OBSERVACIONES & ANÁLISIS – RETO 2

Req3 - María Alejandra Estrada García Cod. 202021060

Req4 - Santiago Martínez Novoa Cod. 202112020

En el documento se presenta la comparación entre eficiencia de consumo de datos y tiempo de ejecución entre el reto 1 y reto 2.

La diferencia entre los dos retos radica en la carga de datos, pues se hizo uso del TAD map implementado en hash table con linear probing, porque consideramos que esas especificaciones de carga si bien son costosas en almacenamiento, reducen los tiempos de consulta (acción que más se realiza). El haber cargado los datos diferente en el reto 2 con relación al 1 significó un cambio en las funciones de consulta principales para cada requerimiento, sin embargo, todas las demás funciones que ayudaban a ordenar, comprobar existencia de entradas de usuario y presentar información en consola no sufrieron cambio alguno. Debido a esto, las pruebas de eficiencia presentadas más abajo solo se realizaron en las funciones de consulta principales.

Máquina entradas por cada req:

- Req 1: 1920 y 1985
- Req 2: 1944-06-06 y 1989-11-09
- Req 3: Various Artists
- Req 4: -Todas las obras-
- Req 5: Drawings & Prints

Complejidad de los requerimientos Reto 2:

Requerimiento 1: [En Grupo] Listar cronológicamente los artistas.

El primer requerimiento se reparte en una función principal, la cual es getArtistByDate(catalog, anoInicial, anoFinal). Es la función principal del requerimiento y es la cual se llama en el controller, el cual da como resultado la lista de cronológica de los artistas según un rango de fechas dado, sacado de una estructura de datos tipo tabla hash. En esta función se hace un ciclo para recorrer que estén dentro del rango y posteriormente por medio de get(...) se retorna el valor asociado al año y dentro hacemos un getValue(..) la cual nos da los valores de cada uno de los artistas. Luego, hacemos otro ciclo con for para sacar la lista de las aristas del paso anterior y añadimos cada artista dentro de la lista creada list_artistDate. En esta función se retorna la lista list_artistDate y el tiempo de ejecución. Debido a que se recorren en dos ciclos diferentes una los rangos de fechas y el otro el mapa de ArtistDate la función tiene complejidad de O(N*M).

Función principal del requerimiento:

```
#Req 1
#-------------------------------------------------------------------------------------------------------------------------
def getArtistByDate(catalog, anoInicial, anoFinal):
    start_time = time.process_time()

    list_artistDate = lt.newList('ARRAY_LIST', compArtistsDates)

    i = anoInicial
    while i >= anoInicial and i <= anoFinal:
        artist_value = mp.get(catalog['ArtistsDates'], str(i))
        if artist_value:
            list_artists= me.getValue(artist_value)
            for a in lt.iterator(list_artists['Artists']):
                lt.addLast(list_artistDate, a)

        i += 1

    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return list_artistDate, elapsed_time_mseg
```

Complejidad temporal final = O(N*M)

Requerimiento 2: [En Grupo] Listar cronológicamente las adquisiciones.

El segundo requerimiento se reparte en diversas funciones las buscan crear una función una lista cronológica de las adquisiciones. La función la cual se llama en el controller es la de getArtworksDate(catalog, inicial, final). En esta función se hace dos ciclos, uno para verificar que este dentro de un rango de fechas (año) y posteriormente, un ciclo de los valores de cada llave del mapa 'ArtworksDateAcquired', en especial de las obras en el rango de años y dentro de ese for (ciclo), se verifica que el DatAcquired de la obra esté dentro del rango completo YYYY-MM-DD inicial y final entregado como parámetro y se añaden a una lista tipo ARRAY_LIST que contenga todas las obras adquiridas en ese rango de fechas. Tiene una complejidad de O(N*M), lo cual posteriormente se organza con la función sortArtworkDateAcquired(list_artworkDateAcquired), con mergesort, para organizar los valores en la lista de menor a mayor. Esta función tiene una complejidad de O(Nlog(N)). Para la función getartworksPurchased(DatesArtworks) donde es de O(N) pues hay un ciclo que se ejecuta completamente cada vez que se llama a esa función. Para finalizar la complejidad sería O(NlogN).

