ANÁLISIS DEL RETO

Wilmer Manuel Arévalo González, 202214720, w.arevalo@uniandes.edu.co

Laura Valentina Cerón Pulgarín, 202214973, l.ceronp@uniandes.edu.co

Mario Velásquez Semanate, 202020502, m.velasquezs@uniandes.edu.co

Requerimiento 1

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Año especifico.
Salidas	Los registros de las películas estrenadas en el año específico que ingrese el usuario. Además, se tabula con la siguiente información: Type, release year, title, duration, stream_service, director, cast.
Implementado (Sí/No)	Sí. Realizado por: Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar cada título en el rango de años a la lista	O(n)
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
TOTAL	***O(n^(3/2))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash del año a la lista	O(1)

Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para "encontrar" uno de sus valores solo se necesita tener la llave, en este caso el año, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

Entrada	**Tiempo (s)**
Catalog, 1920	2.19 ms
Catalog, 1950	1.95 ms
Catalog, 1975	1.85 ms

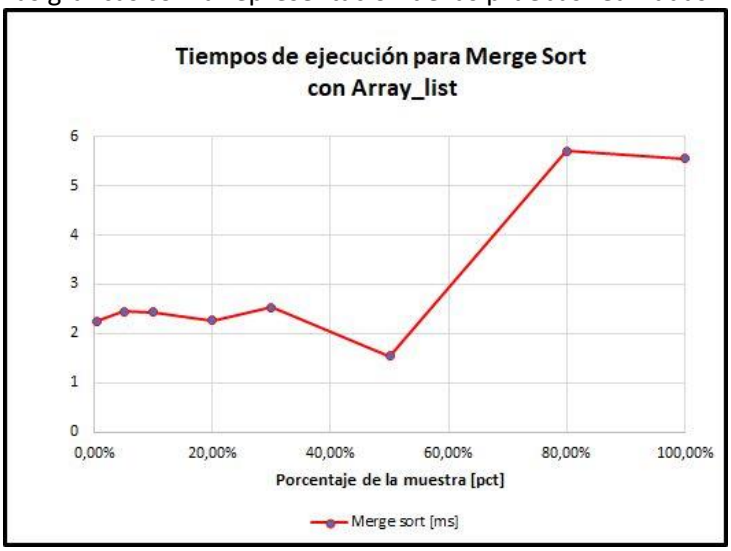
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	2.25 ms
5,00%	1148	2.45 ms
10,00%	2298	2.44 ms
20,00%	4598	2.27 ms
30,00%	689	2.53 ms
50,00%	11498	1.54 ms
80,00%	18397	5.71 ms
100,00%	22998	5.55 ms

Graficas reto 2

Las gráficas con la representación de las pruebas realizadas.



Graficas reto 1

Requerimiento 2

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Año especifico
Salidas	Los registros de los programas de televisión agregados a las plataformas en el año especifico ingresado por el usuario. Además, se tabula con la siguiente información: Type, date_added, title, duration, release_year, stream_service, director, cast.
Implementado (Sí/No)	Sí. Realizado por: Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: Iterar y agregar cada título en el rango de años a la lista	O(n)
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
TOTAL	***O(n^(3/2))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash del año a la lista	O(1)
Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para “encontrar” uno de sus valores solo se necesita tener la llave, en este caso el año, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

