Sustentación Reto 1-Estructura de Datos y Algoritmos

Sofía Escobar 202021047, s.escobart@uniandes.edu.co
Carolina Chinchilla 202011842, m.chinchilla@uniandes.edu.co

Antes de empezar con cada requisito es importante explicar la escogencia en la carga de datos, en esta se eligió cargar todos los datos en un ARRAY_LIST ya que para las funciones que se iban a implementar esta estructura de datos disminuía el tiempo de ejecución. La decisión se tomo después de las pruebas de rendimiento que se realizaron para el laboratorio 3, de estas pruebas también se decidió según los tiempos de ejecución que se iba a usar el recurso de ordenamiento Merge porque es el que menor tiempo de ejecución tenia con una ARRAY_LIST.

Requerimiento 1: En el requerimiento 1 se recibe un rango de fechas (año inicial y año final), se debe retornar el numero de artistas en el rango y los primeros y los últimos tres artistas en el rango. Esto se hizo por medio de 6 funciones que siguen la siguiente secuencia de pasos:

1. Ordenar la lista cronológicamente según el año de nacimiento del artista
2. Cortar la lista para tomar solo el fragmento que esta en el rango de fechas
3. Se crearon dos funciones para primeros y últimos 3, se nombran dos variables con los resultados de implementar ambas variables
4. Se uso una función para imprimir únicamente los datos que están en el requisito.

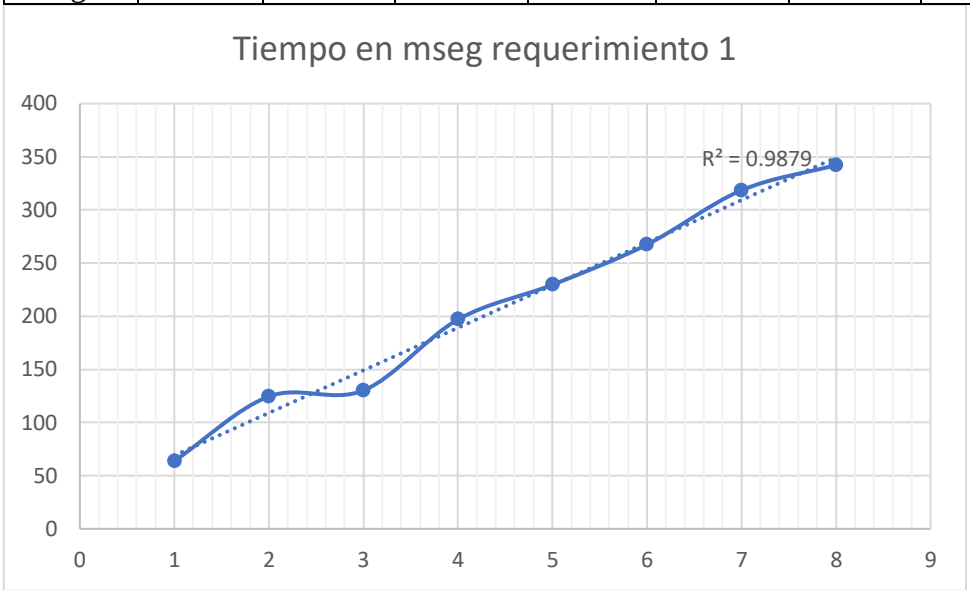
Funciones:

```
sortArrayListArtistMerge(lista)
fechasRangoArtist(listaf, fechai, fechaf)
darUltimosArtistas(lista_final)
darPrimerosArtistas(lista_final)
imprimirDatosArtista(primeras)
cmpArtistByDateBirth(artista1, artista2)
```