Funciones principales del requerimiento:

```python
#Req 2:
#-----------------------------------------------------------------------------------------------------------------------
def getArtworksDate(catalog, inicial, final):
    start_time = time.process_time()

    list_artworkDateAcquired = lt.newList('ARRAY_LIST')

    inicialDate = date.fromisoformat(inicial)
    finalDate = date.fromisoformat(final)

    inicialSplit = inicial.split('-')
    finalSplit = final.split('-')

    i = int(inicialSplit[0])
    print(i)
    while i >= int(inicialSplit[0]) and i <= int(finalSplit[0]):
        #print(catalog['ArtworksDateAcquired'])
        artwork_value = mp.get(catalog['ArtworksDateAcquired'], i)
        if artwork_value:
            list_artwork= me.getValue(artwork_value)
            for a in lt.iterator(list_artwork['Artworks']):
                a1 = date.fromisoformat(a['DateAcquired'])
                if a1 >= inicialDate and a1 <= finalDate and a1 != '' and a1!='0':
                    lt.addLast(list_artworkDateAcquired, a)
        i += 1
```

```python
    sort_DateAcquired =  sortArtworkDateAcquired(list_artworkDateAcquired)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return sort_DateAcquired, elapsed_time_mseg
```

```python
def getartworkPurchased(datesArtworks):
    count = 0
    for a in lt.iterator(datesArtworks):
        if 'purchase' in a['CreditLine'].lower():
            count += 1

    return count
```

*Complejidad temporal final = O(N*M)*

Requerimiento 3: [Individual] Clasificar las obras de un artista por técnica.

El tercer requerimiento se reparte en diferentes funciones para poder clasificar las obras de un artista por técnica. Para esto se utilizó la función principal getArtworksMediumOneArtist(catalog, artistName). Se llama a la función getArtistIDbyName(catalog, artistName), la cual busca el ConstituentID del artista específico que tiene una complejidad de O(N). Luego busca el valor dentro del catalogo 'ArttworksofArtist', la cual busca las obras del artista ingresado. Posteriormente, se llama a la función getMediumOneArtist(ctalog, artworks), la cual crea el mapa de los medios de un artista con sus obras. Luego, se hace un ciclo con for para sacar cada medio dentro del mapa, y añade a la lista ArtistTecnique, con los medios del artista con sus obras y la cantidad de obras. Al final, se ordena la lista con la función sortByMedium(ArtistTecnique), para organizar de mayor a menor con mergesort y tiene una complejidad de O(NLogN). La función retorna al final la lista organizada y el tiempo que se demoró en ejecutar. Para finalizar la complejidad sería O(NLogN).

