María Castro 202020850 m.castroi@uniandes.edu.co
Valentina Calderón 202020771 v.calderonm@uniandes.edu.co

Sustentación Reto 2

Función 1/Carga de Datos:
Objetivo: Cargar la información del archivo de videos y categorías. Al final de cada carga reportar el total de registros de videos cargados del archivo, mostrar información del primer video cargado, imprimir la lista de categorías cargadas mostrando su id y nombre.

MVC:
View:

1. Llama a la función initCatalog del controlador
2. Llama a la función loadData del controlador
3. Después de la carga, imprime el total de registros de videos y total de registros de categorías cargados del archivo.

Controller:

1. Llama a la función newCatalog() del modelo
2. loadData() llama a loadVideos() y loadCategoryIds()
    a. loadVideos() recorre cada línea del archivo de videos O(n) y llama al modelo addVideo() para añadir los datos a la lista de 'videos' del catálogo.
    b. loadCategoryIds() recorre cada línea del archivo de categorías O(n) y llama al modelo addCategory() para que añada cada nueva categoría.

Model:

1. newCatalog() crea la estructura del catálogo de videos.
    a. 'videos' es una lista con todos los videos que se cargaron del archivo.
    b. 'category-id' es un HashMap del tipo PROBING con factor de carga de 0.5. Este guarda parejas llave-valor: llave es el nombre, el valor es su id.
    c. 'by_countries': es un HashMap del tipo PROBING con factor de carga de 0.5. Este mapa guarda parejas llave-valor cuya llave es el nombre de un país, y su valor es una lista con los videos que son de ese país.
    d. 'by_categories': es un HashMap del tipo PROBING con factor de carga de 0.5. Este mapa guarda parejas llave-valor cuya llave es un category id, y su valor es una lista con los videos que son de esa categoría.
2. addVideos() agrega el video que recibe al final de la lista de 'videos' del catálogo (addLast() - O(1)). Llama a las funciones addVideoCountry() y addVideoCategory()
    a. addVideoCountry() se encarga adicionar un video a su lista de países correspondiente dentro del mapa por países.
    b. addVideoCategory() se encarga de adicionar un video a su lista de categorías correspondiente dentro del mapa por categorias.

    c. mp.contains() - **O(1)** , mp.get() - O(1) *best case, mp.put() - **O(n)**

3. addCategory() recibe los datos de una categoría y los guarda dentro del mapa de category-id.
    a. mp.put() - **O(n)**

Complejidad:
O(v) + O(c) → O(n)

- v es la cantidad de videos que hay en el archivo - lineal
- c es la cantidad de categorías que hay en el archivo - lineal

Función 2 / Requerimiento 1:

Objetivo: Conocer cuáles son los n videos con más views que son tendencia en un determinado país, dada una categoría específica.

MVC:
View:

1. Se le pregunta al usuario la cantidad de videos, el país, y la categoría.
2. Verifica los datos con getCategoryId() & getCountry()
3. Llama a la función del controller topCountryCategory()
4. Imprime la siguiente información de los videos que se encuentran en el top: trending:date, title, channel_title, publish_time, views, likes, dislikes. (O(n) – imprimir)

Controller:

1. getCategoryId() - llama a la función del modelo getCategoryId()
2. getCountry() - llama a la función del model getCountry()
3. topCountryCategory() - llama a las funciones del controller getCategory(), sortViews() findTopsCountryCategory()
    a. sortViews(): llama una función del modelo sortViews que devuelve la lista sorteada según la cantidad de views.
    b. findTopsCountryCategory(): llama a la función del modelo findTopsCountryCategory() devuelve una lista con los x videos con más views.