Para el primer paso se usa SortArrayListArtistMerge que recibe la lista de artistas y la ordena usando el criterio de comparación cmpArtworkByDate Se decidio este método de ordenamiento porque en el peor caso es linearitmico O(n(log n)) en su peor caso mientras que otros ordenamientos como QuickSort, InsertionSort o SelectionSort son cuadráticos en su peor caso y ShellSort es de orden O(n^3/2). Sabiendo esto, el ordenamiento que maneja de manera mas rápida grandes datos de ARRAY_LIST es MergeSort. Para el segundo paso se usa la función fechasRangoArtist que recibe el rango de años y la lista ordenada, en este caso la función crea una sublista ARRAY_LIST a la cual se añade el artista si este nacio en el rango de fechas usando la función addLast ya que esta tiene menor complejidad temporal que el addFirst en un ARRAY_LIST y la función retorna una lista con los artistas ordenados en el rango de fechas. Para el tercer paso se crearon dos funciones que crean una sublista usando los subíndices del ARRAY_LIST con la función subList y que retornan respectivamente una lista con los primeros tres artistas y una lista con los últimos 3 artistas. En el ultimo paso se usa la función imprimirDatosArtista para imprimir únicamente la información pedida y limitar los diccionarios que se van a imprimir.

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	63.82	124.527	130.196	196.89	229.599	267.398	318.43	342.173



El r^2 es lineal.

```
elif int(inputs[0]) == 2:
    articulo= 'artistas'
    lista= museo[articulo]
    fechai= input('Inserte el año inicial en el formato AAAA   ')
    fechaf= input('Inserte el año final en el formato AAAA    ')
    start_time = time.process_time()#O(K)
    z= controller.sortArrayListArtistMerge(lista) #linearitmica O(N(logN))

    lista_final= controller.fechasRangoArtist(z, fechai, fechaf) #Lineal O(N)
    stop_time = time.process_time() #O(K)
    elapsed_time_mseg = (stop_time - start_time)*1000 #O(K)
    ultimas=controller.darUltimosArtistas(lista_final) #O(K)
    primeras=controller.darPrimerosArtistas(lista_final)#O(K)

    print("Artistas nacidos en el rango de fechas:   "+ str(lt.size(lista_final)))
    print('Primeros tres artistas:  ')
    imprimirDatosArtista(primeras)
    print('Ultimos tres artistas:   ')
    imprimirDatosArtista(ultimas)
    print('El ordenamiento tomo   '+ str(elapsed_time_mseg)+ '  tiempo en mseg')
```

El algoritmo en este requisito es linearitmico porque el mayor orden de complejidad es el de O(N(logN)).

Requerimiento 2: En este requerimiento se recibe un rango de fechas (fecha inicial y fecha final), se debe retornar el numero de obras en el rango, la cantidad de obras compradas en el rango y las primeras y las últimas tres obras en el rango. Esto se hizo por medio de 7 funciones que siguen la siguiente secuencia de pasos:

1. Ordenar la lista cronológicamente según el año de compra de la obra
2. Cortar la lista para tomar solo el fragmento que esta en el rango de fechas
3. Se contaron la cantidad de obras en este fragmento que se adquirieron por medio de compra
4. Se crearon dos funciones para primeros y últimos 3, se nombran dos variables con los resultados de implementar ambas variables
5. Se uso una función para imprimir únicamente los datos que están en el requisito.

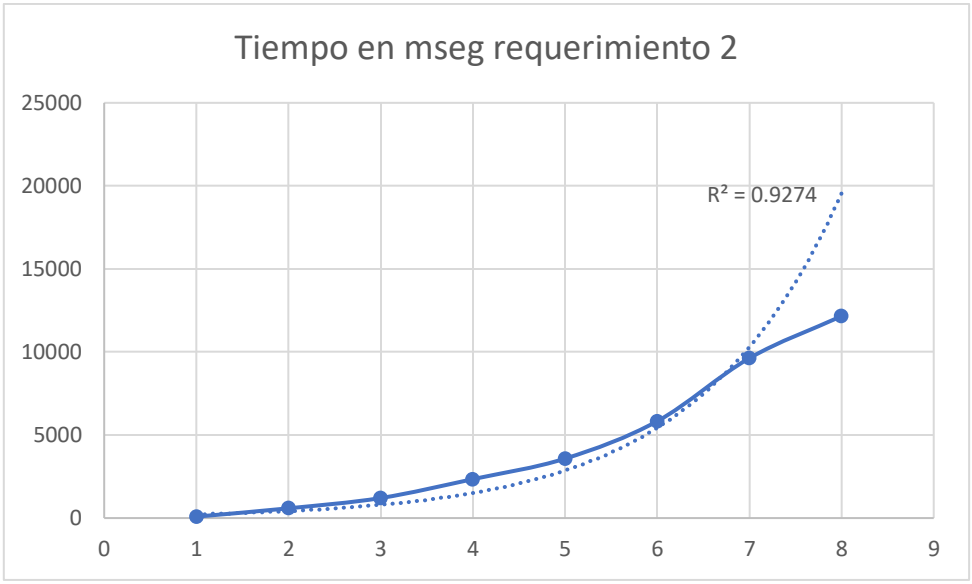
Funciones:

```
sortArrayListMerge(lista)
fechasRango(z, fechai, fechaf)
darPrimerasObras(lista_final)
darUltimasObras(lista_final)
obrasPurchase(lista_final)
imprimirDatosObra(primeras)
cmpArtworkByDateAcquired(artwork1, artwork2)
```

