Grupo 2 | Documento de análisis EDA Reto 1

Link Repositorio reto: https://github.com/EDA2021-2-SEC07-G02/Reto1-G02

Hecho por:

- Daniel Gomez
- Jenifer Arce

Contenido

Observaciones:

- Utilizamos la librería de Prettytable para imprimir los resultados en el view. En el servidor de discord autorizaron usar esta librería el día 8 de septiembre.
- Utilizamos la librería re para verificar que las entradas del usuario tales como fechas tuvieran el formato adecuado para ejecutar la función
- Todas las pruebas fueron hechas en la aplicación de Visual Studio Code de Nukak. Las especificaciones de la máquina eran 64 gb de memoria Ram, con respecto al procesador de la máquina no conocemos sus detalles, dado que solo podíamos acceder a la aplicación en Nukak, no a la configuración del equipo.
- El tipo de implementación de lista que se utilizó para la carga del catalogo fue ARRAY_LIST. La razón de esto es porque necesitábamos ingresar a posiciones específicas en la mayoría de los requerimientos.

- En la mayoría de requerimientos que necesitaban ordenamiento en listas usamos ordenamiento por merge sort. La principal razón de la elección de este ordenamiento fue por los resultados de las pruebas de rendimiento. Sabemos que este algoritmo puede poseer unas deficiencias en el uso de memoria/espacio extra, sin embargo, para este reto nos enfocamos en la rapidez, y este ordenamiento nos brindaba en el peor caso O(N*log(N)), mientras que otros ordenamientos O(N^2)

Requerimiento 1 | Grupal

Explicación general:

Problemas que encontramos.

1. ¿Es mejor ordenar primero el catálogo por fechas? o ¿Es mejor delimitar los artistas por fechas en una nueva lista y después ordenar?

Al hacer este requerimiento decidimos que con el fin de que algoritmo fuera más eficiente se tenía que comprobar primero las fechas de cada artista. Lo anterior, cumple con el propósito principal del proyecto, además nos ayuda a tener una lista final con menores cantidad de elementos a ordenar. Esto nos ayuda a tener una mayor eficiencia, dado que el mejor caso en un algoritmo de ordenamiento es O(N), pero es muy distinto el tiempo en total cuando n=100.000 artistas, a que n=30.000 artistas.

2. ¿Qué ordenamiento es mejor en este requerimiento?

Debido a los resultados de las pruebas del laboratorio de la semana 5 estábamos entre dos algoritmos de ordenamiento, insertion sort y merge sort. Los resultados de esta prueba mostraron que era más eficiente merge sort en cualquier tipo de muestra (Ver pruebas de rendimiento).

Funciones realizadas para este requerimiento:

Función	Categoría	Explicación[1]
cmpArtistDate	Función de ordenamiento	Es la función que se usa para ordenar a los artistas que hayan nacido en las fechas deseadas.
listarArtistasCronologicamente	Función principal / función de consulta	Es la función principal del requerimiento. Primero se filtran los artistas que hayan nacido dentro del rango de fechas y se van añadiendo a una nueva array_lista. Después de eso se hace un ordenamiento con merge y se usa la

[1] En la documentación del model está una explicación detallada de cada una de las funciones.

		función de comparación cmpArtistDate. Funciones TAD usadas: lt.newList(): crea una nueva lista para los artistas que hayan nacido en la fecha adecuada lt.iterator () : permite iterar sobre toda la lista de artistas lt.addLast () : añadé a los artistas que nazcan en las fechas deseadas

Tabla 1. Funciones requerimiento 1

Pruebas de rendimiento:

