Sustentación Reto 1-Estructura de Datos y Algoritmos
Sofía Escobar 202021047, s.escobart@uniandes.edu.co
Carolina Chinchilla 202011842, m.chinchilla@uniandes.edu.co

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.

RETO 1:

Esto se hizo por medio de 6 funciones que siguen la siguiente secuencia de pasos:

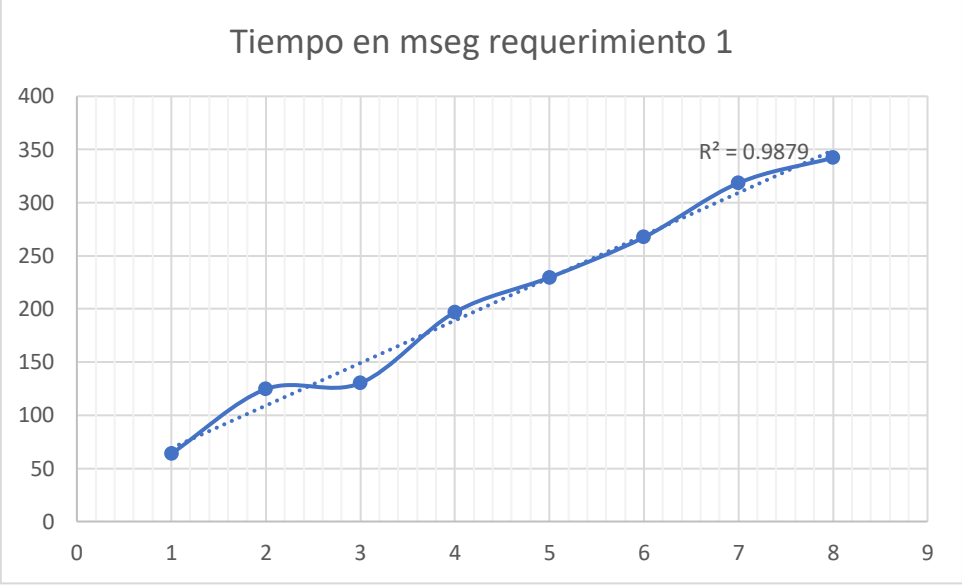
1. Ordenar la lista cronológicamente según el año de nacimiento del artista
2. Cortar la lista para tomar solo el fragmento que esta en el rango de fechas
3. Se crearon dos funciones para primeros y últimos 3, se nombran dos variables con los resultados de implementar ambas variables
4. Se uso una función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

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)).

RETO 2:

Esto se hizo por medio de 6 funciones que siguen la siguiente secuencia de pasos:

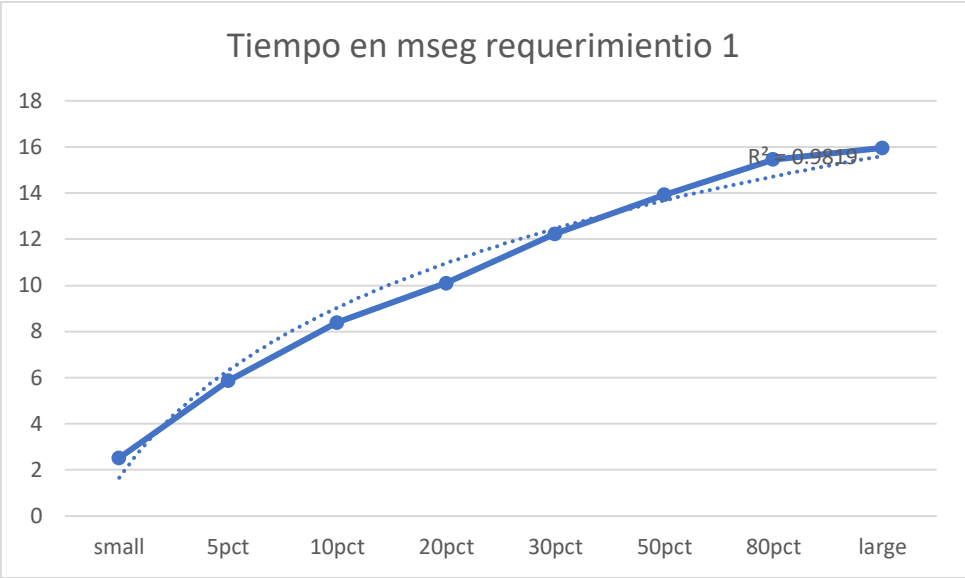
1. Se hace una selección de los artistas que pertenecen al rango de fechas sin necesidad de ordenar la lista, para esto se usa un mapa
2. Se crearon dos funciones para primeros y últimos 3, se nombran dos variables con los resultados de implementar ambas variables
3. Se uso una función para imprimir únicamente los datos que están en el requisito.