Para el primer paso se usa SortArrayListMerge que recibe la lista de obras y la ordena usando el criterio de comparación cmpArtworkByDateAquired. Se decidio este método de ordenamiento porque en el peor caso es linearitmico O(n(log n)) en su peor caso mientras que otros ordenamientos como QuickSort, InsertionSort o SelectionSort son cuadráticos en su peor caso y ShellSort es de orden O(n^3/2). Sabiendo esto, el ordenamiento que maneja de manera mas rápida grandes datos de ARRAY_LIST es MergeSort. Para el segundo paso se usa la función fechasRango que recibe el rango de años y la lista ordenada, en este caso la función crea una sublista ARRAY_LIST a la cual se añade la obra si esta fue adquirida en el rango de fechas usando la función addLast ya que esta tiene menor complejidad temporal que el addFirst en un ARRAY_LIST y la función retorna una lista con las obras ordenadas en el rango de fechas. Para el tercer paso se usa la función obrasPurchase que hace una comparación y cuenta la cantidad de obras en el rango que fueron compradas.Para el cuarto paso se crearon dos funciones que crean una sublista usando los subíndices del ARRAY_LIST con la función subList y que retornan respectivamente una lista con las primeras tres obras y una lista con las últimas 3 obras. En el ultimo paso se usa la función imprimirDatosObra para imprimir únicamente la información pedida y limitar los diccionarios que se van a imprimir.

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	86.06	587.68	1206.44	2343.62	3579.00	5830.32	9626.98	12155.15



El r^2 es exponencial.

```
elif int(inputs[0]) == 3:
    articulo= 'obras'
    lista= museo[articulo]
    fechai= input('Inserte la fecha inicial en el formato AAAA-MM-DD   ')
    fechaf= input('Inserte la fecha final en el formato AAAA-MM-DD    ')
    start_time = time.process_time()#O(K)
    z= controller.sortArrayListMerge(lista)#O(N(logN))
    lista_final= controller.fechasRango(z, fechai, fechaf)#O(N)

    stop_time = time.process_time()#O(K)
    elapsed_time_mseg = (stop_time - start_time)*1000#O(K)
    primeras=controller.darPrimerasObras(lista_final)#O(K)
    ultimas=controller.darUltimasObras(lista_final)#O(K)


    print("Aquisiciones en el rango de fechas:  "+ str(lt.size(lista_final)))
    print("Obras adquiridas por compra:  " + str(controller.obrasPurchase(lista_final)))
    print('Primeras tres obras: ')
    imprimirDatosObra(primeras)
    print('Ultimas tres obras:  ')
    imprimirDatosObra(ultimas)
    print('El ordenamiento tomo   '+ str(elapsed_time_mseg)+ '  tiempo en mseg')
```

El algoritmo en este requisito es linearitmico porque el mayor orden de complejidad es el de O(N(logN)).

Requerimiento 3: En este requisito se recibe un nombre de un artista, se debe retornar el numero de obras del artista, la cantidad de técnicas que uso este, la técnica mas usada y todas las obras de el artista realizadas con esta tecnica. Esto se hizo por medio de 7 funciones que siguen la siguiente secuencia de pasos:

1. Encontrar el constituent ID del artista
2. Buscar todas las obras que de este artista usando el constituent ID

3. Hacer una lista de todas las técnicas usadas
4. Contar cuantas veces se usa cada técnica
5. Encontrar la técnica mas frecuente y cuantas veces se usa
6. Clasificar las obras que usan una técnica especifica para obtener las obras de la técnica mas frecuente
7. Se uso una función para imprimir únicamente los datos que están en el requisito.