Funciones principales del requerimiento:

```
def getArtworksMediumOneArtist(catalog, artistName):
    start_time = time.process_time()
    ArtistTecnique = lt.newList('ARRAY_LIST', cmpfunction=compATecnique)
    artistID = getartistIDbyName(catalog, artistName) #Buscar el ID del artista específico
    artwork_value = mp.get(catalog['ArtworksofArtist'], artistID)
    if artwork_value:
        artworks= me.getValue(artwork_value)
        getMediumOneArtist(catalog,artworks) #Crear el mnapa de los medios de un artista con sus obras
        artwork_value2 = mp.keySet(catalog['ArtworkMedium'])
        for element in lt.iterator(artwork_value2):
            artist_value = mp.get(catalog['ArtworkMedium'], element)
            number_artworks= me.getValue(artist_value) #Lista de obras de ese medio en específico
            tuple_medium = {'Medium':element,'NumbArtworks':lt.size(number_artworks["Artworks"])}
            lt.addLast(ArtistTecnique, tuple_medium)

    sortByMedium(ArtistTecnique)
    #print(ArtistTecnique)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000

    return ArtistTecnique, elapsed_time_mseg
```

```
def getartistIDbyName(catalog, artistName): #Sacar ID de un artista
    for artist in lt.iterator(catalog["Artists"]):
        #print(artist)
        #print(artistName, artist['DisplayName'])
        if artistName.lower() == (artist["DisplayName"]).lower():
            return artist["ConstituentID"]
    return None

def getMediumOneArtist(catalog, artworks): #Creando el mapa de medios de un artista con sus obras
    for artwork in lt.iterator(artworks['Artworks']):
        AddArtworkMedium(catalog, artwork['Medium'], artwork)

def getArtworkOneMedium(catalog, medium): #Sacar obras del medio más utilizado
    artist_value = mp.get(catalog['ArtworkMedium'], medium)
    if artist_value:
        list_artworks = me.getValue(artist_value)
        return list_artworks['Artworks']
    return None
```

Complejidad temporal final = O(NLogN)

Requerimiento 4: [Individual] Clasificar las obras por la nacionalidad de sus creadores.

En este requerimiento se deben organizar todas las artworks por las respectivas nacionalidades de los artistas que trabajaron en ellas, entonces la mayor parte de este código se hará apenas se cargan los datos, y por cada artista cargado se investigarán las obras de dicho artista y finalmente se añadirán a la nacionalidad del mismo en un map cuyo “key” son las nacionalidades, este map implementa el separate chaining, el cual tiene una complejidad temporal de O(N), pues al no tener que iterar al haber una colisión, la operación se reduce a añadir los elementos en su casilla correspondiente. En las demás funciones simplemente se hará referencia a dicho maps, por lo que las complejidades de las demás funciones también estarán entre O(1) y O(N). En estas funciones basicamente se genera una pequeña lista con la cantidad de obras en cada uno de las nacionalidades y se organizan con mergesort(), lo cual tiene una complejidad linearítmica.

Función principal del requerimiento:

```
#Req 4:
#-------------------------------------------------------------------------------------------------
def getArtworkNationality(catalog):
    start_time = time.process_time()
    nationality_pop = lt.newList('ARRAY_LIST',comparenat)
    artwork_value = mp.keySet(catalog['ArtworkNationality'])
    for element in lt.iterator(artwork_value):
        artist_value = mp.get(catalog['ArtworkNationality'], element)
        number_artworks= me.getValue(artist_value)
        tuple_nat = {"Nationality":element,"NumbArtworks":lt.size(number_artworks["Artworks"])}
        lt.addLast(nationality_pop,tuple_nat)
    unknowncorrection(nationality_pop)
    sortByNationality(nationality_pop)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return nationality_pop,elapsed_time_mseg
def getArtworksOneNat(catalog, nationality):
    artist_value = mp.get(catalog['ArtworkNationality'], nationality)
    if artist_value:
        list_artworks= me.getValue(artist_value)
        return list_artworks['Artworks']
    return None
def unknowncorrection(nationality_pop):
    ombe1 = lt.isPresent(nationality_pop, "nationality unknown")
    ayuda1 = lt.getElement(nationality_pop, ombe1)
    lt.deleteElement(nationality_pop,ombe1)
    ombe2 = lt.isPresent(nationality_pop, "")
    ayuda2 = lt.getElement(nationality_pop, ombe2)
    final_number = int(ayuda1["NumbArtworks"])+ int(ayuda2["NumbArtworks"])
    tuple_corr = {"Nationality":"unknown", "NumbArtworks":str(final_number)}
    lt.deleteElement(nationality_pop,ombe2)
    lt.addLast(nationality_pop, tuple_corr)
    return None
```

Complejidad temporal final = O(NlogN).