Model:

1. getCategoryId() - revisa si el mapa by_categories tiene la categoria, si existe saca la pareja llave-valor y luego el valor de dicha pareja el cual retorna, si no esta retorna None.
    a. mp.contains() - **O(n)**, mp.get() - O(1) *best case, O(n) worst case
2. getCountry() - revisa si el mapa by_countries, sí tiene el país y retorna el valor de asociado a dicho país si lo encuentra, de lo contrario retorna None.
    a. mp.get() - O(1) *best case

3. getCategory() - revisa si el mapa by_categories, y retorna la lista de videos asociada a la llave del id, si no lo encuentra retorna none.
    a. mp.get() - O(1) *best case, **O(n) worst case**
4. sortViews() - Se hace un mergesort para organizar la lista comparando los views.
    a. O(nlogn)
5. findTopsCountryCategory() - Se crea una nueva lista vacía. Recorre los elementos de la lista anteriormente ordenada hasta que ésta ya tenga los top x videos o se haya llegado al fin de la lista ordenada. Un video se añade a la nueva lista si su país es el mismo que el que desea el usuario.
6. PeorCaso → O(n)

Complejidad:
O(n) + O(nlogn) + O(n)
O(nlogn) → mayor complejidad

Función 3 / Requerimiento 2 (Maria):
Objetivo: Conocer cuál es el video que más días ha sido trending para un país específico.

MVC:
View:
1. Se le pregunta al usuario el país a consultar el video trending por más días.
2. Verifica el país con getCountry() **
3. Si existe el país se llama a la función del controlador topVidByCountry() que retorna el video que más días ha sido tendencia del país ingresado por parámetro.
4. Imprime la siguiente información del vídeo que ha sido trending por más días: title, channel_title, coutry, número de días como tendencia.

Controller:
1. getCountry() llama a la función del modelo getCountry() que retorna los datos de los videos del país solicitado por parámetro.
2. topVidByCountry() se encarga de:
    a. sortVideoId() llama a la función del modelo sortVideoID() que devuelve una lista ordenada por los video_ id .
    b. findTopCountry(): Llama a la función del modelo findVideoTopCountries() a cual retorna la información del video que más días ha sido tendencia y la cantidad de días tendencia en país ingresado por parámetro

Model:
1. getCountry():revisa si el mapa by_countries, sí tiene el país y retorna el valor de asociado a dicho país (la lista de videos de ese país) si lo encuentra, de lo contrario retorna None.
    a. mp.get() - O(1) *best case, O(n) worst case

b.

2. sortVideoId: Se hace un mergesort para organizar la lista comparando los video_ids. O(nlogn)
3. findTopVideoCountries():
    - Crea una lista en la que se guardará cada video y las veces que este aparece dentro de la lista ordenada.
    - Recorreré la lista ordenada y para cada video revisa si el siguiente es el mismo (tienen el mismo país), si lo son le suma uno a la cantidad de repeticiones. Si es un video diferente, añade a la lista el video con sus datos, y la cantidad de veces que este apareció. (O(n)). El video y las repeticiones se guardan en un hashmap.
        - **mp.put() - O(1), O(n) worst case**
    - Al final de este recorrido, se recorre la lista que se creó con cada video y sus repeticiones para determinar cual tiene la mayor cantidad de repeticiones (O(n)).
    - Se retorna los datos del video con que más veces fue tendencia y su cantidad de repeticiones.

Complejidad:
O(n) + O(nlogn) + O(n) + O(n)
O(nlogn) → Complejidad más grande

Función 4 / Requerimiento 3 (Valentina):

Objetivo: Conocer cuál es el video que más días ha sido trending para una categoría específica.

MVC:
View:

1. Se le solicita al usuario ingresar la categoría a consultar el video trending por x días.
2. Verifica la categoria con la función del controller getCategoryId()
3. Si existe la categoría se llama a la función del controlador topVidByCategory() que retorna el video que más días ha sido trending.
4. Imprime la siguiente información del vídeo que ha sido trending por más días: title, channel_title, category_id, número de días como tendencia

Controller:

1. getCategoryId() - llama a la función del modelo getCategoryId()
2. topVidByCategory()
    a. getCategory() - llama la función del modelo getCategory la cual retorna los datos de los videos de la categoría deseada

b. sortVideoId() - Llama a la función del modelo sortVideoID() la que devuelve una lista ordenada por los video_id.
c. findTopVideo()- Llama la función del modelo findTopVideo() la cual retorna la información del video que más días ha sido tendencia y la cantidad de días tendencia.