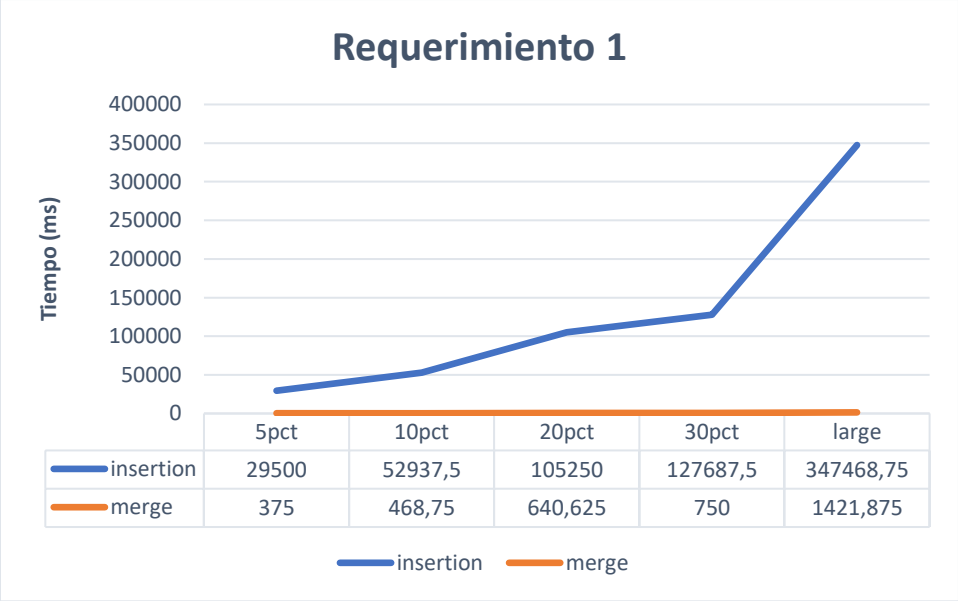


Figura 1. Pruebas de rendimiento requisito 1. Computador con 64gb de Ram

Como se puede observar en la gráfica, en el ordenamiento con merge tuvo un mayor rendimiento que el insertion independiente de la muestra. Mediante este resultado pudimos observar el mejor caso de ordenamiento de merge O(n*log(n)).

An*álisis de complejidad:*

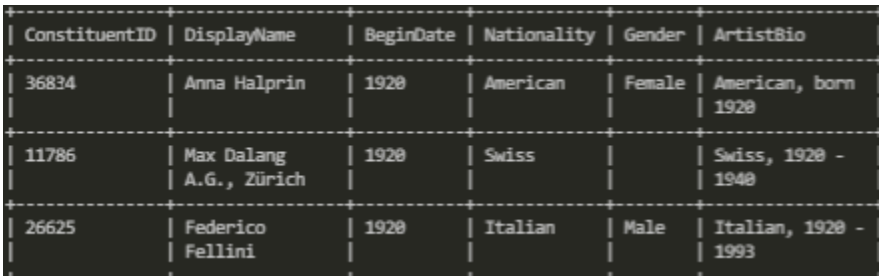
Función principal	Complejidad

listarArtistasCronologicamente	O(N) : dado que se recorre primero la el catálogo de los artistas
Ordenamiento con merge	Mejor y peor caso: O(n*log(n))

Requerimiento 2 | Grupal

Observación:

Para la función de comparación el ordenamiento se hace dependiendo de la fecha de nacimiento. En caso de que esta sea igual se comparan sus fechas de fallecimiento, si la fecha de fallecimiento del artista 1 es menor que a la del 2 se ordenará antes que el artista 2. En el siguiente ejemplo se muestra como se ordena. La razón de este ordenamiento es que quisimos hacer un ordenamiento cronológico "estricto".



ConstituentID	DisplayName	BeginDate	Nationality	Gender	ArtistBio
36834	Anna Halprin	1920	American	Female	American, born 1920
11786	Max Dalang A.G., Zürich	1920	Swiss		Swiss, 1920 - 1940
26625	Federico Fellini	1920	Italian	Male	Italian, 1920 - 1993

Figura 2. Ejemplo de ordenamiento del requerimiento 2

Explicación general:

El requerimiento entrega una lista de obras ordenadas por fecha de adquisición. Además de cuantas obras se adquirieron por "Purchase" o compra del museo.

El reto de mayor complejidad fue optimizar este algoritmo para que el tiempo de ejecución estuviera en rangos moderados. La solución fue utilizar un método selectivo que redujera la lista a organizar. Primero se pasaba por la lista de adquisiciones y se añadían aquellas que cumplieran el rango de fecha, ya descartadas, se organizaban las adquisiciones del rango deseado.

