Documento de análisis del reto 2

Estudiante 1- Maria Alejandra Moreno Bustillo Cod 202021603
Estudiante 2 - Juliana Delgadillo Cheyne Cod 202020986

I. Funciones principales
   a. *Funciones para la carga de datos y la creación de datos:*
      i. **newCatalog:** En esta función se crea la estructura del catálogo a utilizar en la carga de los datos de ambos archivos .csv y las listas correspondientes a cada sección del catálogo. El tipo de estructura de datos implementado en las dos TAD listas escogió fue 'ARRAY_LIST' debido a que en las pruebas realizadas en el laboratorio 4, fue posible evidenciar tiempos mucho mejores con esta que con single linked, además, es mucho más rápido trabajar con posiciones en array_ list (O(1)) que con single linked (O(n)). Para los TAD maps que se implementaron se utilizó la estructura de datos "linear probbing" con un factor de carga de 0.5 pues en el laboratorio 6 se pudo probar que esta tiene mejores tiempos y un muy buen manejo de colisiones.
      ii. **addArtist:** esta función crea un diccionario en el cual se específica cuales datos se desean extraer del archivo .csv donde se encuentra la información de los artistas (MoMA/Artists-utf8-large.csv). Para ello en el diccionario se establecen siete llaves y se les asigna como valor la información de la columna del archivo al que corresponden los datos solicitados. Cabe resaltar que una de estas llaves, la cual lleva el nombre de "Artworks" tiene como valor una lista de tipo "ARRAY_LIST" la cual guarda todas las obras de un autor y cada obra contiene a su vez información específica de dicha obra.
      iii. **addArtwork:** la función crea un diccionario en el cual se específica cuales datos se desean extraer del archivo .csv donde se encuentra la información de las obras de arte (MoMA/Artworks-utf8-large.csv). En el diccionario se establecen 18 llaves y se les asigna como valor la información de la columna del archivo al que corresponden los datos solicitados. Dentro de esta función también se crea un ciclo con un for, en el cual se obtiene los ID de cada artista con el fin de hacer la relación entre obra(s) y artista con la función addArtistTecnique y en addNationality, así mismo se especifica el formato en el que debe ingresar el dato de ID, para que este llegue sin espacios o caracteres adicionales que puedan afectar la carga de datos.
      iv. **addArtistTecnique:** Esta función es llamada en la función anterior de addArtwork por cada ConstituentID presente en una obra. En esta primero se hace la búsqueda en la lista de artistas del artista al que pertenece ese ConstituentID y una vez encontrado, se busca en el mapa de 'ArtistTecnique' el DisplayName del artista de esa obra de arte, en el caso de que se encuentre, se obtiene la pareja llave-valor, en la cual la llave es el nombre del artista y el valor es otro mapa que clasifica sus obras de arte según su técnica. Luego, se obtiene el valor asociado a ese artista y se llama la función AddMedium para agregar la obra a la lista que corresponde con la su técnica; adicionalmente, en el valor asociado al artista también hay una llave llamada 'TotalArtworks' a la cual se le suma 1 por cada obra de arte de ese artista

y esto se hace con el propósito de obtener con mayor facilidad el total de obras de ese artista. Cuando la llave no exista, se usa la función newArtist para crear el valor asociado al artista, se ejecute la función de AddMedium y luego se agrega al mapa de 'ArtistTecnique' la llave con el nombre del artista y el valor asociado que es el nuevo diccionario creado en newArtist.

v. **newArtist:** En esta función se crea un diccionario que consta de varias llaves: una de ellas es el artista en la cual se coloca el nombre del nuevo artista, la siguiente es TotalArtworks en la cual se lleva el conteo del total de obras encontradas de ese artista, la siguiente es MediumMayor para poder guardar en ella la información de la técnica más utilizada por ese artista, luego esta Artworks donde se crea el mapa de técnicas y por último esta TotalMedium que lleva un conteo del total de medios utilizados por el artista. Para el mapa de Artworks se utilizó un mapa de tipo probing con un factor de carga de 0.5 y una cantidad inicial de 200, pues estos tipos de mapa se pueden demorar un poco más durante la carga que los mapas de separate chaining, pero su manejo de colisión permite una consulta más rápida y sencilla de los elementos y el factor de carga elegido es el "default" ya que mantiene un buen equilibrio entre rapidez y uso de la memoria

vi. **newMedium:** Esta función crea una nueva estructura para modelar las obras de arte de un artista según su técnica. En esta hay dos llaves, una que corresponde al nombre de la técnica usada y la otra que es un TAD lista para ir almacenando las obras de arte de ese artista que se vayan encontrando y que utilicen dicha técnica.