Funciones:

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

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	2,52	5,86	8,39	10,1	12,23	13,93	15,45	15,96



El R^2 es logarítmico.

```
elif int(inputs[0]) == 2:
    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)
    lista_final= controller.fechasRangoArtist(museo, fechai, fechaf) #Lineal O(N^2)
    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)
```

Podemos concluir que es mas eficiente en el reto 2 ya que toma menos tiempo el proceso porque no hay necesidad de ordenar la lista.

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.

RETO 1:

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.

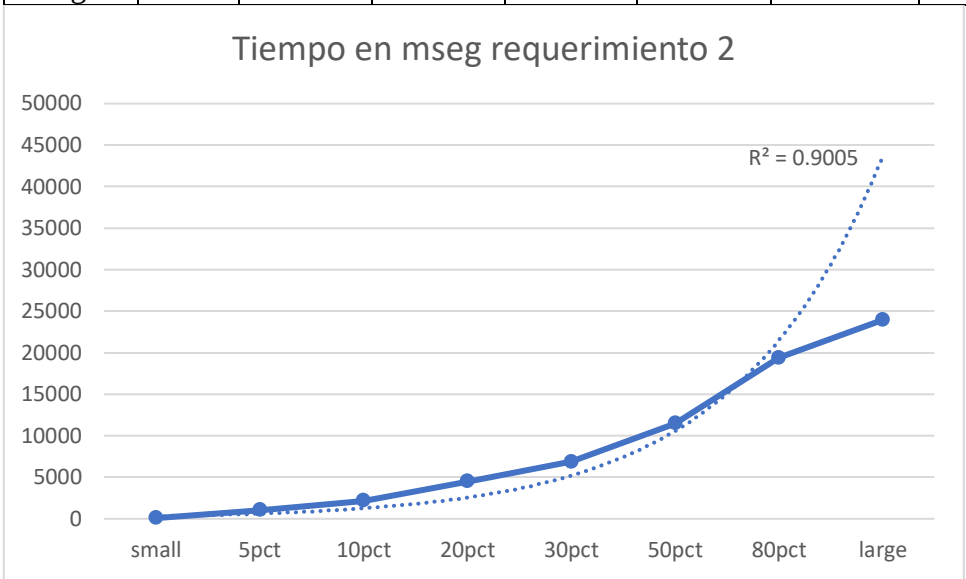
Funciones:

```
sortArrayListMerge(lista)
fechasRango(z, fechai, fechaf)
darPrimerasObras(lista_final)
```

```
darUltimasObras(lista_final)
obrasPurchase(lista_final)
imprimirDatosObra(primeras)
cmpArtworkByDateAcquired(artwork1, artwork2)
```

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	84,55	1052,51	2174,46	4494,23	6879,66	11462,88	19366,4	23937,14



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)).

