obtenidos en las pruebas coinciden con el análisis de complejidad hecho del algoritmo. Se utilizó la combinación Array List y Merge Sort pues de acuerdo a los resultados en los laboratorios fueron los óptimos para el manejo de datos. Para este requerimiento en particular se utilizó la librería datetime que facilitó la comparación de fechas y perimitió que la complejidad no aumentara, quedando una complejidad lineal O(n).

Requerimiento 3

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El nombre del actor dentro del elenco (ej.: Scott Adkins, Adam Sandler, etc.).
Salidas	• El número total de películas de ese actor. • El número total de programas de ese actor. • Los tres primeros y tres últimos registros de dicha lista, en donde cada elemento contendrá la siguiente información: o El nombre de la película o programa (title). o Fecha de lanzamiento (release_year). o El nombre del director (director). o La plataforma de distribución en línea o streaming. o La duración (duration). o Los actores (cast). o El país de producción (country). o El género (listed_in). o Descripción (description)
Implementado (Sí/No)	Si. Jesús Misael Reséndiz Cruz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ordenando con merge sort por título, fecha de lanzamiento y duración (sortByName)	O(n log n), esto por la literatura de la matemática en la que se describe el algoritmo.
Encontrar el contenido para el actor dado (getContentByActor)	O(n*m), donde n es la cantidad de registros de peliculas y programas, y m es la cantidad de caracteres que existe en cada campo de Cast de cada registro. Para la mayoría de casos m no cambia su cantidad, por lo que podemos dejarla como constante. Aproximando a **O(n).**
Formato para imprimir los datos.	O(1), cuanto máximo la cantidad de registros a mostrar siempre va a ser de 6 muestras.
TOTAL	***O(n log n) + O(n) = O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

Para las búsquedas siempre se realizaron con Merge Sort, sin embargo, los tiempos de ejecución siempre cambian dado a que se ingresan distintas cantidades de registros.

Para seguir una misma tendencia de congruencia en la entrada de los datos, el actor que se escogerá en todas las pruebas de ejecución será: **Nicki Richards**

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

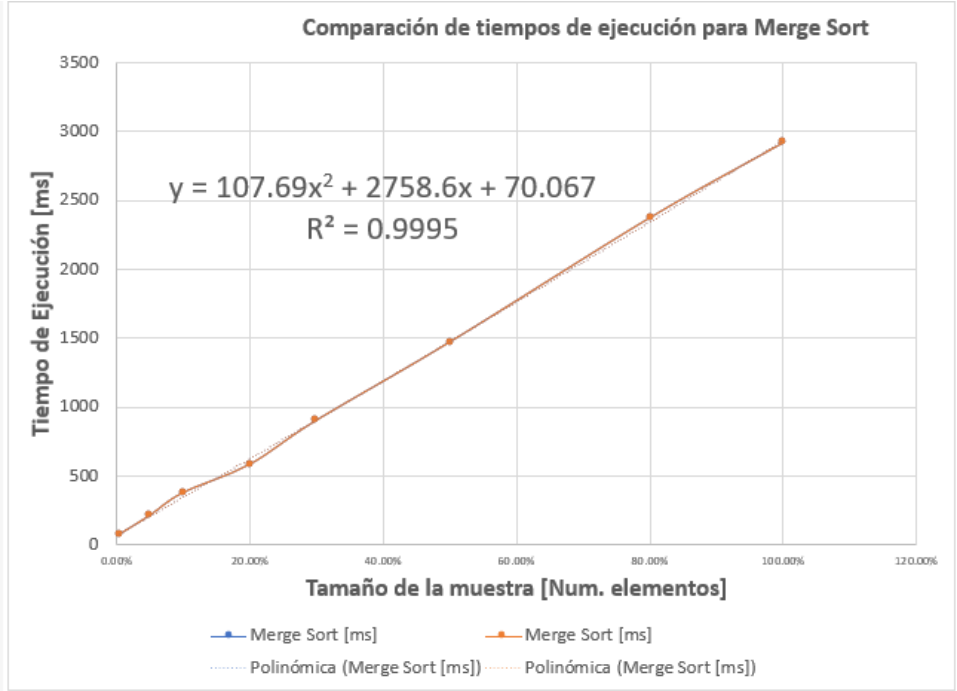
Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	AMD A8-7410 APU with AMD Radeon R5 Graphics 2.20 GHz
Memoria RAM (GB)	8
Sistema Operativo	Windows 10