Funciones del modelo realizadas para este requerimiento:

Función	Categoría	Explicación[2]

[2] En la documentación del model está una explicación detallada de cada una de las funciones.

listarAdquisicionesCronologicamente	Función de consulta	Entrega una lista con las adquisiciones organizadas por fecha de adquisición en un rango seleccionado por el usuario.
cmpArtworkByDateAcquired	Función de comparación	Compara dos obras (adquisiciones) por su fecha de adquisición. *Peor caso: O(k)*
sortList	Función de ordenamiento	Función global de ordenamiento, en este caso se utilizo MergeSort para los ordenamientos. *Peor Caso O(Nlog(N))*

Análisis de Complejidad

Caso	Complejidad	Explicación
Mejor Caso	O(N)	El mejor caso sucede cuando no hay obras de arte en el rango seleccionado, por lo que solo se recorren las obras.
Peor Caso	O(NlogN)	El peor caso es cuando todas las obras entran en el rango de fecha y todas se deben organizar. Se deja la complejidad de orden mayor que será la del ordenamiento MergeSort, este resultado es independiente de si las obras están organizadas o no.

Pruebas de rendimiento:

	Small	5%	10%	20%	30%	50%	80%	large
Insertion	3453,13	330484						
Selection	437,5	5171,88	10953,1	23500	41625,5	67500,8	123734	141063
Merge	328,5	3765,63	7593,75	16296,9	30359,8	51125,6	76843,8	97171,9
Quick	526,553	21140,6	67328,1	249516	572469			

[3]

Figura 3. Pruebas de rendimiento requisito 2

Debido a los resultados obtenidos en las pruebas de rendimiento seleccionamos el ordenamiento por merge sort. Dado que su rendimiento fue el mejor en tiempo independiente de la muestra.

Requerimiento 3 | Hecho por Daniel

Explicación general:

El requerimiento consiste en clasificar las obras de un artista por técnicas *(Medium).* Indicando la cantidad de obras que tenía en cada técnica, y una lista con todas las obras de la técnica más utilizada.

Los retos de mayor complejidad que se encontraron fueron los siguientes:

- Encontrar el ConsituentID dado un nombre de artista para buscar en la base de datos de las obras.

[3] No se tomaron algunos valores de tiempos dado que la máquina fallo. El sistema nos entregó un mensaje de que Python no respondía.

- Guardar los datos únicamente con tipos de datos de la librería DISCLib. Se debían guardar las obras de arte en una lista de cada técnica y las técnicas a su vez debían estar en una lista sin utilizar diccionarios ni listas nativas de Python.
- Optimizar el problema de tal manera que el tiempo de ejecución del algoritmo estuviera en un rango de tiempos razonable.

La solución para cada problema:

- Para mantener una complejidad baja primero se hace una búsqueda del artista y se extrae el ConstituentID, en el peor caso de esta búsqueda será O(N), el mejor O(K).
- Para guardar los datos se planteó la siguiente estructura: una lista de la librería DISCLib que almacena en cada espacio una lista (también de la librería DISCLib) de obras correspondiente a cada técnica. En la vista para obtener el nombre de la técnica más utilizada se extrae el primer elemento de la lista de cada técnica y se encuentra la técnica de la obra.
- Para reducir la complejidad al máximo primero se seleccionan las obras del artista y se van añadiendo a la lista de obras para cada técnica correspondiente. Luego se organiza la lista de técnicas (lista de listas) por el tamaño de la lista interior (cantidad de obras en cada técnica). El ordenamiento en la mayoría en el caso promedio tendrá baja complejidad ya que hay un limitado número de técnicas por artistas.

Funciones del modelo realizadas para este requerimiento:

Función	Categoría	Explicación[4]
tecnicasObrasPorArtista	Función de consulta	Entrega una lista de listas en donde cada elemento de la lista primera es una lista de obras de una sola ténica, esta lista está ordenada por tamaño. Además retorna el número total de las obras del artista.
cmpFunctionTecnicasArtistas	Función de comparación	Compara dos técnicas por su tamaño, si. *Peor caso: O(k)*
sortList	Función de ordenamiento	Función global de ordenamiento, en este caso se utilizo MergeSort para

[4] En la documentación del model está una explicación detallada de cada una de las funciones.