Requerimiento 5: [En Grupo] Transportar obras de un departamento.

Hablando un poco de la función lo que se puede decir es que se encarga de, al brindarle un departamento del museo, extraer dichas obras y ordenarlas de diferentes maneras y sacarles el precio individualmente para luego hacer precios estimados, tanto de peso como de precio total por transportar todas esas obras. Esta función no cambió demasiado entre el requerimiento 1 y el requerimiento 2, pues si bien se cambia la manera de almacenar los datos, siempre se deberán hacer los mismos cálculos para el precio estimado y el peso estimado, además de la función en la que se calcula cada uno de los precios para cada obra. Siguiendo esta idea y recogiendo las complejidades del reto anterior la única distinta será al generar el maps, el cual por utilizar linear probing tendrá una complejidad de (N*M).

Función principal del requerimiento:

```python
#Req5
#-----------------------------------------------------------------------------------------------------------
def getArtworksByDepartment(catalog, department):
    start_time = time.process_time()
    artist_value = mp.get(catalog['ArtworkDepartment'], department)
    if artist_value:
        list_artworks= me.getValue(artist_value)
        listaconprecio = precioest(list_artworks)
        pesoestim = pesoest(list_artworks, "Weight (kg)")
        precioestim = pesoest(list_artworks, "Price")
        sorted_listbyprice = ms.sort(listaconprecio["Artworks"], compareprice)
        print(sorted_listbyprice.keys())
        artworkingsub = lt.subList(listaconprecio["Artworks"],1, lt.size(list_artworks["Artworks"]))
        sorted_listbyage = ms.sort(artworkingsub, compareage)
        stop_time = time.process_time()
        elapsed_time_mseg = (stop_time - start_time)*1000

        return sorted_listbyprice, sorted_listbyage, pesoestim, precioestim, elapsed_time_mseg

    return None
```

*Complejidad temporal final = O(N*M)*

Ambientes de prueba:

	Máquina 1
Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
Memoria RAM (GB)	8.00 GB (7.82 GB utilizable)
Sistema Operativo	Windows 10 64-bit x64-based processor

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

El tiempo de ejecución promedio para cada requerimiento Reto 1 vs Reto 2.

Resultados

Carga de Datos:

Máquina	***Tamaño de muestra***	***Tiempo promedio Reto 1***	***Tiempo promedio Reto 2***
1	2716	703.13	145.84
1	12658	12359.38	682.3
1	21664	34187.5	1026.05

1	38213	82953.13	2406.25

Requerimiento 1: [En Grupo] Listar cronológicamente los artistas.

Máquina	*Tamaño de muestra*	*Tiempo promedio Reto 1*	*Tiempo promedio Reto 2*
1	2716	15.625	0.0
1	12658	140.625	0.0
1	21664	156.25	10.42
1	38213	171.875	15.63

Requerimiento 2: [En Grupo] Listar cronológicamente las adquisiciones.

Máquina	*Tamaño de muestra*	*Tiempo promedio Reto 1*	*Tiempo promedio Reto 2*
1	2716	31.25	10.42
1	12658	156.25	260.42
1	21664	296.88	317.71
1	38213	515.63	610.3

Requerimiento 3: [Individual] Clasificar las obras de un artista por técnica.

Máquina	*Tamaño de muestra*	*Tiempo promedio Reto 1*	*Tiempo promedio Reto 2*
1	2716	0.0	0.0
1	12658	15.63	0.0
1	21664	15.63	0.0
1	38213	31.25	15.63

Requerimiento 4: [Individual] Clasificar las obras por la nacionalidad de sus creadores.