Entrada, porcentaje de la muestra	**Tiempo (ms)**
0.50%	79.015
5%	214.058
10%	380.981
20%	585.637
30%	905.449
50%	1473.050
80%	2375.834
100%	2920.996

Con respecto a la complejidad espacial de todo el proceso del requerimiento, tenemos que lo considerable es el algoritmo de ordenamiento merge sort que da una complejidad de O(n). Para el resto de los pasos son variables temporales que no dependen de la cantidad de datos y siempre son fijas, por lo que no se toman a la hora de dar el total. Siendo así **O(n)**.

Graficas



Análisis

Ante los laboratorios previos para realizar por completo el reto 1, nos preparó para identificar con gran rigidez qué tipo de estructura y ordenamiento seleccionar para realizar este requerimiento. En este caso se optó por guardar los registros en una estructura de tipo Array_list y un ordenamiento de tipo merge ya que funciona bastante bien para la mayoría de los casos en los que nos encontrábamos. Al pasar por todos los pasos del proceso algorítmico del requerimiento, tenemos un tiempo de complejidad temporal de **O(n log n) + O(n)** y para el espacial tenemos **O(n).**

Requerimiento 4

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	-Lista ordenada de datos -Género que se desea buscar
Salidas	El número total de películas de ese género. El número total de programas de ese género. Los tres primeros y tres últimos registros de dicha lista, en donde cada elemento contendrá la siguiente información El nombre de la película o programa (title). Fecha de lanzamiento (release_year).

	El nombre del director (director). La plataforma de distribución en línea o streaming. La duración (duration). Los actores (cast). El país de producción (country). El género (listed_in). Descripción (description).
Implementado (Sí/No)	Si. Maria Fernanda De la Hoz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
La lista es ordenada con Mergesort, por nombres. (sortByName2)	O(n log n), dada que esta es la complejidad que posee este algoritmo de ordenamiento.
Recorrido de toda la lista de datos para realizar operaciones de comparación con cada elemento. (getContentByGenero)	O(n), siendo n la cantidad de elementos que posee la lista de datos.
Recorrido de los datos almacenados en una variable, luego de haber hecho .split(), para luego ser comparados con la variable de género ingresada	O(mn),siendo m la cantidad de elementos obtenidos a partir del .spli() y n la cantidad de elementos de la lista
TOTAL	***O(n log n) + O(n)+O(mn) = O(mn)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

Se estableció que para el ordenamiento de los datos el algoritmo utilizado para todas las pruebas es MergeSort, pues muestra un mejor rendimiento en términos de eficiencia. Se realizó una variación en los porcentajes de la muestra usados durante las pruebas.

Como variables de prueba, se utilizaron los valores de:

genero=1920

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina

Procesadores	1,8 GHz Dual-Core Intel Core i5
Memoria RAM (GB)	8
Sistema Operativo	macOS

Entrada, porcentaje de la muestra	Tiempo (ms)
0.50%	36.580
5%	76.196
10%	115.062
20%	200.338
30%	303.454
50%	506.594
80%	872.106
100%	1184.317

De la complejidad espacial de la función se puede establecer que, a partir de la complejidad que posee el algoritmo de ordenamiento MergeSorte de o(n), de la creación de variables a partir de la invocación a funciones del tipo lt.getElement(), tendríamos una complejidad de o(1). Por su parte, la función lt.newList(), al crear una nueva lista tiene un o(n).

Graficas

Análisis

A partir de los resultados obtenidos, se puede identificar que la función además de cumplir con los requerimientos pedidos, está modelado bajo un formato de lista Arraylist, el cual para el manejo grandes cantidades de datos presenta un mejor desempeño. De igual manera, fue identificado el algoritmo de ordenamiento MergeSort como el de mejor rendimiento en cuestión del manejo de tiempo.