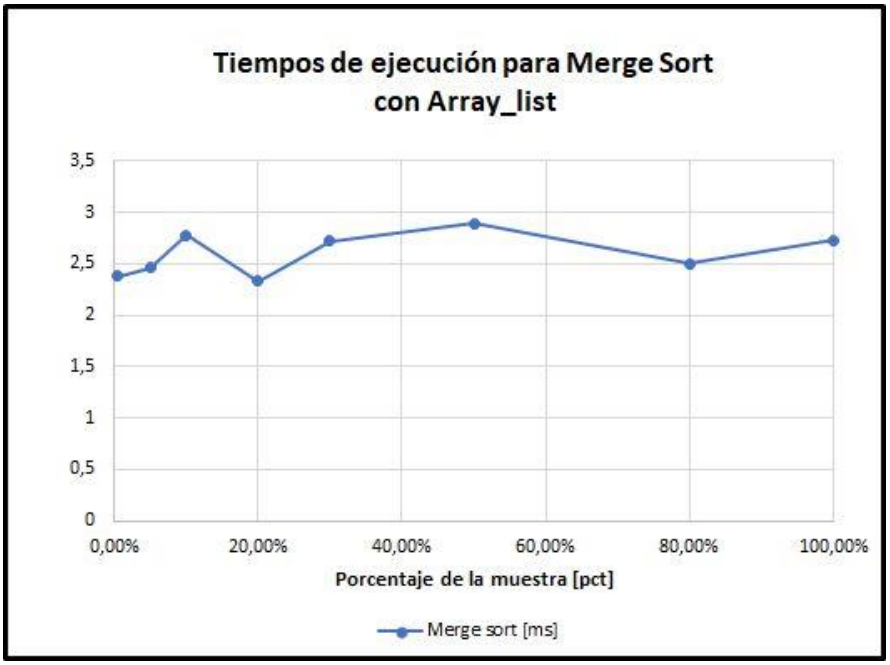
Entrada	**Tiempo (s)**
Catalog, January 07, 1920	2.92 ms
Catalog, January 07, 1950	2.28 ms
Catalog, January 07, 1975	2.89 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	2.38 ms
5,00%	1148	2.46 ms
10,00%	2298	2.78 ms
20,00%	4598	2.33 ms
30,00%	6898	2.72 ms
50,00%	11498	2.89 ms
80,00%	18397	2.5 ms
100,00%	22998	2.73 ms

Graficas reto 2



Graficas reto 1

Requerimiento 3

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Actor.
Salidas	Los registros de las películas y los programas de televisión en los que participa el actor ingresado por el usuario ordenado alfabéticamente. Además, se tabula con la siguiente información: 1. **Primera tabla:** Type*(movie y Tv Show)* y Count*(la cantidad de veces en las que aparece el actor, ya sea en movie o Tv Show)* 2. **Segunda tabla**: release_year, title, duration, director, stream_service, type, cast, country, rating, listed_in, description.
Implementado (Sí/No)	Sí. Realizado por: Laura Valentina Cerón

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: Iterar y agregar cada título que contenga al actor/actriz de interés en su cast	O(n)
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
TOTAL	***O(n^(3/2))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash correspondiente al nombre del actor/actriz a la lista	O(1)
Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para “encontrar” uno de sus valores solo se necesita tener la llave,

en este caso el nombre del actor/actriz, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

Reto 1:

Entrada	**Tiempo (s)**
Catalog, Tom Cruise	3.02 ms
Catalog, Sissy Spacek	6.61 ms
Catalog, Gerard Butler	7.21 ms

Tablas de datos

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	7.91 ms
5,00%	1148	10.07 ms
10,00%	2298	8.96 ms
20,00%	4598	8.79 ms
30,00%	6898	16.36 ms
50,00%	11498	26.25 ms
80,00%	18397	22.51 ms
100,00%	22998	28.37 ms

Reto 2:

Entrada	**Tiempo (s)**
Catalog, Tom Cruise	3.02 ms

Catalog, Sissy Spacek	6.61 ms
Catalog, Gerard Butler	7.21 ms

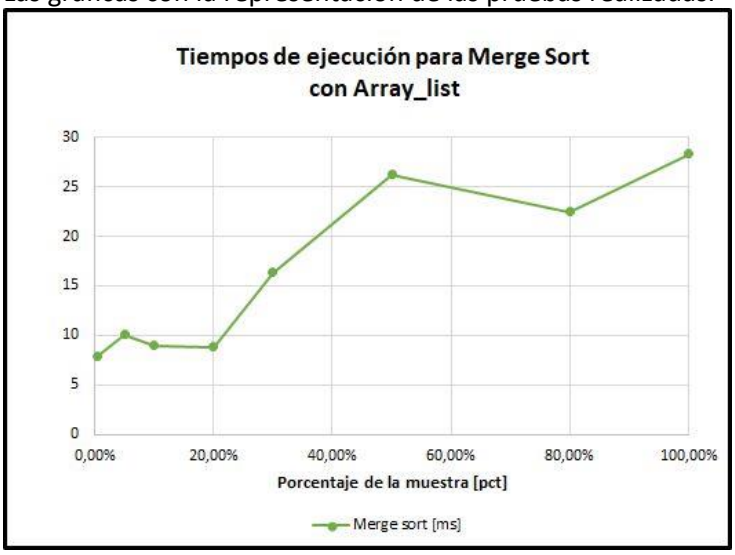
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