Model:

1. getCategoryId() - revisa si el mapa by_categories tiene la categoría, si existe saca la pareja llave-valor y luego el valor de dicha pareja el cual retorna, si no esta retorna None.
    a. mp.contains() - **O(n)**, mp.get() - O(1) *best case
2. getCategory() - revisa si el mapa by_categories, y retorna la lista de videos asociada a la llave del id, si no lo encuentra retorna none.
    a. mp.get() - O(1) *best case, O(n) worst case
3. sortVideoId(): Se hace un mergesort para organizar la lista comparando los video_ids.
    a. O(nlogn)
4. findTopVideo():
    - Crea una lista en la que se guardará cada video y las veces que este aparece dentro de la lista ordenada.
    - Recorreré la lista ordenada y para cada video revisa si el siguiente es el mismo (tienen el mismo país), si lo son le suma uno a la cantidad de repeticiones. Si es un video diferente, añade a la lista el video con sus datos, y la cantidad de veces que este apareció. (O(n)). El video y las repeticiones se guardan en un hashmap.
        - **mp.put() - O(1) *best case, O(n) worst case**
    - Al final de este recorrido, se recorre la lista que se creó con cada video y sus repeticiones para determinar cual tiene la mayor cantidad de repeticiones (O(n)).
    - Se retorna los datos del video con que más veces fue tendencia y su cantidad de repeticiones.

Para los requerimientos 2 y 3: No se tiene en cuenta los elementos que tengan como video_id '#NAME?". Aunque estos videos se podrían tener en cuenta si también se utilizará como criterio el title del vídeo, hacer esta doble verificación implicaría volver a recorrer toda la lista y comprar el elemento actual con el título de todos los videos. (Como la lista está organizada por video_id, nada garantiza que antes del elemento actual no haya habido videos con el mismo título). Este doble recorrido doble de la lista ordenada, O(n^2), afecta significativamente la eficiencia.

Complejidad:

O(1) + O(1) + O(nlogn) + O(n) + O(n) + O(n)
O(nlogn) - mayor complejidad

Función 5 / Requerimiento 4:

Objetivo: Conocer cuáles son los n videos diferentes con más likes en un país con un tag específico.

MVC:

View:

1. Se le solicita al usuario que ingrese la cantidad, el país y el tag a consultar de videos con más likes.
2. Verfica el pais con getCountry()
3. Se llama a una función del controller getCountry() que retorna una lista de videos pertenecientes al país solicitado.
4. Si se encuentra el país, se llama a una función del controller listVidTag()
5. que retorna una lista con los videos con más likes dados un pais y un tag
6. Imprime la siguiente información de los videos que se encuentran en el top de likes: title, channel_title, publish_time, views, likes, dislikes, tags. (O(n) – imprimir)

Controller:

1. getCountry() - llama a la función del model getCountry()
2. listVidTag()
    a. findWithTags(): llama a la función del modelo findWithTags() que retorna una lista filtrada con los videos que tenga dentro de sus tags el tag deseado por el usuario.
    b. sortLikes(): Llama a la función del modelo sortLikes() la cual retorna una lista ordenada por likes
    c. findMostLikes(): Llama a la función del modelo findMostLikes() la cual retorna la lista con los x videos con más likes.

Model:

1. getCountry():revisa si el mapa by_countries, sí tiene el país y retorna el valor de asociado a dicho país (la lista de videos de ese país) si lo encuentra, de lo contrario retorna None.
    a. mp.get() - O(1) *best case, O(n) worst case
2. findWithTags() :
    a. O(n)
    - Del mapa de países, saca la pareja llave valor del países deseado y luego el valor que es la lista de videos de ese país.
    - Crea un lista vacia
    - Recorre todos los videos de la lista y si estos tiene el tag deseado los agrega a la lista nueva. Devuelve la lista filtrada por tags.
3. sortLikes() - Se hace un mergesort para organizar la lista comparando los likes. O(nlogn)