Acerca de la complejidad tanto espacial como de tiempo, se presentan algoritmos que se mantienen en el estándar de rendimiento bueno. Al no llegarse a ver ordenes de complejidad mayores que linearítmicos.

Requerimiento 5

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El país (ej.: Argentina, Colombia, etc.).

Salidas	• El número total de películas producidos en ese país.
	• El número total de programas producidos en ese país.
	• Los tres primeros y tres últimos registros de dicha lista, en donde cada elemento contendrá la siguiente información:
	  ○ El nombre de la película o programa (title).
	  ○ Fecha de lanzamiento (release_year).
	  ○ El nombre del director (director).
	  ○ La plataforma de distribución en línea o streaming.
	  ○ La duración (duration).
	  ○ Los actores (cast).
	  ○ El país de producción (country).
	  ○ El género (listed_in).
	  ○ Descripción (description).

Implementado (Sí/No)	Si. Emmanuel Bolteada Manzo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ordenando con merge sort por título. (sortByName2)	**O(n log n)**, esto por la literatura de la matemática en la que se describe el algoritmo.
Encontrar el contenido para el periodo dado (getShowsByCountry)	**O(n),** donde n es el tamaño de la lista a recorrer para buscar el país coincidente.
Formato para imprimir los datos.	**O(1)**, cuanto máximo la cantidad de registros a mostrar siempre va a ser de 6 muestras.
TOTAL	***O(n log n) + O(n) = O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

El ordenamiento de la lista siempre se realizó con Merge Sort ya que este fue con el que se obtuvo mejores resultados en el laboratorio, así como con la estructura de datos Array List. La variable a tomar en cuenta para las pruebas entonces fue el tamaño de los datos con el que se prueba el requerimiento.

Para seguir una misma tendencia de congruencia en la entrada de los datos, el periodo de fechas en los que se buscará es: Canada.

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

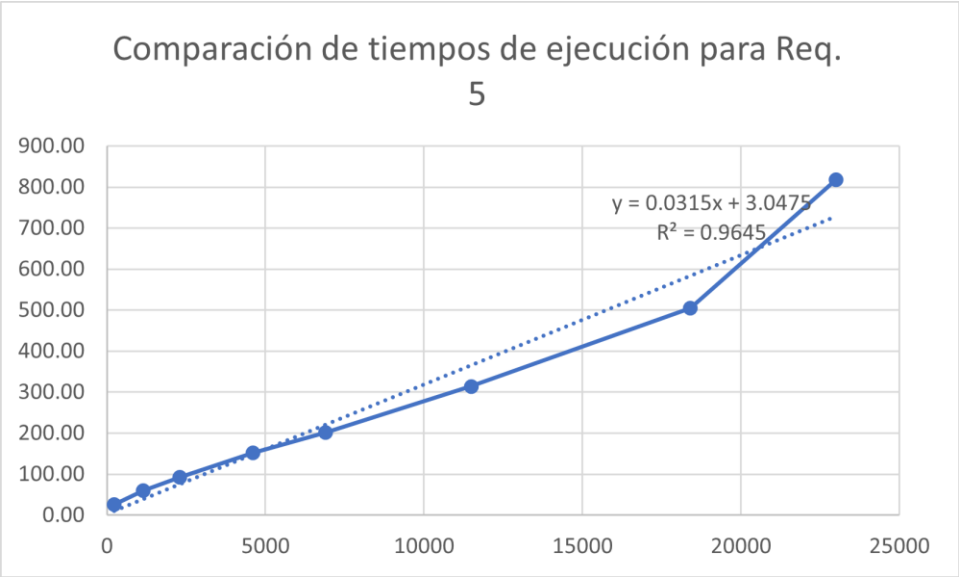
Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Memoria RAM (GB)	16
Sistema Operativo	Windows 10

Entrada, porcentaje de la muestra	**Tiempo (ms)**
0.50%	26.595
5%	60.241
10%	92.208
20%	152.082
30%	201.401
50%	314.181
80%	505.103
100%	817.710

Con respecto a la complejidad espacial de todo el proceso del requerimiento, tenemos que lo considerable es el algoritmo de ordenamiento merge sort que da una complejidad de O(n). Para el resto de los pasos son variables temporales que no dependen de la cantidad de datos y siempre son fijas, por lo que no se toman a la hora de dar el total. Siendo así **O(n)**.