RETO 2:

Esto se hizo por medio de 7 funciones que siguen la siguiente secuencia de pasos:

1. Se crea una lista a partir de los mapas cargados, esta lista contiene todas las obras
2. Se ordena la lista con un SortMerge
3. Cortar la lista para tomar solo el fragmento que esta en el rango de fechas
4. Se contaron la cantidad de obras en este fragmento que se adquirieron por medio de compra
5. Se crearon dos funciones para primeros y últimos 3, se nombran dos variables con los resultados de implementar ambas variables
6. Se uso una función para imprimir únicamente los datos que están en el requisito.

```
crearListaObras(museo
sortArrayListMerge(lista)
fechasRango(z, fechai, fechaf)
obrasPurchase(lista_final)
darPrimerasObras(lista_final)
darUltimasObras(lista_final)
```

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	85,02	1157,67	2415,99	6288,54	20637,33	80234,45	271124,88	406929,35

El R^2 es exponencial.

```
lista = controller.crearListaObras(museo)#O(N^2)
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)
```

En este caso es mas eficiente el trabajo realizado con listas ya que el manejo de las llaves para seleccionar un rango no es posible, por ende, los métodos para llegar a esto disminuyen en sobremanera la eficiencia en términos de tiempo de ejecución de este requerimiento.

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.

RETO 1:

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

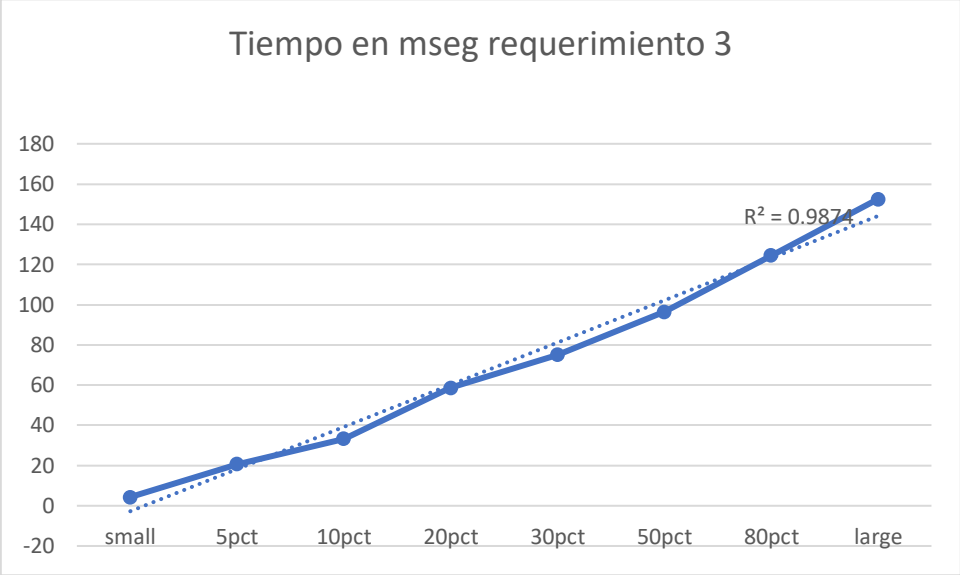
6. Clasificar las obras que usan una técnica especifica para obtener las obras de la técnica mas frecuente
7. Se uso una función para imprimir únicamente los datos que están en el requisito.

Funciones:

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

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.

RETO 2:

Esto se hizo por medio de 6 funciones que siguen la siguiente secuencia de pasos:

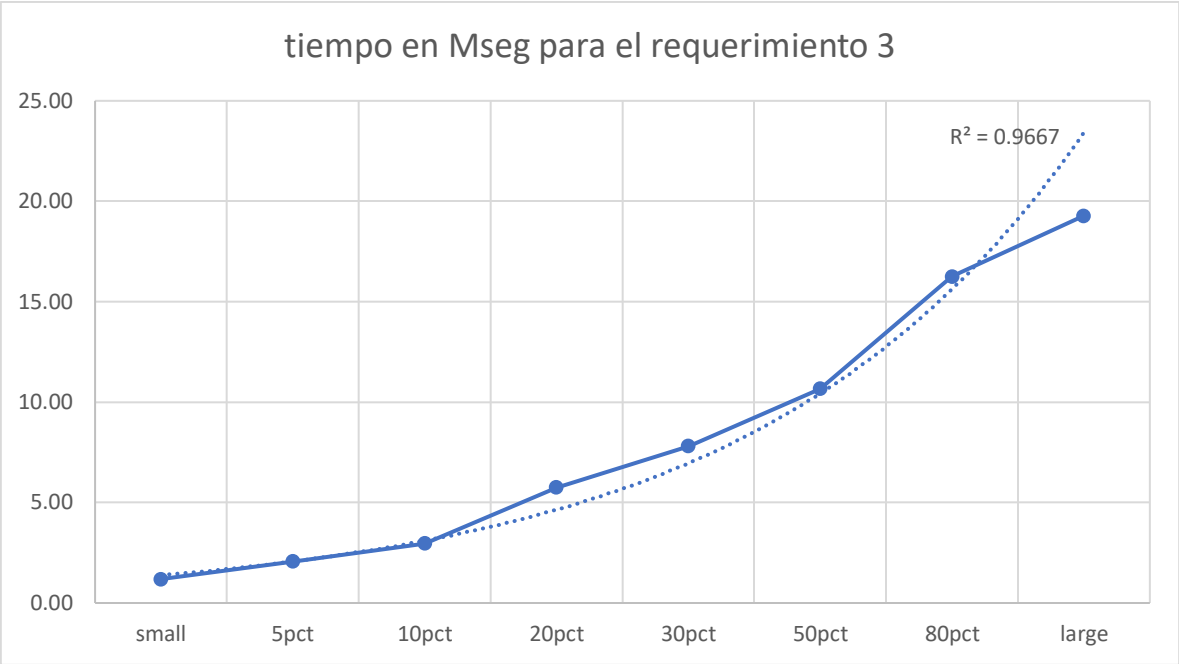
1. Encontrar el constituent ID del artista por medio de un mapa que tenia de llave el nombre del artista.
2. Buscar todas las obras que de este artista usando el constituent ID, esto se hace por medio de un mapa que tiene de llave el consitiuent ID y de valor la información de todas las obras asociadas a ese 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

Funciones:

```
getArtistaNombre(museo, nombre

obrasID(museo, id)

listarTecnicas(obras)

contarTecnicas(técnicas)

tecnicaMasFrecuente(listaT)

clasificarObrasPorTecnica(obras, lt.firstElement(tecnicaMasFrecuente))
```

Pruebas de tiempo de ejecución:

archivo	small	5pct	10pct	20pct	30pct	50pct	80pct
tiempo en mseg	1,19	2,06	2,97	5,75	7,81	10,67	16,25



El r^2 es exponencial.

```
    nombre= input('Ingrese el nombre del artista que desea consultar  ')
    start_time = time.process_time()#O(K)
    id= controller.getArtistaNombre(museo, nombre)#O(N)
    obras = controller.obrasID(museo, id)#O(K)
    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)
```

Con lo analizado anteriormente y con los tiempos de ejecución podemos concluir que es mas eficiente en términos de tiempo de ejecución el uso de mapas que el uso de listas.

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.

RETO 1:

Para esto se usaron 7 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

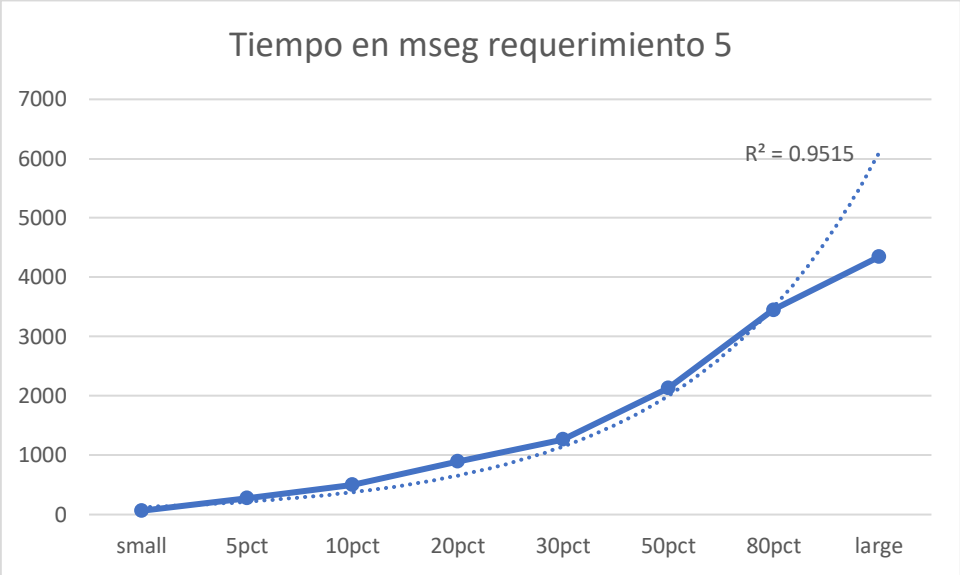
7. Para finalizar se limita el diccionario de las obras que se van a imprimir para que solo contenga la información pertinente