Máquina	*Tamaño de muestra*	*Tiempo promedio Reto 1*	*Tiempo promedio Reto 2*
1	2716	15.63	0.0
1	12658	78.13	0.0
1	21664	156.25	0.0
1	38213	312.25	0.0

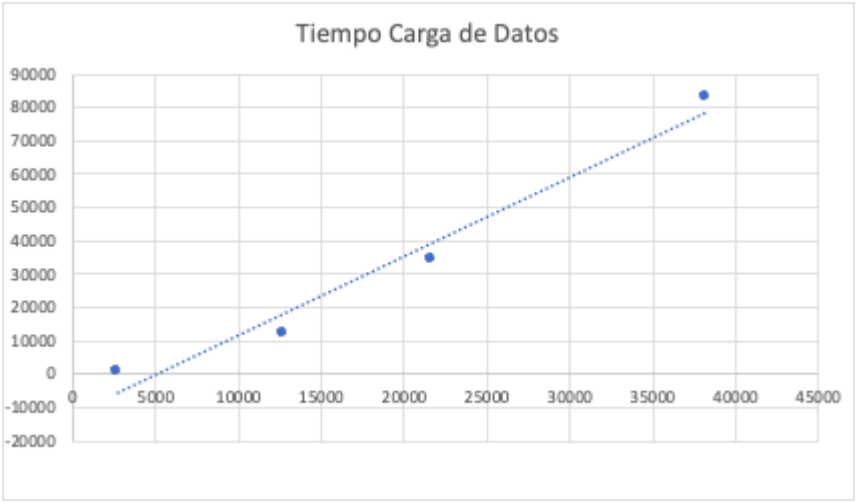
Requerimiento 5: [En Grupo] Transportar obras de un departamento.

Máquina	*Tamaño de muestra*	*Tiempo promedio Reto 1*	*Tiempo promedio Reto 2*
1	2716	15.63	36.46
1	12658	187.5	385.42
1	21664	421.88	807.3
1	38213	859.38	1296.85

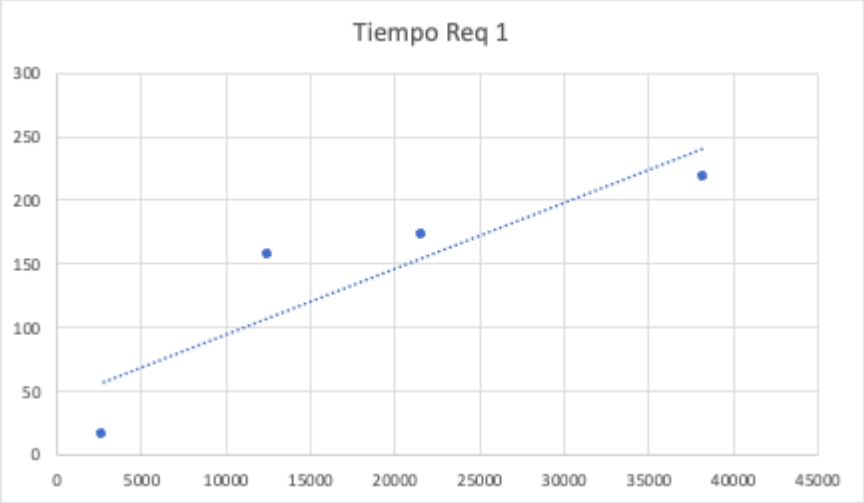
Gráficas de complejidad:

RETO 1:

Carga de datos:

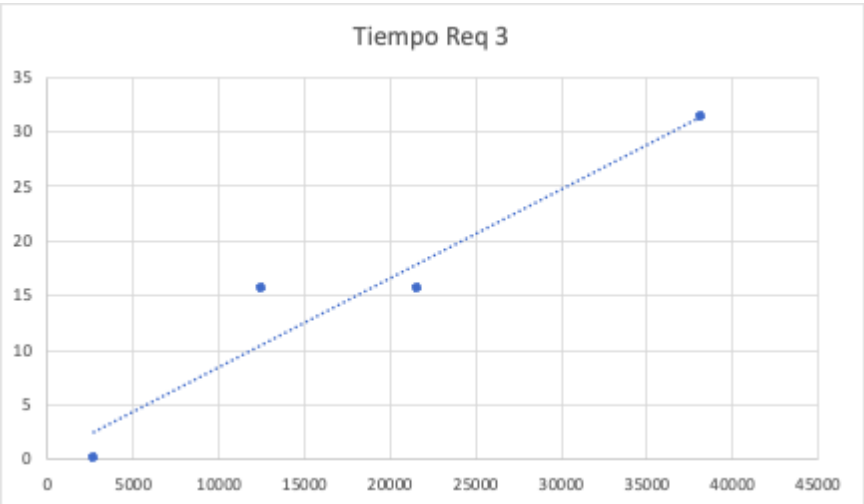


Requerimiento 1: [En Grupo] Listar cronológicamente los artistas.



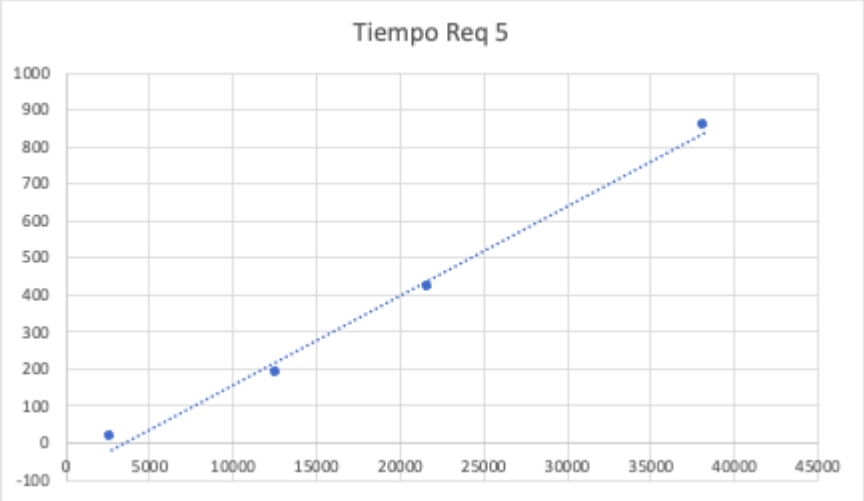
Requerimiento 2: [En Grupo] Listar cronológicamente las adquisiciones.

Requerimiento 3: [Individual] Clasificar las obras de un artista por técnica.



Requerimiento 4: [Individual] Clasificar las obras por la nacionalidad de sus creadores.

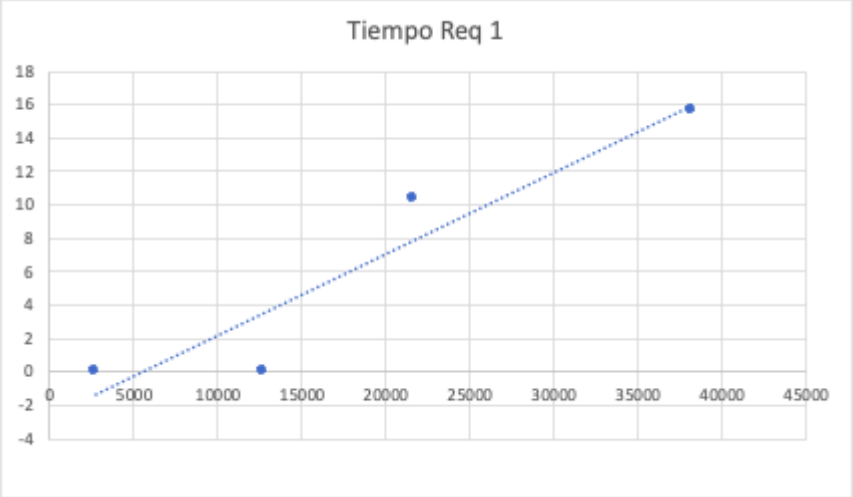
Requerimiento 5: [En Grupo] Transportar obras de un departamento.