vii. **addMedium:** En esta función se reciben de parámetros el diccionario correspondiente al artista dentro del mapa ArtistTecnique, el nombre de la técnica que utiliza la obra y la obra de arte. Primero se accede al mapa de Artworks en el que se clasifican las obras por técnica y se verifica si el nombre de la técnica se encuentra en el mapa, si este existe, entonces se obtiene la pareja llave-valor de esa técnica y se obtiene el valor de esta la cual es una lista de obras de arte. En el caso de no existir, se utiliza la función newMedium y se agrega al mapa la llave que es el nombre de la técnica y el valor asociado obtenido de newMedium, adicionalmente se aumenta en 1 el conteo de medios utilizados que se lleva en el diccionario correspondiente al artista dentro del mapa ArtistTecnique. Por último, se agrega a la lista de obras de arte la obra de arte que se recibió como parámetro.

viii. **addArtistDate:** En esta función primero se verifica que el begindate del artista que le entra por parámetro sea diferente de 0 y de espacio vacío. Como en las funciones anteriores se accede al mapa 'ArtistDates' y se verifica si dentro de este se encuentra el begindate del artista actual, en el caso de que si exista se obtiene la pareja llave-valor y se extrae el valor asociado. Si no existe, se utiliza la función newDate para crear un diccionario que será el valor asociado a la nueva fecha y esta pareja llave-valor se agrega al mapa, para finalmente agregar a la lista de artistas en el valor, el artista que entró como parámetro.

ix. **newDate:** Esta función es utilizada en AddArtisDate cuando se ingresa al condicional que especifica que la fecha aún no existe en el mapa. El propósito de newDate es crear un nuevo diccionario de dos llaves donde la primera es la fecha que ingresa y la segunda llave con el nombre "Artists" corresponde a una lista de tipo ARRAY_LIST la cual será posteriormente llenada con la información del artista que corresponda. Cabe aclarar que en esta función únicamente se lleva a cabo la creación de la lista. Esta función recibe como parámetro la fecha que se

desea añadir y retorna el diccionario asociado a dicho input, con las dos llaves mencionadas anteriormente. Cada vez que se añade una nueva fecha, el dato pasa por esta función, razón por la cual todas quedan cargadas en el mapa con el mismo formato descrito en la parte superior.

x. **addArtworkDate:** inicialmente, se comprueba que la fecha recibida tenga valor, luego se lleva a cabo un split para transformar el formato de ingreso a una lista de tres elementos y se extrae la primera posición que equivale al año. Posteriormente, se verifica si el año ya existe en el mapa, de ser así, simplemente se busca la pareja llave-valor y se obtiene el valor asociado, de lo contrario, se añade al mapa como una nueva fecha, utilizando la función newArtworkDate la cual crea un diccionario con la información que se necesita de esa obra en ese año. Finalmente, se agrega a la lista de obras de arte en el valor, la obra que entró como parámetro. Para esta función se decidido que el mapa utilizara el año mas no la fecha completa, pues esto nos ayuda a reducir el tamaño del mapa sustancialmente, sin disminuir su eficiencia al momento de buscar los datos solicitados.

xi. **newArtworkDate:** Esta función es usa en AddArtworkDate cuando se ingresa al condicional que especifica que la fecha no existe en el mapa. El objetivo de newArtworkDate es crear un nuevo diccionario de dos llaves, la primera es la fecha que ingresa y la segunda es a una lista de tipo ARRAY_LIST la cual será posteriormente llenada con la información de la obra que corresponda (en esta función únicamente se lleva a cabo la creación de la lista). La función recibe como parámetro la fecha que se desea añadir y retorna el diccionario asociado, con las dos llaves mencionadas anteriormente. Toda fecha que no haga parte del mapa pasa por esta función, en otras palabras, todas las fechas quedan cargadas en el mapa con el mismo formato descrito en la parte superior.