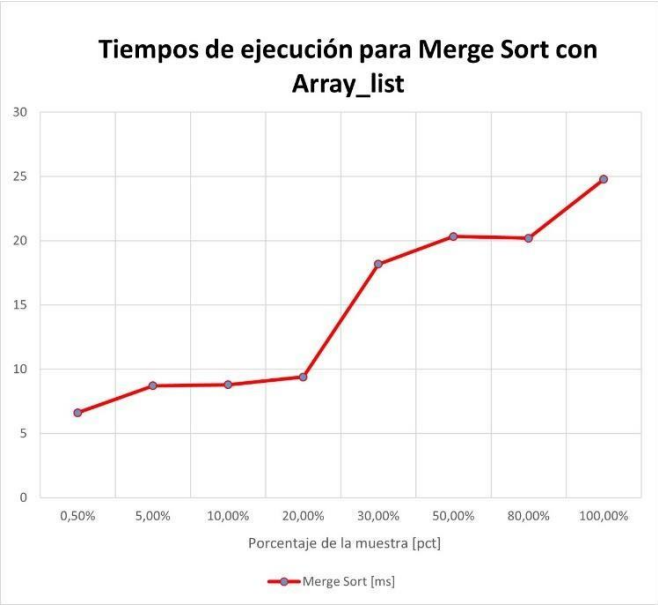
Porcentaje de la muestra [pct]	Porcentaje de la muestra (ARRAY_LIST)	Merge Sort [ms]
0,50%	228	6.61 ms
5,00%	1148	8.72ms
10,00%	2298	8.79ms
20,00%	4598	9.4 ms
30,00%	6898	18.18 ms
50,00%	11498	20.33 ms
80,00%	18397	20.2 ms
100,00%	22998	24.78 ms

Graficas reto 2

Las gráficas con la representación de las pruebas realizadas.



Graficas reto 1



Requerimiento 4

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Género.
Salidas	Los registros de las películas y los programas de televisión clasificadas en el género ingresado por el usuario en orden alfabetico. Además, se tabula con la siguiente información: 1. **Primera tabla:** Type*(movie y Tv Show)* y Count*(la cantidad de veces en las que aparece, ya sea en movie o Tv Show)* **Segunda tabla**: release_year, title, duration, stream_service, director, type, cast, country, rationg, listed_in, description.
Implementado (Sí/No)	Sí. Realizado por: Wilmer Manuel Arévalo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**

Paso 1: Crear lista	O(1)
Paso 2: Iterar y agregar cada título que estén listados dentro de la categoría de interés	O(n)
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
TOTAL	***O(n^(3/2))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash correspondiente a la categoría de interés a la lista	O(1)
Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para "encontrar" uno de sus valores solo se necesita tener la llave, en este caso el nombre de la categoría, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

Reto 1:

Entrada	**Tiempo (s)**

Catalog, Fantasy	17.1 ms
Catalog, Drama	29.35ms
Catalog, Independent Movies	12.79 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Porcentaje de la muestra (ARRAY_LIST)	Merge Sort [ms]
0,50%	228	19.07 ms
5,00%	1148	21.07 ms
10,00%	2298	18.4 ms
20,00%	4598	38.53 ms
30,00%	6898	29.1 ms
50,00%	11498	42.11 ms
80,00%	18397	57.59 ms
100,00%	22998	82.8 ms

Reto 2:

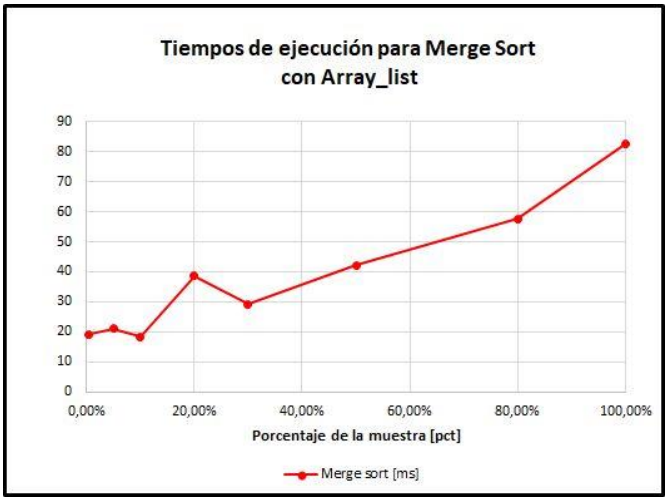
Entrada	Tiempo (s)
Catalog, Fantasy	18.85 ms
Catalog, Drama	24.57 ms
Catalog, Independent Movies	15.78 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Porcentaje de la muestra (ARRAY_LIST)	Merge Sort [ms]
0,50%	228	17.1 ms
5,00%	1148	17.49ms
10,00%	2298	17.17ms
20,00%	4598	17.31ms
30,00%	6898	20.51 ms
50,00%	11498	33.14 ms
80,00%	18397	53.01 ms
100,00%	22998	74.79 ms

Graficas reto 2



Graficas reto 1

Requerimiento 5

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	País.
Salidas	Las películas y programas de televisión producidos en un país específico. Además, se tabula con la siguiente información:

	1. **Primera tabla:** Type*(movie y Tv Show)* y Count*(la cantidad de películas producidos en ese país y la cantidad de programas producidos en ese país.*) 2. **Segunda tabla**: release_year, title, duration, stream_service, director, type, cast, country,rating, listed_in, description.
Implementado (Sí/No)	Sí. Realizado por: Mario Velasquez Semanate

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: Iterar y agregar cada título que hayan sido producidos en el país de interés	O(n)
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
TOTAL	***O(n^(3/2))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash correspondiente al país de interés a la lista	O(1)
Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para “encontrar” uno de sus valores solo se necesita tener la llave, en este caso el nombre del país, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el

efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

Reto 1:

Entrada	**Tiempo (s)**
Catalog, Colombia	7.42 ms
Catalog, Argentina	3.62 ms
Catalog, United States	15.32 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	7.42 ms
5,00%	1148	12.02ms
10,00%	2298	14.89ms
20,00%	4598	17.9 ms
30,00%	6898	18.03 ms
50,00%	11498	20.72 ms
80,00%	18397	25 ms
100,00%	22998	22.48 ms

Reto 2:

Entrada	**Tiempo (s)**
Catalog, Colombia	7.42 ms
Catalog, Argentina	3.62 ms
Catalog, United States	15.32 ms

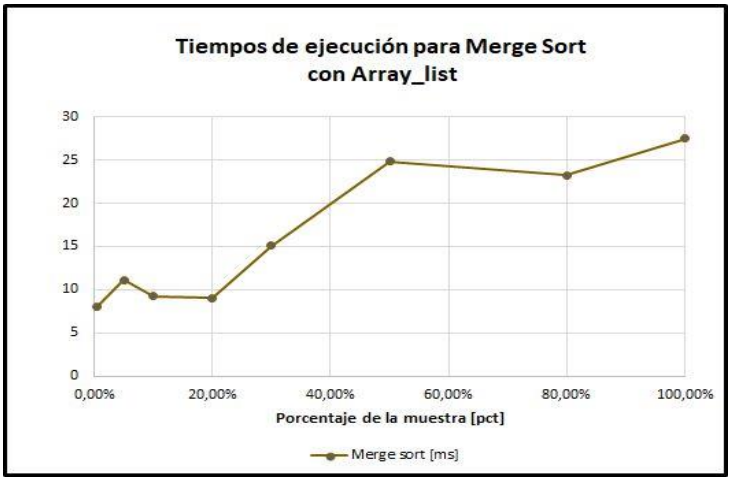
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