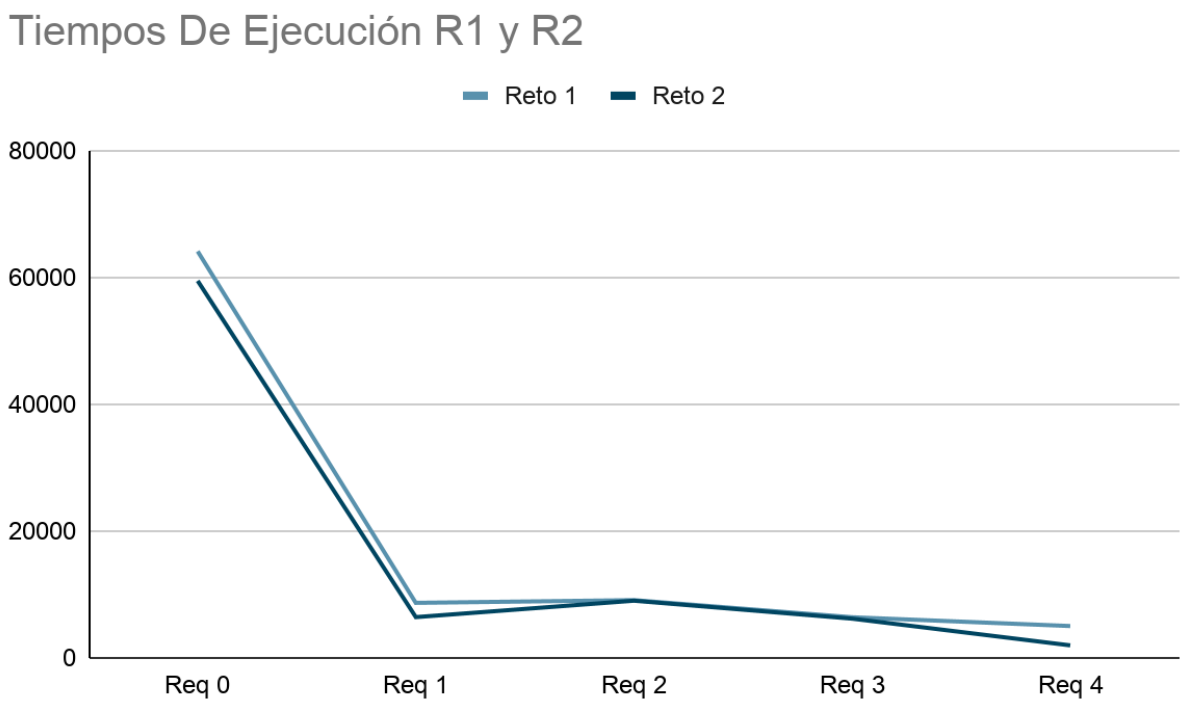
4. findMostLikes():
    - Se crea una lista vacía que tendrá los x top videos, que comprara por video id.
    - A esta lista le agrega el último elemento de la lista ordenada por likes.
    - Se recorre la lista de todos los elementos organizados por likes hasta que ya se tengan los top x elementos o se llegue al principio de la lista (se recorre al revés ya que los views se ordena de menor a mayor)
        - O(n) - peor.
    - Se añade una video a lista de top si no existe ya un video con el mismo video_id
        - (isPresent-O(n), n máx será 1 menos que la cantidad de videos que solicite el usuario).
    - Se retorna la lista de videos top