Funciones:

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

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large
Tiempo en mseg	64,36	278,53	499,02	894,98	1265,73	2133,01	3451,92	4343,2



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)).

RETO 2:

Para esto se usaron 7 funciones que siguen los siguientes pasos:

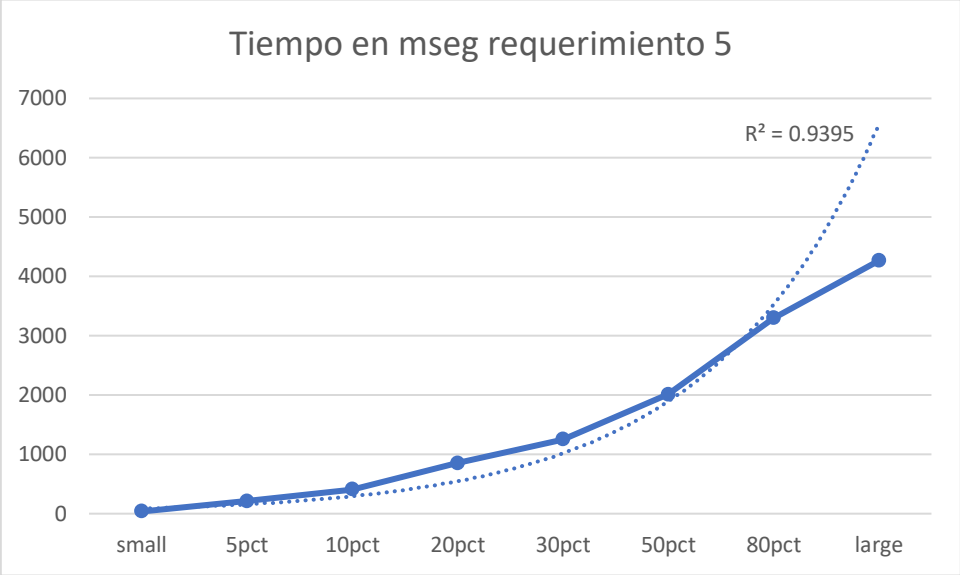
1. Se usa un mapa con el departamento de llave y las obras que pertenecen al mismo de valor para poder seleccionar de manera rápida todas las obras del departamento
2. Despues se calcula el precio de cada una y se adiciona a cada obra una llave ‘Costo’ que contiene como valor el precio de transportar 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

```
obraDepartamento(museo, departamento)#O(N)
precioObra(obras)#O(N)
sortArrayListMergeDate(obras)#O(N(logN))
darPrimerasObras5(obras)#O(K)
sortArrayListMergeCost(obras)#O(N(logN))
darUltimasObras5(obrasP)#O(K)
pesoObra(obras)#O(N)
sumaPrecios(obras)#O(N)
```

Pruebas de tiempo de ejecución:

Archivo	small	5pct	10pct	20pct	30pct	50pct	80pct	large

Tiempo en mseg	36,48	212,21	408,09	849,54	1250,87	2010,92	3294,0	4264,58



El R^2 es exponencial.

```
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(K)
print(lt.size(obras))
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)
```

Aunque la diferencia en tiempo no es muy grande, de igual forma es temporalmente más eficiente el requerimiento si se usan los mapas.