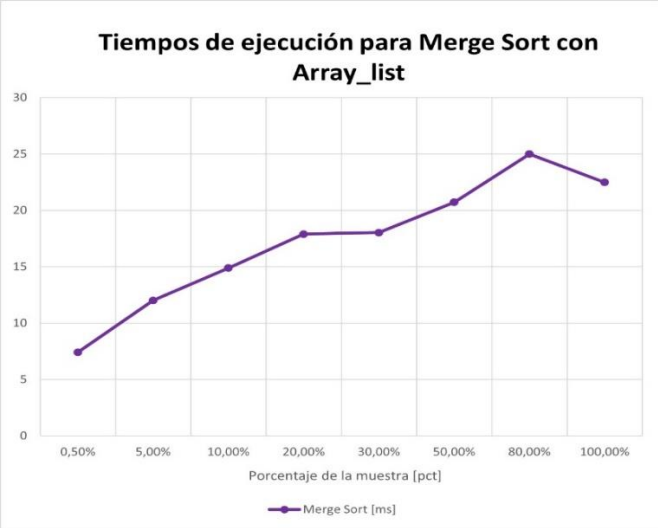
Porcentaje de la muestra [pct]	Porcentaje de la muestra (ARRAY_LIST)	Merge Sort [ms]
0,50%	228	8.03 ms
5,00%	1148	11.09 ms
10,00%	2298	9.25 ms
20,00%	4598	9.05 ms
30,00%	6898	15.08 ms
50,00%	11498	24.85 ms
80,00%	18397	23.25 ms
100,00%	22998	27.48 ms

Graficas reto 2

Las gráficas con la representación de las pruebas realizadas.



Graficas reto 1



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

Requerimiento 6

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Director.
Salidas	Los registros de las películas y los programas de televisión dirigidos por un director ingresado por el usuario. Además, se tabula con la siguiente información: 1. **Primera tabla:** Type*(movie y Tv Show)* y Count*(la cantidad de veces en las que aparece el director, ya sea en movie o Tv Show)* 2. **Segunda tabla:** El número total de películas y programas por plataforma. 3. **Tercera tabla:** El número total de películas y programas por cada género. 4. **Cuarta tabla**: release_year, title, duration, director, stream_service, type, cast, country, rating, listed_in, description.
Implementado (Sí/No)	Sí. Realizado por: Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: Iterar y agregar cada título que hayan tenido como Director/a al Director/a de interés. Además de recolectar la información de las categorías en las que sus películas han sido listadas.	O(n*m) “m” siendo el número máximo de categorías en las que una película es listada
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
TOTAL	***O(n*m))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash correspondiente a la directora/a de interés a la lista además de la información adicional correspondiente a tipo de producción y plataforma	O(1)
Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para “encontrar” uno de sus valores solo se necesita tener la llave, en este caso el nombre del director/a, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

Reto 1:

Entrada	**Tiempo (s)**
Catalog, Steven Spielberg	5.79 ms
Catalog, John Hughes	8.94 ms
Catalog, Mark Williams	8.48 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	5.79 ms
5,00%	1148	7.51 ms
10,00%	2298	10.34ms
20,00%	4598	9.63ms
30,00%	6898	13.19 ms
50,00%	11498	12.54 ms
80,00%	18397	28.52 ms
100,00%	22998	15.03 ms

Reto 2:

Entrada	**Tiempo (s)**
Catalog, Steven Spielberg	5.19 ms
Catalog, John Hughes	10.36 ms
Catalog, Mark Williams	9.34 ms