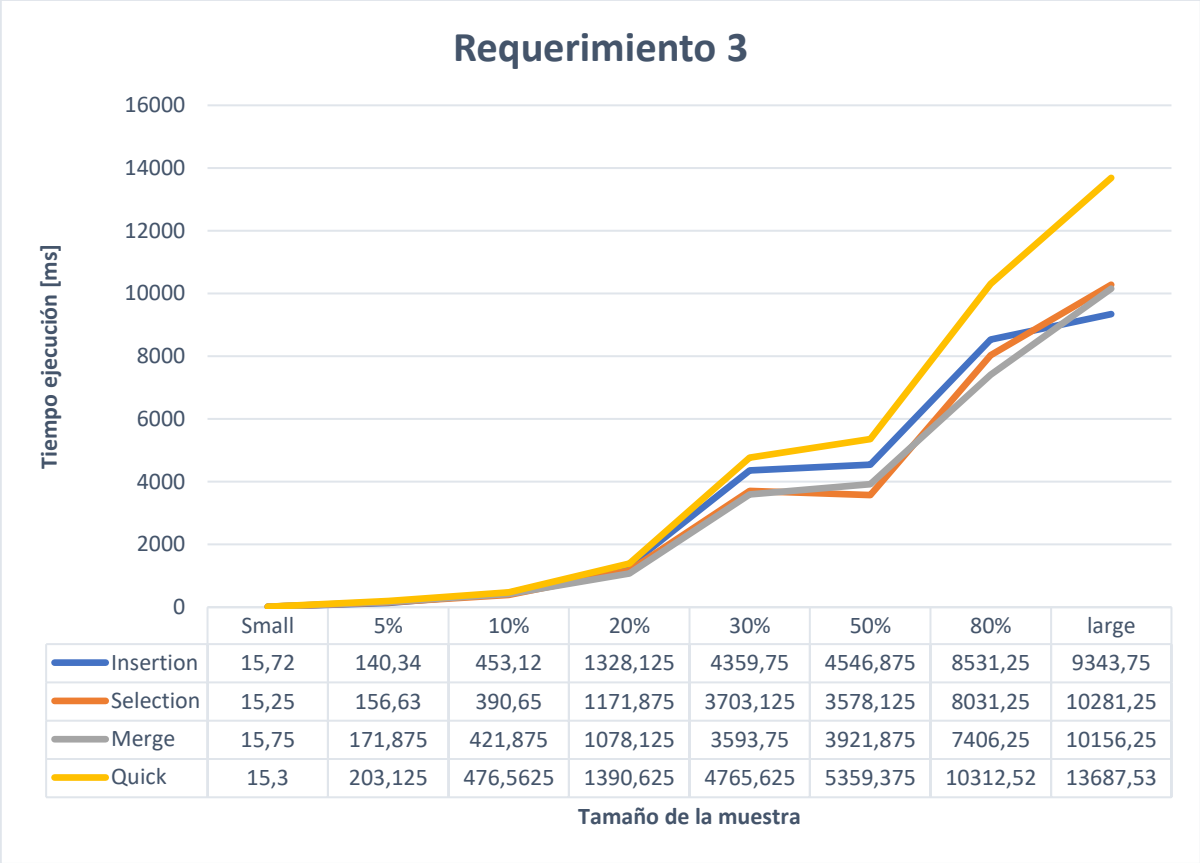
		los ordenamientos. *Peor Caso O(Nlog(N))*

Análisis de Complejidad

Caso	Complejidad	Explicación
Mejor Caso	O(N)	El mejor caso sucede cuando el primer artista es encontrado en la primera posición de la lista de artistas. Independiente de si encuentra o no obras que correspondan con su id, el recorrerá todas las obras así que su complejidad como mínimo será O(N)
Peor Caso	O(NlogN)	El peor caso es cuando todas las obras registradas son del artista ingresado y además cada obra tiene su propia técnica. Se deja la complejidad de orden mayor que será la del ordenamiento MergeSort

Pruebas de rendimiento

Todas las pruebas se realizaron ingresando el nombre del ejemplo del enunciado: Louise Bourgeois. Aunque si esperaba un mejor rendimiento en este método, pues en el caso promedio la lista a organizar es muy pequeña respecto al tamaño de la muestra, si me sorprendió que los cambios entre un tamaño de 80% y large es de segundos.



	Small	5%	10%	20%	30%	50%	80%	large
Insertion	15,72	140,34	453,12	1328,125	4359,75	4546,875	8531,25	9343,75
Selection	15,25	156,63	390,65	1171,875	3703,125	3578,125	8031,25	10281,25
Merge	15,75	171,875	421,875	1078,125	3593,75	3921,875	7406,25	10156,25
Quick	15,3	203,125	476,5625	1390,625	4765,625	5359,375	10312,52	13687,53

Figura 4. Pruebas de rendimiento requerimiento 3

Requerimiento 4 | Hecho por Jenifer

Explicación general:

Los problemas que encontré al resolver en esta función fueron:

1. Relacionar el constituentID de cada una de las obras con su respectivo autor

   Nota: En muchas de las obras este dato contenía a más de un autor, así que había tener en cuenta estos casos.

a. ¿Cómo se resolvió?