RETO 2:

Carga de datos:

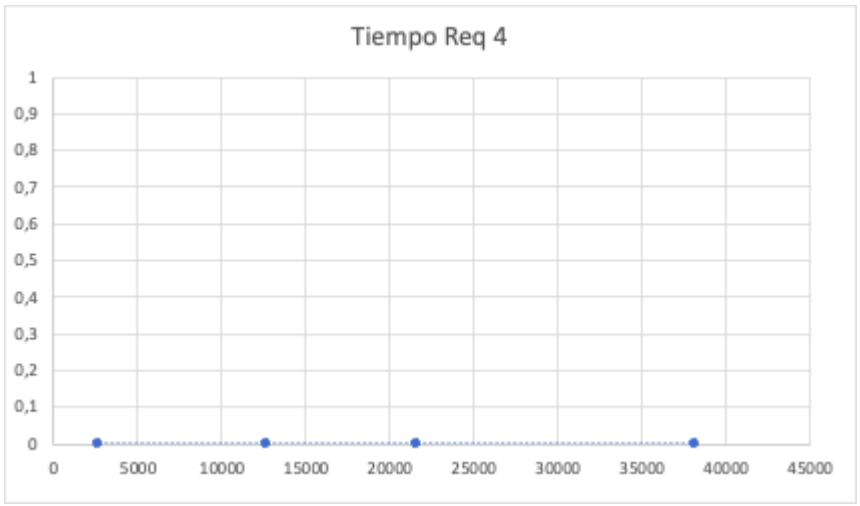
Requerimiento 1: [En Grupo] Listar cronológicamente los artistas.



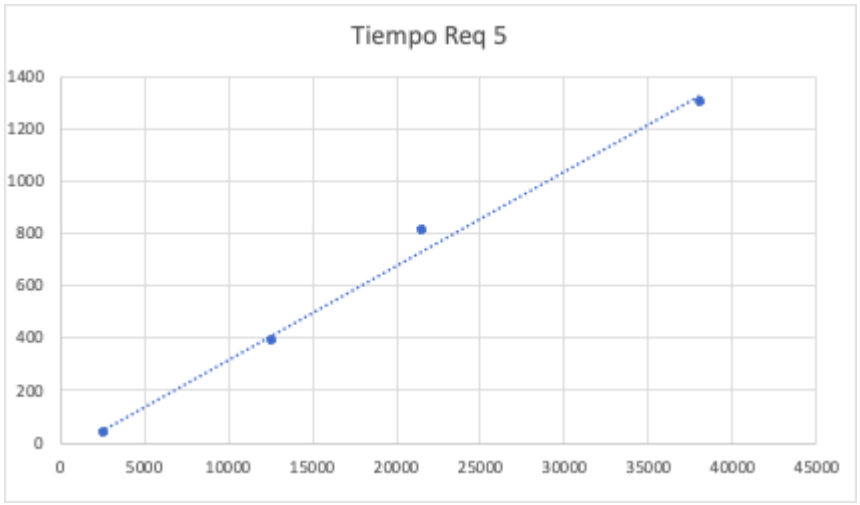
Requerimiento 2: [En Grupo] Listar cronológicamente las adquisiciones.

Requerimiento 3: [Individual] Clasificar las obras de un artista por técnica.

Requerimiento 4: [Individual] Clasificar las obras por la nacionalidad de sus creadores.



Requerimiento 5: [En Grupo] Transportar obras de un departamento.



Análisis:

A partir del análisis realizado datos, pudimos concluir el reto 2 tiene un tiempo y memoria mayor al reto 1, que a pesar de que al momento de realizar los demás requerimientos son bastante la carga de los eficientes en comparación con el reto 1. Por lo que se puede confirmar la premisa sobre la eficiencia de la estructura de mapas para la consulta de datos comprometiendo un poco de tiempo y memoria al momento de la carga.