Graficas



Análisis

Los resultados obtenidos en las pruebas coinciden con el análisis de complejidad hecho del algoritmo. Se utilizó la combinación Array List y Merge Sort pues de acuerdo a los resultados en los laboratorios fueron los óptimos para el manejo de datos. Para este requerimiento en particular se utilizó la comparación de países, lo cual es una operación lineal y perimitió que la complejidad no aumentara, quedando una complejidad lineal O(n).

Requerimiento 6

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El director (ej.: Steven Spilberg, Woody Allen, etc.)
Salidas	• El número total de películas y programas dirigidos por ese director. • El número total de películas y programas por cada género (listed_in). • En caso de ser muy numerosos, imprimir los tres primeros y tres últimos registros de dicha lista con la siguiente información: o El genero con el que está relacionado el contenido (listed_in). o el contenido de las producciones que están asociadas a ese género. • El número total de películas y programas por plataforma. • Los tres primeros y tres últimos registros de dicha lista, en donde cada elemento contendrá la siguiente información: o La fecha de lanzamiento (release_year).

	o El nombre de la película o programa (title). o La duración (duration). o La plataforma de distribución en línea o streaming. o El nombre del director (director). o Los actores (cast). o El país de producción (country) o El género (listed_in) o Descripción (description).
Implementado (Sí/No)	Si. En equipo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ordenando con algoritmo merge sort por fecha de lanzamiento, título y duración (sortByReleaseYear)	**O(n log n)**, esto por la literatura de la matemática en la que se describe el algoritmo.
Encontrar el contenido para el director dado. (getContentByDirector)	O(n*m), donde n es la cantidad de registros de películas y programas, y m es la cantidad de caracteres que existe en cada campo de Director de cada registro. Para la mayoría de los casos m no cambia su cantidad, por lo que podemos dejarla como constante. Aproximando a **O(n)**.
Formato para imprimir los datos:	
El número total de películas y programas por plataforma.	O(m), donde m es la cantidad de registros encontrados en donde se involucra dicho director. Es difícil que haya muchos registros para cada director, esta cantidad suele ser muy pequeña con respecto a la variable n, por lo que podemos aproximar a O(1).
El número total de películas y programas por cada género (listed_in)	O(m), donde m es la cantidad de registros encontrados en donde se involucra dicho director. Por lo anterior descrito, aproximamos a O(1).
Los registros en los que se involucra el director, mostrando información completa.	O(1), cuanto máximo la cantidad de registros a mostrar siempre va a ser de 6 muestras.
TOTAL	***O(n log n) + O(n)= O(n)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

Para las búsquedas siempre se realizaron con Merge Sort, sin embargo, los tiempos de ejecución siempre cambian dado a que se ingresan distintas cantidades de registros.

Para seguir una misma tendencia de congruencia en la entrada de los datos, el director que se escogerá en todas las pruebas de ejecución será: **David Jung**