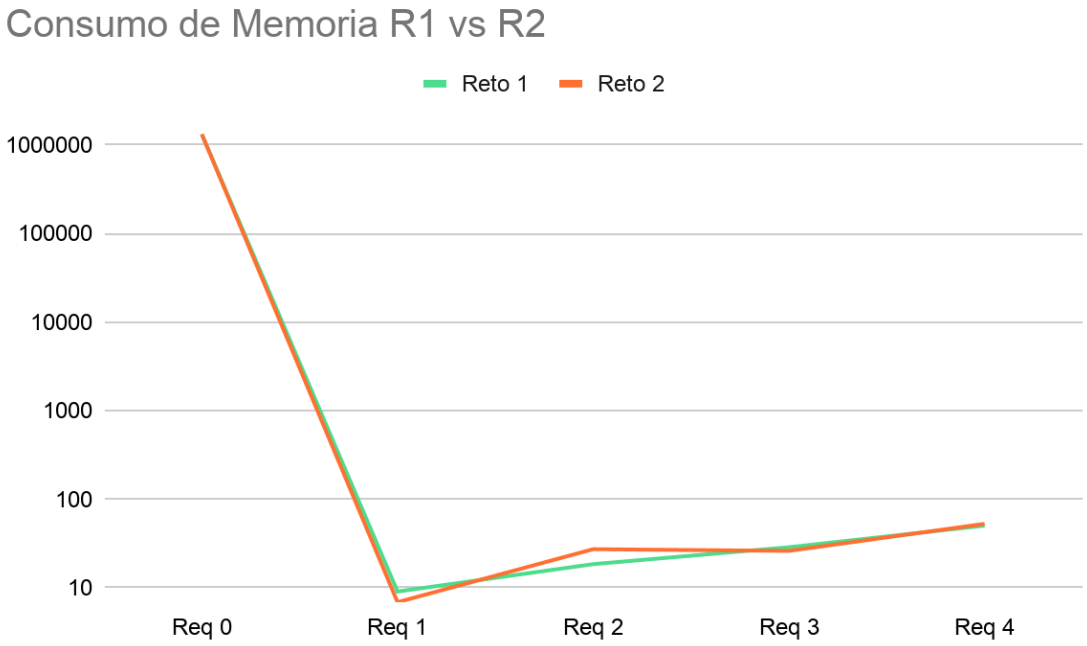
Complejidad:
$O(n_1) + O(nlogn)$ $O(nlogn) + O(n_2)$
O(nlogn) → Mayor complejidad
n_1 número de elementos que tiene el tag
n_2 número de elementos que ta hay en la lista de los top (depende de cuantos elementos haya pedido el usuario y cuantos ya se han agregado a esta lista)

Parte 4: Análisis de Resultados

Requerimiento	Caso de Prueba	Dato	Reto 1	Reto 2
Req 0	Carga de datos	Tiempo de Ejecución [ms]: Consumo de Memoria[kB]:	64111.3 1327231.699	59458.849 1327208.32
Req 1	Top: 10 Pais: Canada Categoria: Music	Tiempo de Ejecución [ms]: Consumo de Memoria[kB]:	8685.546 9.08	6436.017 6.885
Req 2:	Pais: Canada	Tiempo de Ejecución [ms]: Consumo de Memoria[kB]::	9114.353 18.516	9027.212 27.311
Req 3	Categoria: Music	Tiempo de Ejecución [ms]: Consumo de Memoria[kB]:	6418.519 28.727	6193.468 26.132
Req 4	Top: 10 Pais: Canada Tag: 2018	Tiempo de Ejecución [ms]: Consumo de Memoria[kB]:	5031.155 50.420	1984.047 52.812

En casi todos los casos, el Reto 2 tuvo un tiempo de ejecución y memoria menor al Reto 1.
Las estructura de datos creada para el Reto 1 fue muy similar a la del Reto 2, en el reto uno se usaron diccionarios para guardar listas en las que ya había un filtro sobre la categoría o sobre el país. Del mismo modo, en Reto 2 utilizamos el TAD Map para crear una estructura muy similar. La gran diferencia entre la implementación de estos Retos fue el uso de los HashMaps en vez de los diccionarios nativos de Python, dado que estas son estructuras muy similar los tiempos de ejecución no fueron drásticamente diferentes, sin embargo, el uso del TAD y sus respectivas funciones agilizaron los procesos ya que la funciones como .get() aseguran una complejidad de O(1) si los HashMaps están bien distribuidos.



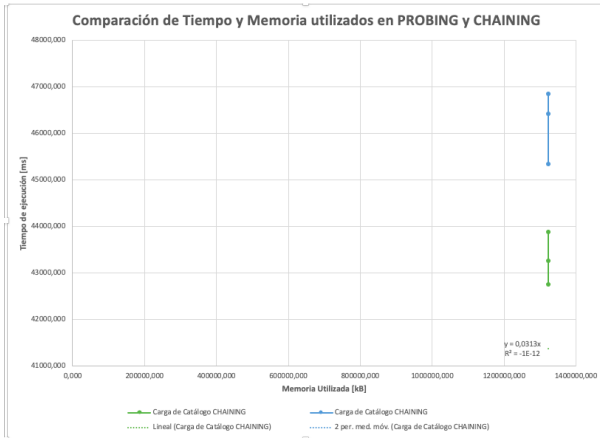


Carga de Catálogo PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución [ms]
0,30	1324009,838	43252,579
0,50	1324009,838	42742,973
0,80	1324009,838	43873,408