xii. **AddArtistNation:** la función recibe como parámetros el catálogo general, el id de un artista y la obra de arte. En primer lugar, se llama la lista de artistas del catálogo la cual con tiene toda la información de los artistas y se busca si el id ingresado corresponde a alguno de los artistas guardados en la lista, de ser así, se extrae de la lista de dicho artista y se encuentra su nacionalidad. Posteriormente, se comprueba si esta nación se encuentra en el mapa de nacionalidades. Si ya existe, se busca la pareja llave-valor y se obtiene el valor asociado, de lo contrario, se añade al mapa como una nueva nacionalidad, por medio de la función newNationality en la cual se crea en un diccionario donde el valor de la llave corresponde a una lista con la información de las obras de arte, y luego, se guarda en el mapa, como valor del nombre de la nacionalidad. Finalmente se agrega a la lista de obras de arte en el valor, la obra que entró como parámetro. Cabe mencionar que esta función cuenta con un filtro en el cual toda nacionalidad que ingrese con un valor vacío o con el nombre "nationality unknown" serán agrupadas dentro de una misma llave llamada "Unknown".

xiii. **newNationality**: crea un diccionario cuya llave "Artworks" corresponde a una lista de tipo ARRAY_LIST, en la cual se añadirá la información de la obra que se desea ingresar (en esta función únicamente se lleva a cabo la creación de la lista). La función, no necesita de parámetros para su funcionamiento y el retorno corresponde al diccionario con la llave de obras de arte descrita con anterioridad.

xiv. **addDpto:** en esta función se ingresa al mapa calificado por departamentos y se busca si el nombre del departamento que ingresa como parámetro hace parte del

mapa. Si ya existe, se busca la pareja llave-valor y se obtiene el valor asociado. Si no se encuentra presente, se añade como un nuevo departamento dentro del mapa utilizando la función newDpto en la cual se crea un diccionario cuya llave es "Artoworks" y tiene como valor una lista con la información de las obras de arte. A continuación, este diccionario se guarda dentro del mapa como valor del nuevo departamento y por último se agrega a la lista de obras de arte en el valor, la obra que entró como parámetro.

xv. **newDpto:** en esta función, se crea un diccionario con una llave que corresponde a una lista de tipo ARRAY_LIST en la que se añadirá la información de la obra que se desea ingresar, sin embargo, en esta función únicamente se lleva a cabo la creación de la lista. La función no recibe ningún parámetro y su retorno es el diccionario con la llave Artworks, descrita antenriormente.

b. *Funciones para comparar elementos dentro de una lista o mapa:*

i. **compareArtistID:** Esta función es utilizada para comparar los Constituent IDs de dos artistas y poder determinar si estos son iguales o diferentes. Por parámetro le entra el string del ID de un artista y un diccionario que contiene la información de otro artista.

ii. **compareObjectID:** Esta función es utilizada para comparar los Object IDs de dos obras de arte y poder determinar si estos son iguales o diferentes. Por parámetro le entra el string del ID de una obra de arte y un diccionario que contiene la información de otra obra de arte.

iii. **compareMapArtistDate:** Esta función es utilizada para realizar comparaciones dentro de un map, una de las entradas es un string de un año y la otra es una entrada del mapa (pareja llave-valor). El propósito de esta función es poder determinar si un año dado es igual a alguna de las llaves en el mapa.

iv. **compareArtistsByName:** Esta función es utilizada para comparar los Display Names de artistas dentro de un mapa que clasifica sus obras por artista. Una de sus entradas es un string del nombre del artista y el otro es una pareja llave-valor.

v. **compareMapMediums:** Esta función es utilizada para comparar dos técnicas y saber si estas son las mismas durante la construcción de un mapa. Una de las entradas es el nombre de una técnica y la otra es una pareja llave-valor, donde la llave era el nombre de una técnica.