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

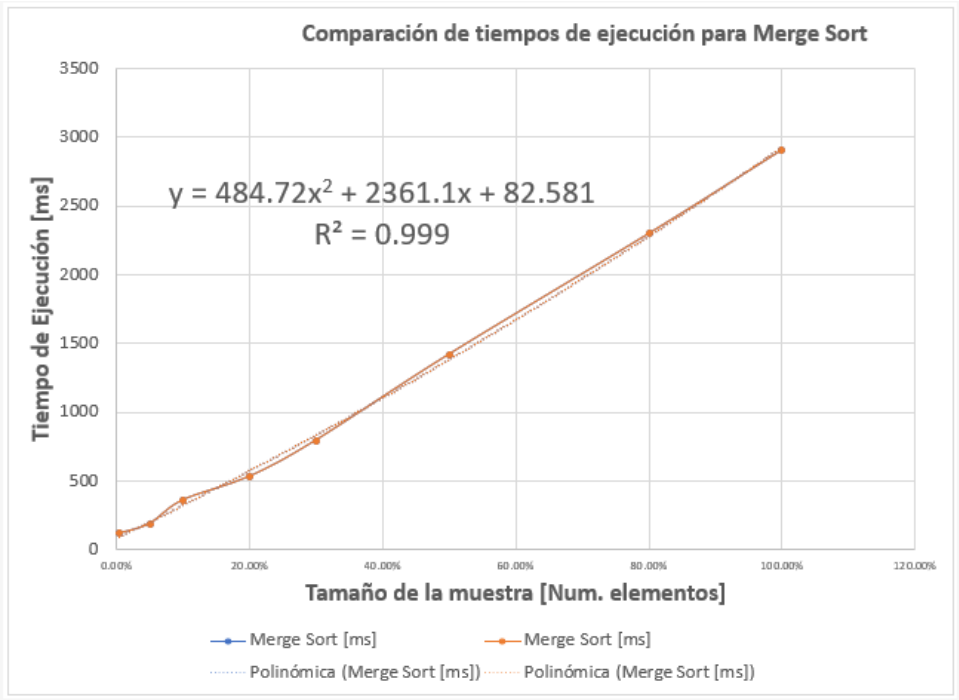
Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	AMD A8-7410 APU with AMD Radeon R5 Graphics 2.20 GHz
Memoria RAM (GB)	8
Sistema Operativo	Windows 10

Entrada, porcentaje de la muestra	**Tiempo (ms)**
0.50%	117.265
5%	186.402
10%	359.024
20%	531.775
30%	795.732
50%	1422.803
80%	2302.163
100%	2911.569

Con respecto a la complejidad espacial de todo el proceso del requerimiento, tenemos que lo considerable es el algoritmo de ordenamiento merge sort que da una complejidad de O(n). Para el resto de los pasos son variables temporales que no dependen de la cantidad de datos y siempre son fijas, por lo que no se toman a la hora de dar el total. Siendo así **O(n)**.

Graficas



Análisis

Ante los laboratorios previos para realizar por completo el reto 1, nos preparó para identificar con gran rigidez qué tipo de estructura y ordenamiento seleccionar para realizar este requerimiento. En este caso se optó por guardar los registros en una estructura de tipo Array_list y un ordenamiento de tipo merge ya que funciona bastante bien para la mayoría de los casos en los que nos encontrábamos. Al pasar por todos los pasos del proceso algorítmico del requerimiento, tenemos un tiempo de complejidad temporal de **O(n log n) + O(n)** y para el espacial tenemos **O(n).**

Requerimiento 7

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	-Lista ordenada de datos -Número de datos que se desea ver en el top
Salidas	El grupo de N géneros organizados por el número de películas y programas

	De cada género se debe presentar la siguiente información El nombre del género.El número total de películas y programas por plataforma El número de películas.El número de programas.
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrer cada elemento de la lista de datos	O(n), siendo n la cantidad de elementos que posee la lista de datos.
Recorrido de los datos almacenados en una variable, luego de haber hecho .split(), para luego ser comparados con la variable de género ingresada	O(mn),siendo m la cantidad de elementos obtenidos a partir del .split() y n la cantidad de elementos de la lista.
Recorrido de los datos almacenados en una variable.	O(pn) siendo p la cantidad de elementos en la nueva variable, y n la cantidad de elementos de la lista de datos
TOTAL	***O(n) + O(mn)+O(pn) = O(mn)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

Se realizó una variación en los porcentajes de la muestra usados durante las pruebas.

Como variables de prueba, se utilizaron los valores de:

n=5

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	1,8 GHz Dual-Core Intel Core i5
Memoria RAM (GB)	8
Sistema Operativo	macOS

Entrada, porcentaje de la muestra	**Tiempo (ms)**

0.50%	96.895
5%	690.204
10%	2364.724
20%	4134.085
30%	6983.110
50%	11620.614
80%	18515.108
100%	23074.072

La creación de variables a partir de la invocación a funciones del tipo lt.getElement(), tendríamos una complejidad de o(n), debido a que inicialmente tiene una complejidad de o(1), pero al encontrarse iterando en un ciclo n veces, es de n(o(1)). Por su parte, la función lt.newList(), al crear una nueva lista tiene un o(n).