Tablas de datos

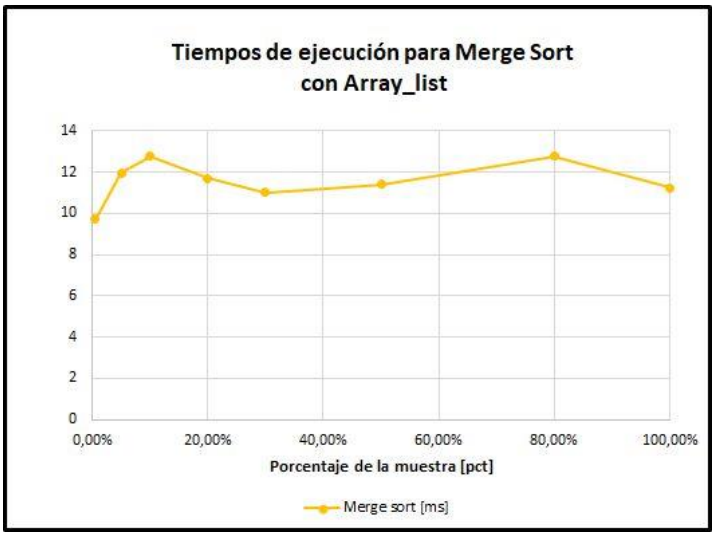
Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	9.72 ms
5,00%	1148	11.94 ms
10,00%	2298	12.78 ms

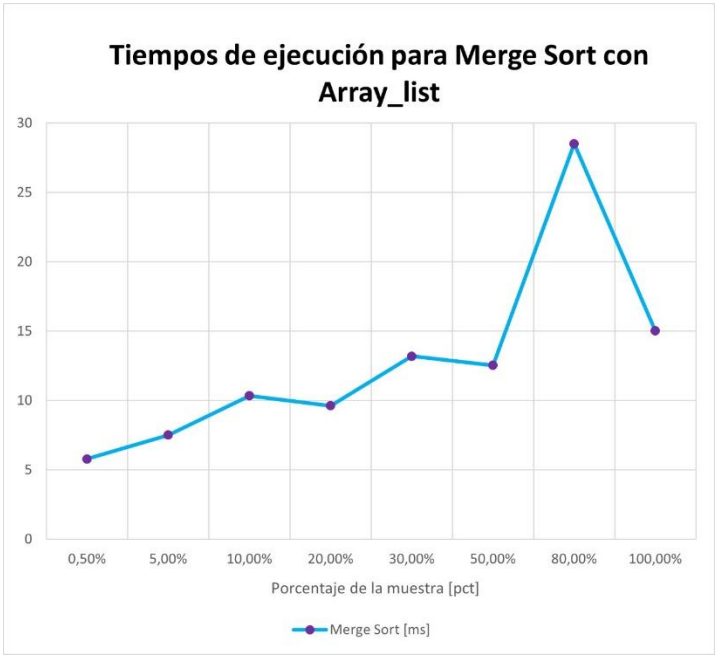
20,00%	4598	11.68 ms
30,00%	6898	11 ms
50,00%	11498	11.4 ms
80,00%	18397	12.76 ms
100,00%	22998	11.24 ms

Graficas reto 2

Las gráficas con la representación de las pruebas realizadas.



Graficas reto 1



Requerimiento 7

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Número de géneros a identificar.
Salidas	Los registros de los géneros con el mayor número de películas y programas de televisión. Además, se tabula con la siguiente información: 1. **Primera tabla:** El grupo de N géneros organizados por el número de películas y programas (listed_in y count) 2. **Segunda tabla:** De los géneros de la anterior tabla se tabula: rank, listed_in, count, type, stream_service.
Implementado (Sí/No)	Sí. Realizado por: Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Reto 1:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: Iterar y agregar cada título que hayan pertenezca a la género de interés. Además de recolectar la información adicional de los títulos.	O(n*m) “m” siendo el número máximo de géneros en las que una película es listada
Paso 3: Ordenar la lista (Shell Sort)	O(n^(3/2))
Paso 4: Filtrar el top (N) de títulos	O(1)
TOTAL	***O(n^3/2))***

Reto 2:

Pasos	**Complejidad**
Paso 1: Crear lista	O(1)
Paso 2: agregar la entrada de la tabla de hash correspondiente al género de interés a la lista además	O(1)

de la información adicional correspondiente a tipo de producción y plataforma	
Paso 3: Ordenar la lista (Merge Sort)	O(n*log(n)))
Paso 4: Filtrar el top (N) de títulos	O(1)
TOTAL	***O(n*log(n)))***

En las anteriores gráficas se evidencia que en el paso 2 de ambos algoritmos se da el cambio en complejidad relacionado con el uso de tablas de Hash en vez de listas como en el reto 1, pues la complejidad pasa de O(n) en el reto 1 a O(1) en el reto 2. Lo anterior se debe a que las tablas de Hash no iteran sobre cada elemento, sino que para "encontrar" uno de sus valores solo se necesita tener la llave, en este caso el nombre del género, por lo que se logra desarrollar este ´paso de manera mucho más eficiente.