Carga de Catálogo CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución [ms]
2,00	1324009,838	45344,070
4,00	1324009,838	46411,244
6,00	1324009,838	46841,351

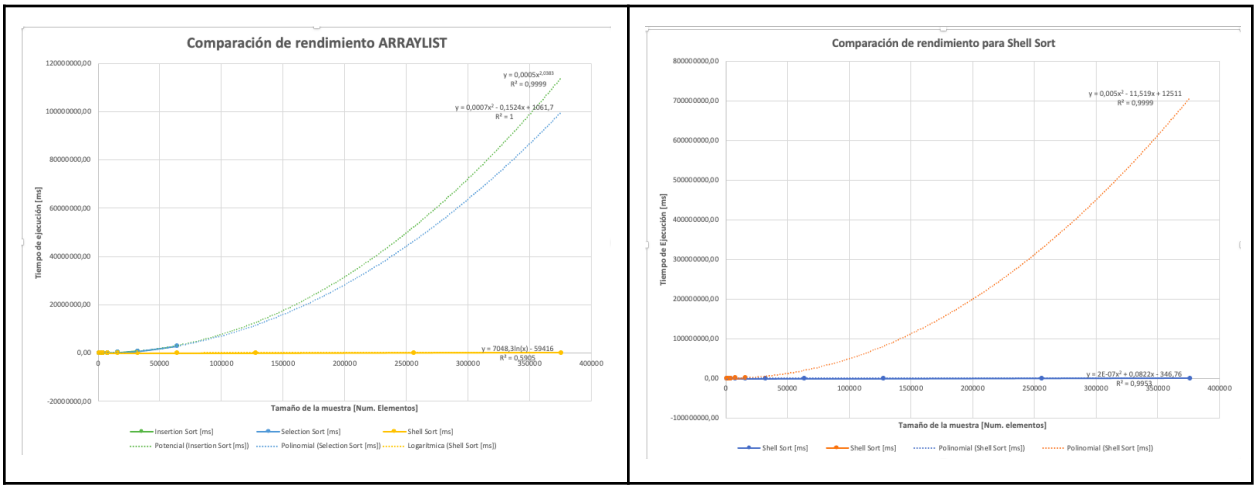


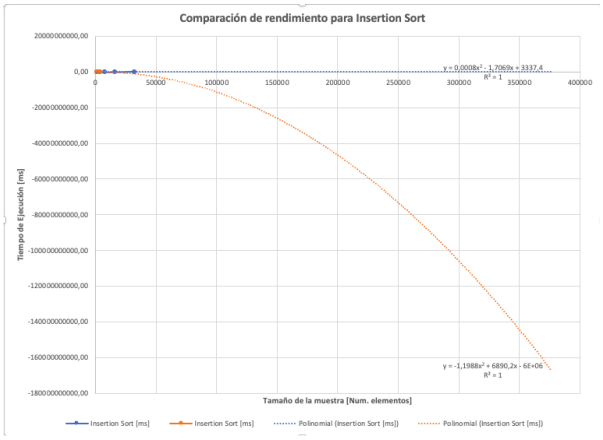
Basado en estos resultados y en la teoría vista en clase, escogimos una carga de catálogo de PROBING con un factor de carga de 0.5.

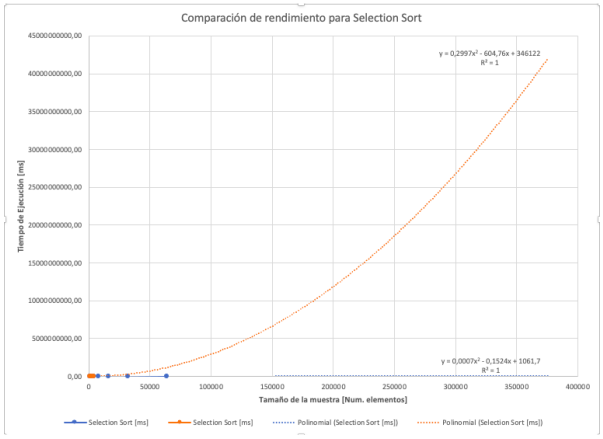
Tamaño de la muestra (ARRAYLIST)	Insertion Sort [ms]	Selection Sort [ms]	Shell Sort [ms]
1000	637,60	670,63	37,69
2000	2713,14	2916,61	85,10
4000	10936,16	10710,49	192,73
8000	43601,07	44929,52	431,34
16000	179375,07	186474,09	1017,22
32000	771539,28	716213,22	2438,77
64000		2890360,03	5651,87
128000			10518,68
256000			35082,15
375942			54937,90