Graficas

Análisis

A partir de los resultados obtenidos, se puede identificar que la función además de cumplir con los requerimientos pedidos está modelado bajo un formato de lista Arraylist, el cual para el manejo grandes cantidades de datos presenta un mejor desempeño. A diferencia de las otras funciones, en esta era necesario realizar diversos recorridos de las listas que contenían los datos, para poder modificar los contadores, pero para acceder a estas variables se hacía necesario el paso por cada elemento de la lista. Lo cual no es óptimo en términos de espacio ni de tiempo. Característica que se puede notar en el incremento de los órdenes de crecimiento con respecto a anteriores requerimientos.

Requerimiento 8

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El número (N) de actores a identificar (ej.: TOP 3, 5, 10 o 20)
Salidas	• El grupo de N actores organizados por su número de participaciones. • De cada actor se debe presentar la siguiente información. o El nombre del actor. o El número total de películas y programas por plataforma de streaming. o El número de películas en las que ha participado. o El número de programas de televisión en los que ha participado. o El listado de directores con los que ha trabajado, ordenados alfabéticamente. o El listado de actores con los que ha trabajado, ordenados alfabéticamente. o El género (listed_in) con mayor número de películas o programas en el que ha participado.
Implementado (Sí/No)	Si. En equipo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrer cada elemento de la lista de datos	O(n), siendo n la cantidad de elementos que posee la lista de datos.
Recorrido de los datos almacenados en una variable, luego de haber hecho .split(), para luego ser comparados con la variable de actor ingresada	O(mn),siendo m la cantidad de elementos obtenidos a partir del .split() y n la cantidad de elementos de la lista.
Recorrido de los datos almacenados en una variable.	O(pn) siendo p la cantidad de elementos en la nueva variable, y n la cantidad de elementos de la lista de datos
TOTAL	***O(n) + O(mn)+O(pn) = O(mn)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados librerías, computadores donde se ejecutan las pruebas, entre otros).

Se realizó una variación en los porcentajes de la muestra usados durante las pruebas.

Como variables de prueba, se utilizaron los valores de:

n=5

Para estimar el tiempo ejecutado utilizamos la librería Timer, en el que tomamos el tiempo actual en el que inicia el algoritmo de procesamiento y hacemos la diferencia con el tiempo al momento de terminar dicho procesamiento.

Especificaciones de la máquina para ejecutar las pruebas de rendimiento:

	Máquina
Procesadores	1,8 GHz Dual-Core Intel Core i5
Memoria RAM (GB)	8
Sistema Operativo	macOS

Entrada, porcentaje de la muestra	**Tiempo (ms)**
0.50%	98.67
5%	694.564
10%	2364.724
20%	4454.045
30%	7283.310
50%	11740.834
80%	18815.308
100%	23098.099

La creación de variables a partir de la invocación a funciones del tipo lt.getElement(), tendríamos una complejidad de o(n), debido a que inicialmente tiene una complejidad de o(1), pero al encontrarse iterando en un ciclo n veces, es de n(o(1)). Por su parte, la función lt.newList(), al crear una nueva lista tiene un o(n).

Graficas

Análisis

A partir de los resultados obtenidos, se puede identificar que la función además de cumplir con los requerimientos pedidos está modelado bajo un formato de lista Arraylist, el cual para el manejo grandes cantidades de datos presenta un mejor desempeño. A diferencia de las otras funciones, en esta era necesario realizar diversos recorridos de las listas que contenían los datos, para poder modificar los contadores, pero para acceder a estas variables se hacía necesario el paso por cada elemento de la lista. Lo cual no es óptimo en términos de espacio ni de tiempo. Característica que se puede notar en el incremento de los órdenes de crecimiento con respecto a anteriores requerimientos.