1. Usé la función TAD de iterator() para recorrer el catálogo de autores, después de eso utilicé la función TAD getElement() para conseguir la nacionalidad según el constituentID.
2. Realicé un slice al tener un constituentID que tuviera más de un autor. A partir los elementos que contuviera este slice se usaba el método mencionado anteriormente.

[Implementado en Funciones: getNationality y parte de req4]

2. Categorizar obras por nacionalidades.
    a. ¿Cómo se resolvió?

Cree una nueva lista tipo array list llamada countries. Cada uno de sus elementos es un diccionario que continene a las llaves “Nationality” y “Artworks”. El dato que contiene “Nationality” es una nacionalidad en especifico, “artworks” es un array list que contiene todas las obras de esa nacionalidad.

i. Principales TADS utilizados:
    1. newList(“ARRAY_LIST”, cmpNationalities)
        a. Donde cmpnationalities es una función de comparación hecha para acceder a la llave de nacionalidades
    2. lt.isPresent() para saber si la nacionalidad ya está dentro dentro de la lista de nacionalidades
    3. lt.getElement() en caso de que una nacionalidad ya esté dentro de la lista y se tenga que agregar un nuevo dato
    4. lt.addLast() para añadir un elemento a la última posición. Ya sea en la lista de countries, o en las obras de arte de una nacionalidad en especifico.

Nota: La obra podía estar repetida en una nacionalidad, o en distintas nacionalidades debido a que había casos en donde esta tenía más de un autor.

[Implementado en Funciones: NewNationalityArt, addNationality, compareNationalities y parte de req4]

3. Ordenar esta categorización de nacionalidades y obras de una manera eficiente
    a. ¿Cómo se resolvió?

Después de haber categorizado las obras en la lista de countries, utilicé la insertion sort junto la función de comparación compareNationalities

[Implementado en Funciones: cmpNationalitiesSize y parte de req4]

Lo anterior fue implementada en distintas funciones que hice para este requerimiento. En la siguiente sección se presenta una breve descripción de cada una de ellas.

Funciones realizadas para este requerimiento:

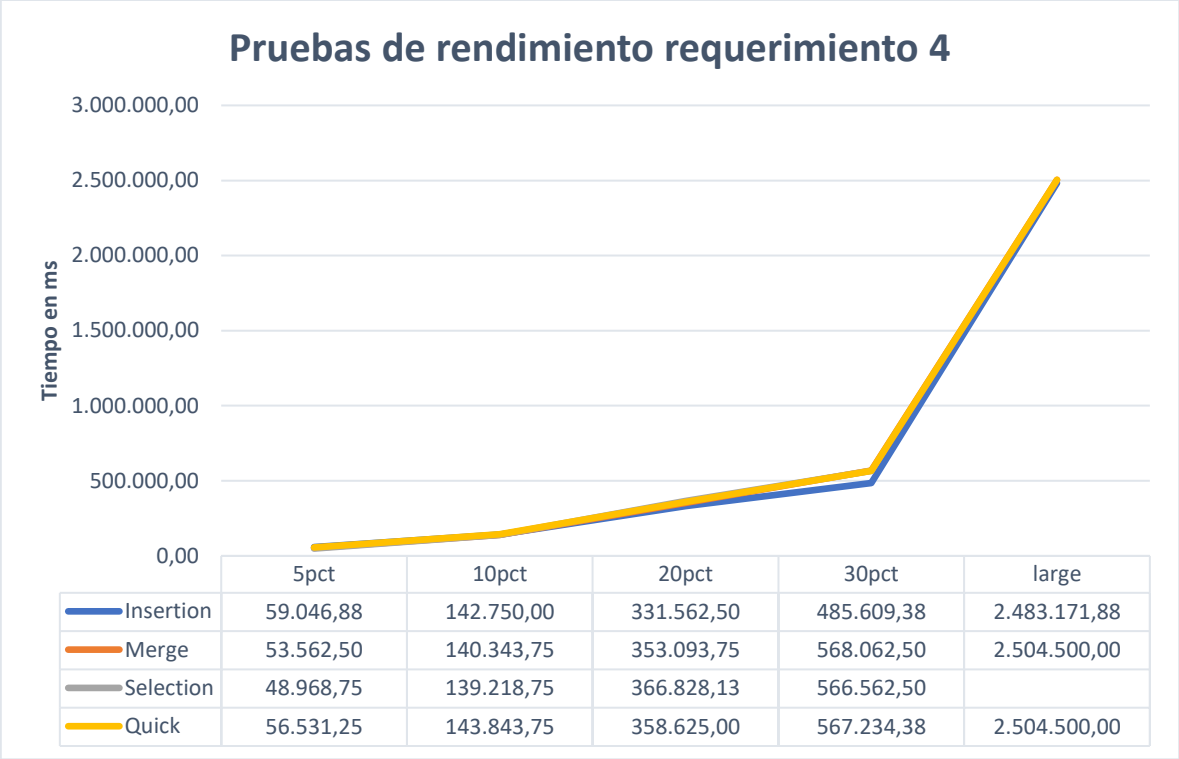
Función	Categoría	Explicación[5]
getNationality	Función de consulta	La función getNationality retorna la nacionalidad dependiendo el constituentID de la obra de arte. Se toma como unknown cuando la nacionalidad no existe o es una cadena vacía
NewNationalityArt	Función de creación	Crea un nuevo diccionario que tiene como llaves la nacionalidad y artworks. Donde artworks es un array_list con las respectivas obras de la nacionalidad
addNationality	Función para agregar información	Añade una nacionalidad a la lista de countries en la función principal
compareNationalities	Función de comparación	Función de comparación para acceder a las llaves al hacer la array_list de countries en la función principal
cmpNationalitiesSize	Función de compración	Función de comparación para ordenar los países con mayor cantidad de obras a menor cantidad de obras
ClasificarObrasNacionalidad	Función principal	La función principal creará una nueva lista llamada countries. Cada uno de sus elementos es una diccionario que representa a una