Por otro lado, en el reto 1 nuestro grupo utilizo el método de Shell Sort mientras que en el presente reto se utilizó el Merge Sort que es más eficiente. El cambio es especialmente importante si se tiene en cuenta que una complejidad O(n*log(n)) [la del Merge Sort] es cóncava mientras que la complejidad O(n^(3/2)) [la del Shell Sort] es convexa, es decir, cada vez que se agregan más datos el efecto marginal por dato va decreciendo con el método Merge Sort pero con el método Shell Sort los rendimientos marginales son crecientes lo que la hace muy ineficiente comparado al método utilizado en el reto 2.

En general, se evidencia una mejora de eficiencia en los pasos 2 y 3 entre el reto 1 y el reto 2 por lo que el código actual para este requerimiento en este reto es más eficiente que en el reto 1.

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).

Reto 1:

Entrada	**Tiempo (s)**
Catalog, top 7	11.93 ms
Catalog, top 10	13.76 ms
Catalog, top 25	38.31 ms

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	12 ms

5,00%	1148	14.1 ms
10,00%	2298	19.28 ms
20,00%	4598	23.49 ms
30,00%	6898	26.46 ms
50,00%	11498	34.72 ms
80,00%	18397	49.21 ms
100,00%	22998	58.12 ms

Reto 2:

Entrada	**Tiempo (s)**
Catalog, top 7	11.93 ms
Catalog, top 10	11.34 ms
Catalog, top 25	23.47 ms

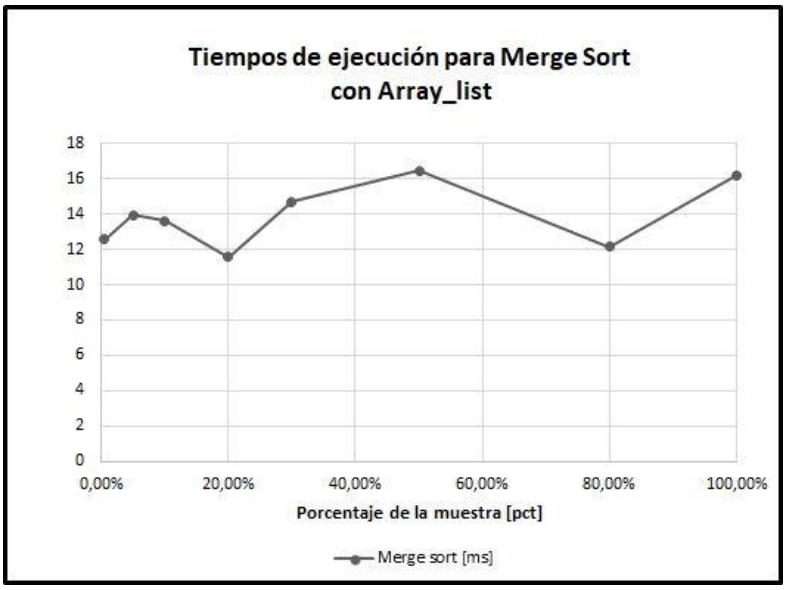
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	**Porcentaje de la muestra (ARRAY_LIST)**	**Merge Sort [ms]**
0,50%	228	12.59 ms
5,00%	1148	13.96 ms
10,00%	2298	13.62 ms
20,00%	4598	11.55 ms
30,00%	6898	14.7 ms
50,00%	11498	16.45 ms
80,00%	18397	12.14 ms
100,00%	22998	16.21 ms

Graficas reto 2

Las gráficas con la representación de las pruebas realizadas.



Graficas reto 1