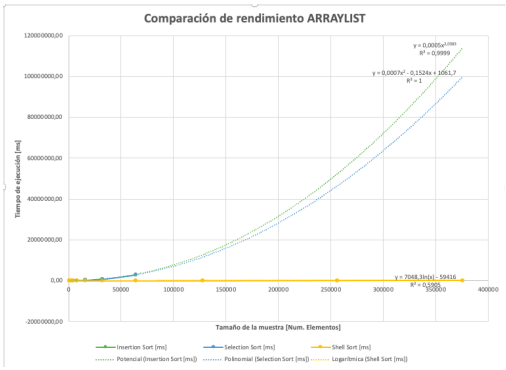
Tamaño de la muestra (LINKED_LIST)	Insertion Sort [ms]	Selection Sort [ms]	Shell Sort [ms]
1000	45575,74	41042,89	2074,96
2000	3339334,56	335315,18	11181,70
4000	2734041,40	2721915,21	53057,86
8000			238796,86
16000			1119501,94
32000			
64000			
128000			
256000			
375942			

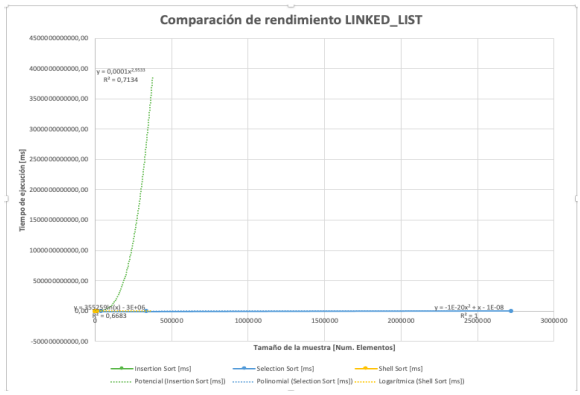
Basándonos en estos resultados del Reto 1, decidimos usar los ARRAY_LIST como estructura de datos para las listas dentro del Reto.




Comparación de rendimiento para Insertion Sort
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
y = 0,0008x² - 1,7069x + 3337,4
R² = 1
y = -1,1988x² + 6890,2x - 6E+06
R² = 1
Insertion Sort [ms]
Insertion Sort [ms]
Polinomial (Insertion Sort [ms])
Polinomial (Insertion Sort [ms])


Comparación de rendimiento para Selection Sort
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
y = 0,2997x² - 604,76x + 346122
R² = 1
y = 0,0007x² - 0,1524x + 1061,7
R² = 1
Selection Sort [ms]
Selection Sort [ms]
Polinomial (Selection Sort [ms])
Polinomial (Selection Sort [ms])


Comparación de rendimiento ARRAYLIST
Tiempo de ejecución [ms]
Tamaño de la muestra [Num. Elementos]
y = 0,0005x^2.1581
R² = 0,9999
y = 0,0007x² - 0,1524x + 1061,7
R² = 1
y = 7048,3ln(x) - 59416
R² = 0,5905
Insertion Sort [ms]
Selection Sort [ms]
Shell Sort [ms]
Potencial (Insertion Sort [ms])
Polinomial (Selection Sort [ms])
Logaritmica (Shell Sort [ms])


Comparación de rendimiento LINKED_LIST
Tiempo de ejecución [ms]
Tamaño de la muestra [Num. Elementos]
y = 0,0001x^2.9939
R² = 0,7134
y = 35525ln(x) - 3E+06
R² = 0,6683
y = -1E-20x² + x - 1E-08
R² = 1
Insertion Sort [ms]
Selection Sort [ms]
Shell Sort [ms]
Potencial (Insertion Sort [ms])
Polinomial (Selection Sort [ms])
Logaritmica (Shell Sort [ms])