vi. **compareNationality:** Esta función fue creada para ser utilizada en el mapa de nacionalidades. Una de las entradas es un string de una nacionalidad y la otra es una entrada llave-valor del mapa de nacionalidades, donde la llave es una nacionalidad. El propósito de esta función es poder saber si la nacionalidad que entra como parámetro se encuentra dentro del mapa de nacionalidades.

vii. **cmpartworkyear:** Esta función de comparación se encarga de comparar las fechas de adquisición de dos obras de arte, de las cuales primero se verifica que tenga un valor diferente de None y, en segundo lugar, se ejecuta el comando d.date dado el formato en el que ingresa esta fecha. La función tiene como propósito evaluar si una es menor que la otra. Es utilizada por la función de ordenamiento del requerimiento 2.

viii. **compareMapDptos:** se utilizada para el mapa de Departamentos. La función recibe como parámetros el nombre de un departamento (str) y una entrada llave-

valor del mapa de departamentos, del cual se extraen las llaves que corresponden a los departamentos cargados. Esta función se encarga de determinar si el departamento que ingresa hace parte del mapa de departamentos, de tal manera que si no pertenece se retornará un 0.

ix. **cmpTranspOld:** Esta función recibe como parámetro dos obas de arte y se encarga de comparar las fechas de dichas obras. En primer lugar, se verifica que la fechas tengan un valor diferente de None y a continuación se lleva a cabo la comparación "menor que" con el propósito de evaluar si una es menor que la otra. Es utilizada por una de las funciones de ordenamiento del requerimiento 5.

x. **cmpTranspCost:** Esta función se encarga de comparar el costro que tiene dos obras de arte, para ello se extrae de la información de la obra su precio que se encuentra guardado en la llave "Cost" y se determina que dato es mayor que el otro. Esta comparación, se utiliza en una de las funciones de ordenamiento necesarias para el requerimiento 5.

c. *Funciones de ordenamiento:*

```
857  # Funciones de ordenamiento
858  def sortTecSize(tecnique_map):
859
860      ms.sort(tecnique_map, cmpTecSize)
861
862  def sortYear(lista_obras):
863      "lab 5"
864      ms.sort(lista_obras, cmpArtworkDate)
865
866  def sortTranspOld(list_old):
867
868      ms.sort(list_old, cmpTranspOld)
869
870  def sortTransportation(transp_cost):
871
872      ms.sort(transp_cost, cmpTranspCost)
873
874  def sortNationalitysize(nationalities):
875
876      ms.sort(nationalities, cmpNationalitysize)
877
878  def sortArtwork(artwork_inrange):
879
880      ms.sort(artwork_inrange, cmpartworkyear)
881
```

Como es posible evidenciar en la imagen anterior, las funciones de ordenamiento utilizadas para cumplir los requerimientos del Reto 2, tienen la misma estructura y usan el algoritmo de ordenamiento "merge sort" (ms). Este fue el algoritmo que se eligió para todos los ordenamientos, porque su complejidad de O(nlog(n)) es constante sin importar la cantidad de datos suministrados. Al igual, que en el desarrollo del reto 1, encontramos que Merge genera mejores tiempos de ejecución en el código en comparación con otros algoritmos y que además de esto es estable. Cabe mencionar que este algoritmo recursivo no es in-place

lo que implica mayor necesidad de espacio en la maquina en la que se ejecuta el código, sin embargo, dadas las condiciones de la computadora que utilizamos, consideramos que vale la pena usar más espacio a cambio de reducir tiempos en la ejecución.