Funciones:

```
artistaID(museo, nombre)
obrasID(museo, id)
listarTecnicas(obras)
contarTecnicas(tecnicas)
tecnicaMasFrecuente(listaT)
clasificarObrasPorTecnica(obras, lt.firstElement(tecnicaMasFrecuente))
imprimirDatosObra2(obrasTecnica)
```

Para el primer paso se usa artistaID que recibe la lista de artistas y el nombre del artista para poder encontrar y retornar el ID del mismo. Para el segundo paso se utiliza obrasID que recibe la lista de obras y el ID del artista para poder hacer una sublista con todas las obras de este artista. En el tercer paso de usa el listar técnicas que busca hacer otra sublista con las técnicas usadas para después poder contarlas en el cuarto paso (contarTecnicas) y seleccionar la mas frecuente en el quinto paso (tecnicaMasFrecuente). En el sexto paso se hace una sublista con la sobras del artista que usan la técnica mas frecuente y al final se usa una función para imprimir solo lo necesario de las obras.

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	4.34	20.66	33.25	58.69	75.12	96.46	124.40	152.52

El r^2 es lineal.

```
elif int(inputs[0]) == 4:
    nombre= input('Ingrese el nombre del artista que desea consultar  ')
    start_time = time.process_time()#O(K)
    id= controller.artistaID(museo, nombre)#O(N)
    obras = controller.obrasID(museo, id)#O(N)
    numero= lt.size(obras)#O(1)
    tecnicas=controller.listarTecnicas(obras)#O(N)
    listaT = controller.contarTecnicas(tecnicas)#O(N)
    tecnicaMasFrecuente=controller.tecnicaMasFrecuente(listaT)#O(N)
    obrasTecnica= controller.clasificarObrasPorTecnica(obras, lt.firstElement(tecnicaMasFrecuente))#O(N)
    stop_time = time.process_time()#O(K)
    elapsed_time_mseg = (stop_time - start_time)*1000#O(K)

    print('Hay  '+str(numero)+'  obras del artista  '+nombre)
    print('El artista usa '+ str(len(listaT.keys())) +' tecnicas')
    print('La tecnica mas utilizada es: '+ lt.getElement(tecnicaMasFrecuente,1) + " , con un total de:  " + str(lt.getElement(tecnicaMasF
    print('Las obras que utilizan  '+ lt.firstElement(tecnicaMasFrecuente) +' son:')
    imprimirDatosObra2(obrasTecnica)
    print('El ordenamiento tomo   '+ str(elapsed_time_mseg)+ '  tiempo en mseg')
```

El algoritmo en este requisito es lineal porque el mayor orden de complejidad es el de O(N), lo cual coincide con lo evidenciado en la gráfica.

Requerimiento 5: Para este requerimiento se buscaba calcular el precio de transportar las obras de un departamento, por ende, se recibia el departamento del museo y el retorno incluía la cantidad de obras por transportar, el precio, el peso, las 5 obras mas antiguas y las 5 obras mas caras. Para esto se usaron x funciones que siguen los siguientes pasos:

1. Primero se filtran las obras para que queden solo las que pertenecen al departamento en cuestión y se crea una sublista que contenga solo estas obras
2. Despues se calcula el precio de cada una y se adiciona a cada obra una llave ‘Costo’ que contiene como valor el precio de transportar la obra
3. Despues se calcula el precio total sumando lo que vale transportar cada obra
4. Se ordenan según fecha y se seleccionan las ultimas 5 que son las mas antiguas
5. Se ordenan según costo y se seleccionan las primeras 5 que son las mas costosas
6. Después se suma el peso de todas las obras
7. Para finalizar se limita el diccionario de las obras que se van a imprimir para que solo contenga la información pertinente

Funciones:

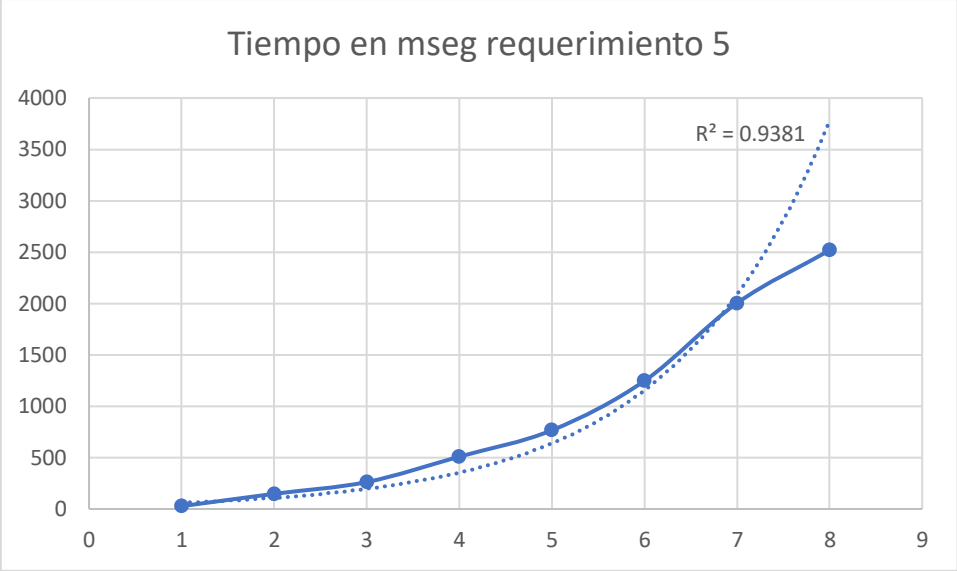
```
obraDepartamento(museo, departamento)
precioObra(obras)
sortArrayListMergeDate(obras)
darUltimasObras5(obras)
sortArrayListMergeCost(obras)
darPrimerasObras5(obrasP)
pesoObra(obras)
sumaPrecios(obras)
imprimirDatosObra4(antiguas)
```

Para el primer paso se usa obraDepartamento que recibe la información del museo y el departamento que deseamos trasladar, esta retorna una lista ARRAY_LIST que contiene la información de todas las obras que pertenecen a cada departamento. A continuación se uso la función precioObra para el segundo paso que no retorna nada pero si modifica la lista con

las obras. Adicionalmente se uso sortArrayListMergeDate y cmpArtworkByDate para ordenar la lista según fecha y con darUltimasObras5 se sacaron las obras mas antiguas. El proceso se repitió con sorArrayL.istMergeCost y y cmpArtworkByCost pero se escogieron las primeras 5 obras porque son las mas caras. Después se sumaron los costos y los pesos. Al final se corta la información de las obras para contener solo lo pertinente. En este caso se eligió Merge ya que según las pruebas que se hicieron era el método de ordenamiento que realizaba el trabajo mas rápido ya que como se discutió en los anteriores requerimientos la complejidad temporal de este ordenamiento es menor a las de los otros.

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	27,48	147,26	262,10	509,48	765,92	1246,77	2003,52	2517,38



El r^2 es exponencial.

```
elif int(inputs[0]) == 6:
    departamento= input('Inserte el departamento en el que se desea realizar el analisis   ')
    start_time = time.process_time()#O(K)
    obras= controller.obraDepartamento(museo, departamento)#O(N)
    controller.precioObra(obras)#O(N)
    obrasA=controller.sortArrayListMergeDate(obras)#O(N(logN))
    antiguas= controller.darPrimerasObras5(obras)#O(K)
    obrasP=controller.sortArrayListMergeCost(obras)#O(N(logN))
    caras= controller.darUltimasObras5(obrasP)#O(K)
    peso= controller.pesoObra(obras)#O(N)
    precio=controller.sumaPrecios(obras)#O(N)
    stop_time = time.process_time()#O(K)
    elapsed_time_mseg = (stop_time - start_time)*1000#O(K)

    print("El total de obras a transportar es: " +str(lt.size(obras)))
    print("El estimado en USD del precio del servicio es " + str(precio))
    print ("El peso estimado de las obras es: " + str(peso))
    print("Las 5 obras mas antigua a transportar son: ")
    imprimirDatosObra4(antiguas)
    print("Las 5 obras mas costosas para transportar son: " )
    imprimirDatosObra4(caras)
    print('El ordenamiento tomo   '+ str(elapsed_time_mseg)+ '  tiempo en mseg')
```

El algoritmo en este requisito es linearitmico porque el mayor orden de complejidad es el de O(N(logN)).