[5] En la documentación del model está una explicación detallada de cada una de las funciones.

		nacionalidad y sus obras. La función ordenará con insertion de mayor a menor cantidad de obras.

Tabla 2. Funciones hechas para el requerimiento 4

Pruebas de rendimiento:



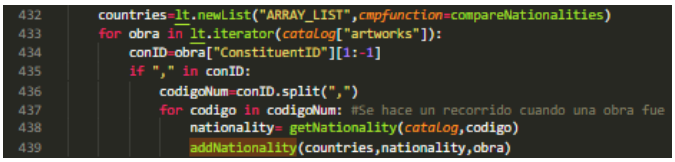
	5pct	10pct	20pct	30pct	large
Insertion	59.046,88	142.750,00	331.562,50	485.609,38	2.483.171,88
Merge	53.562,50	140.343,75	353.093,75	568.062,50	2.504.500,00
Selection	48.968,75	139.218,75	366.828,13	566.562,50	
Quick	56.531,25	143.843,75	358.625,00	567.234,38	2.504.500,00

Figura 5. Pruebas de rendimiento requerimiento 4. Computador con 64gb de Ram

La función tuvo tiempos similares al aplicar cualquier tipo de ordenamiento. A pesar de esto, insertion posee un mejor tiempo en muestras más grandes. Por lo cual, decidí dejar un ordenamiento por este tipo de algoritmo, además que posee características como su ordenamiento inPlace y su estabilidad.

Análisis de complejidad:

Función	Complejidad
getNationality	Mejor caso O(k): El artista y su nacionalidad se encuentran de primeras. Peor caso O(N): El artista se encuentra de últimas o no se encuentra al artista.
NewNationalityArt	O(k) dado que se crea un diccionario

addNationality	O(N): Peor caso cuando existe la nacionalidad en la lista, pero se encuentra en la última posición O(k) en caso de que la nacionalidad aún no exista y se necesite agregar a la última posición de la lista de countries
compareNationalities	[Función de comparación en arraylist] Usado para acceder a llaves dentro de la lista de countries. Peor caso O(N), mejor caso O(k)
cmpNationalitiesSize	[Función de comparación] O(k) dado que se comparan dos valores.
ClasificarObrasNacionalidad	[Primera parte]  Mejor caso: O(N): Al recorrer el catálogo de artworks ninguna de las obras tiene más de un artista. Peor caso: O(N^2): Al recorrer el catálogo todas las obras tienen más de un artista [Segunda parte (ordenamiento con insertion)]  Mejor caso: O(N) Peor caso: O(N^2)

```
countries=lt.newList("ARRAY_LIST",cmpfunction=compareNationalities)
for obra in lt.iterator(catalog["artworks"]):
    conID=obra["ConstituentID"][1:-1]
    if "," in conID:
        codigoNum=conID.split(",")
        for codigo in codigoNum: #Se hace un recorrido cuando una obra fue
            nationality= getNationality(catalog,codigo)
            addNationality(countries,nationality,obra)
```

```
    sortList(countries,cmpNationalitiesSize,sortType)
```

Requerimiento 5 | Grupal

Explicación general:

El requerimiento consistía en calcular el costo de transportar un departamento de obras. Para ello nos entregan el valor fijo en USD por $m^3|m^2|kg$, y el valor en USD en caso de no tener información de dimensiones ni peso.

Los retos de mayor complejidad que se encontraron fueron los siguientes:

- Manejo de diccionario en lista DISClib. Para poder mostrar el precio que costó cada obra tocaba editar el diccionario de cada obra en la misma estructura de DISClib.
- Calcular con precisión el valor de transportar un departamento.
- Optimizar el problema de tal manera que el tiempo de ejecución del algoritmo estuviera en un rango de tiempos razonable.

La solución para cada problema:

- Investigando dentro de la subida de los datos se encontró que es posible utilizar las llaves de diccionario en la obra para obtener e ingresar datos.
- Para establecer el valor de cada obra se calcularon los diferentes precios por m2 m3 y peso, después de hacer unas comprobaciones necesarias de los datos, se queda el más grande y se añade a la obra.
- Para reducir la complejidad al máximo primero se seleccionan las obras del departamento y se iban añadiendo a la lista con el precio calculado. Luego se organizaban las obras por precio y por antigüedad, logrando que el ordenamiento, la acción de mayor complejidad, no se efectuara en todas las obras sino únicamente en las obras del departamento.

Funciones del modelo realizadas para este req:

Función	Categoría	Explicación[6]
transportarObrasDespartamento	Función de consulta	Función que calcula el precio de transportar cada obra de un departamento
cmpArtworkByPrice	Función de comparación	Compara dos obras por su precio *Peor Caso: O(K)*
cmpArtworkByDate	Función de comparación	Compara dos obras por la fecha de realización *Peor Caso: O(K)*
sortList	Función de ordenamiento	Función global de ordenamiento, en este caso se utilizó MergeSort para los ordenamientos. *Peor caso: O(Nlog(N))*

[6] En la documentación del model está una explicación detallada de cada una de las funciones.

Análisis de Complejidad

Caso	Complejidad	Explicación
Mejor Caso	O(N)	El mejor caso es que no se encuentren obras. Por lo que solo existirá un recorrido O(N) por la obras. Ahora bien esto es poco probable ya que se seleccionan por rango de fecha así que el método usualmente tendrá que organizar grandes cantidades de obras.
Peor Caso	O(NlogN)	El peor caso es que todas las obras se deban transportar. Asi que se debe efectuar un ordenamiento en todas las obras, como se utiliza MergeSort el resultado es independiente de si la lista está o no organizada.

Pruebas de Rendimiento

Para las pruebas se utilizó el siguiente rango de fechas 1000-01-01 a 2000-01-01, aunque sabemos que al aumentar la muestra la cantidad de obras que se ordenan no necesariamente está aumentando linealmente, aun así se decidió esté rango ya que es suficientemente amplio

para que por lo menos exista un aumento cerca de ser proporcional y no un aumento nulo en el caso de utilizar una fecha muy reducida.

	Small	5%	10%	20%	30%	50%	80%	large
Insertion	750	83843,75	328171,875	1070195,313	1812218,75	3238414,063	4.664.609,3	
Selection	93,75	1515,625	2890,625	6132,8125	11343,75	18218,75	27656,25	38218,75
Merge	125	1109,375	2296,875	5046,875	6187,5	11468,75	17609,375	24810,5
Quick	140,625	1578,125	4984,375	14609,375	30343,75	78531,25	75859,375	

Figura 6. Pruebas de rendimiento requerimiento 5

Requerimiento 6 | Grupal

Explicación general:

El requerimiento nos pedía proponer obras para una exposición por épocas para el museo dependiendo de un área disponible en metros cuadrados. La característica que resaltaba de estas obras es que tenían que ser un objeto plano. Por lo cual, definimos a objeto plano como un objeto que tuviera un ancho o largo, pero no profundidad. Esto para evitar que el relieve de un objeto influyera negativamente en la exposición.

Al momento de pensar en el requerimiento notamos que para optimizar mejor el espacio se podía “cortar una obra”, de esta manera se puede lograr completar el área total propuesta por el museo. Por lo cual, uno de los parámetros principales de la función (obraCortadaB) nos dice si el museo acepta cortar una de las obras para lograr el máximo en la ocupación del área.

Por último, notamos que algunas obras poseen un ancho o largo vacío, por lo cual tomamos la altura o ancho en 1 metro cuando uno de estos datos es vacío. Esta asignación de un metro solamente se hará en caso de que se cumpla nuestra definición de "objeto plano", para que una de sus dimensiones sea por defecto 1 metro tiene que tener un ancho o un largo previamente.

Funciones realizadas para este requerimiento:

Función	Categoría	Explicación
expoEpocaArea	Función principal	Se proponen unas obras dependiendo del rango de fechas deseado por el museo. Se usa un ciclo con while que comprueba cuando se completa el área disponible, o en su defecto, cuando se acaban las obras por analizar.

Pruebas de rendimiento:

Las pruebas de rendimiento se realizaron con los parámetros:

- fechaInicialSt: 1001
- fechaFinalSt:2020
- obraCortadaB: True
- areaExpo: Va incrementando acorde a la muestra de datos. Si la muestra es de 5% el área será 5 metros cuadrados, si es 20% es 20 metros cuadrados.

Tamaño de muestra	**Área (m^2)**	**Tiempo de duración (ms)**
5pct	5	171,875
10pct	10	281,25
20pct	20	156,25
30pct	30	265,625
50pct	50	218,75
80pct	80	156,25

large	100	156,25

Tabla 3. Pruebas de rendimiento - Área utilizada

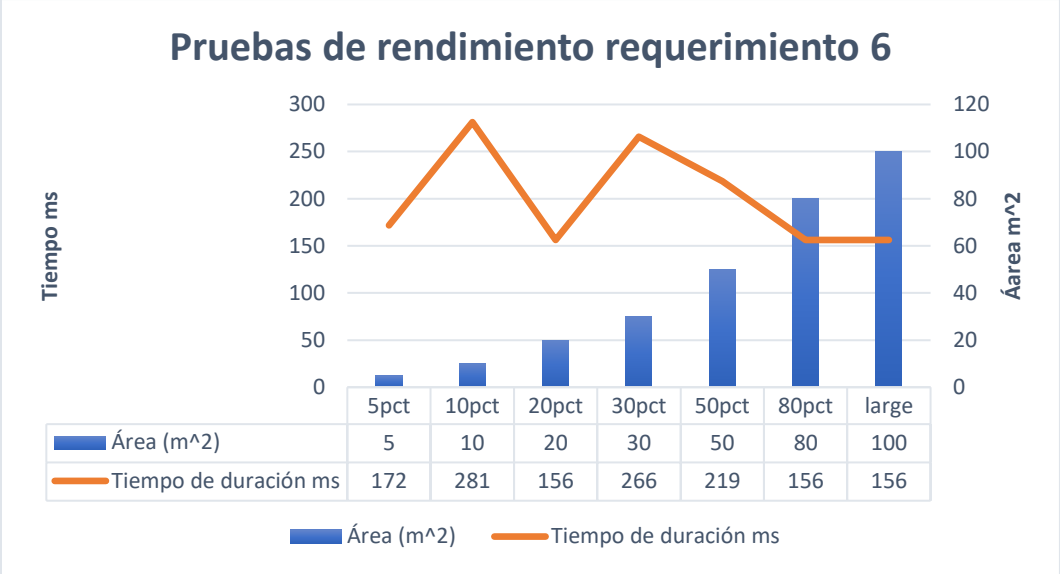


Figura 7. Pruebas de rendimiento en relación con el área y muestra

Análisis de complejidad:

Función	Complejidad
expoEpocaArea	Mejor caso: O(k) la primera obra cumple con el área exacta requerida por el ejercicio y/o el museo acepto cortar la obra para cumplir los requisitos del área. Peor caso: O(N) se recorre todo el catalogo de obras de arte dado que no se alcanza a llenar el área disponible.