II. Requerimiento 1 (Grupal) – Listar cronológicamente los artistas

En el código se puede evidenciar el código implementado para resolver el requerimiento 1 en el cual era necesario listar cronológicamente a los artistas que habían nacido en un rango de años dado por el usuario. En un primer momento se realiza un while con el cual se van sacando con la función get y getValue la lista de los artistas nacidos en el rango de años dado y con ayuda de un for loop se va agregando cada artista a la lista auxiliar "artist_inrange", la cual será la respuesta final y dará la lista de todos los artistas nacidos en ese rango de años. Debido a que se van sacando los artistas en orden, no es necesario realizar ningún tipo de sort, ya todo viene organizado. El primer while otorga una complejidad de O(m) donde m es la cantidad de años dentro del rango solicitado, por otro lado, el for loop otorga una complejidad de O(n) donde n es la cantidad de artistas que se encuentran dentro de cada año que se recorrió. La complejidad final del requerimiento sería de O (n*m), una complejidad menor a la del requerimiento en el reto pasado, pues no tuvimos que realizar ningún ordenamiento, ya que los años se fueron obteniendo de manera ascendente y se pudo obtener de una vez la respuesta ordenada. Adicionalmente, no tocó recorrer todos los artistas para poder obtener a todos aquellos que pertenecían al rango, por lo que, en su caso promedio, rendirá mejor que en el reto 1.

```
def getArtistYear(catalog, year_i, year_f):
    artist_inrange = lt.newList('ARRAY_LIST')
    i = year_i

    while i >= year_i and i <= year_f:
        pareja_year = mp.get(catalog['ArtistDates'], i)

        if pareja_year:
            year_value = me.getValue(pareja_year)

            for artist in lt.iterator(year_value['Artists']):
                lt.addLast(artist_inrange,artist)

        i += 1

    return artist_inrange
```

III. Requerimiento 2 (Grupal) – Listar cronológicamente las adquisiciones

Como se evidencia en el código, lo primero que se realiza es un split, con el fin de representar la fecha que ingresa como una lista de tres elementos. A partir de lo anterior se extrae el año de la fecha inicial, es decir la primera posición de la lista y se crea un ciclo while (O(k) siendo k el número de años en el rango) el cual recorre únicamente los años que se encuentran dentro del rango de la fecha inicial y la fecha final. Dentro del ciclo se utiliza la función d.datetime para considerar la fecha completa que se ingresó tanto para fecha inicial como fecha final, de tal forma que también se tenga en cuenta el mes y el día y se obtiene del mapa ArtworkDates la pareja llave valor asociada al año que se está utilizando, de la cual se extrae su valor. El valor obtenido corresponde a todas las obras de arte asociadas a la llave. A continuación, dentro de un bucle for (O(m) siendo m las obras que fueron adquiridas en ese año) se iteran todas las obras de arte que se encontraron en ese año y a cada una de ellas se les extrae su fecha de adquisición, se formatea con el comando d.datetime y se realiza una comparación entre esta fecha y aquellas ingresadas por paramento (inicial y final). Si la fecha extraída en el bucle pertenece al rango delimitado por los parámetros

se añade la obra de arte a una lista de tipo ARRAY_LIST y se verifica si dicha obra fue comprada o no (purchased). Finalmente, con la lista de obras completa para el rango de tiempo solicitado, se obtiene su tamaño (número de elementos), y se ejecuta una función de ordenamiento (O(nlog(n))) para ordenarlas por fecha de adquisición de menor a mayor. La complejidad total para este requerimiento fue de (O(k*m) + O(nlog(n))) mientras que para el reto 1, la complejidad final fue de O(n) + O(nlog(n)). Luego de analizar cada una de estas complejidades, encontramos que es más eficiente la implementación realizada en el reto 2, debido a que, a pesar de tener dos ciclos, estos solo son ejecutados dentro de un rango de datos especifico el cual se encuentra delimitado por las fechas inicial y final que ingresa el usuario, a diferencia del código del reto 1 en el que se recorría en un solo ciclo, todas las obras del archivo pertenecieran o no al rango solicitado. Lo anterior nos permite afirmar que la complejidad (O(k*m)) es menor a O(n) en todos los casos posibles casos a excepción del peor caso para ambos códigos, en el cual el usuario ingrese un rango de fechas que abarque todos los datos del archivo, para esta situación, en ambos casos la complejidad seria de O(n), pues se deberán recorrer todos los datos. Es decir, en el peor de los casos, tendrá que recorrer todos los años junto con las obras asociadas a esos años que termina siendo O(n) siendo n el total de obras de arte, pero en el caso promedio, este debería comportarse mejor que el algoritmo implementado en el reto 1.

```
def getArtworkYear(catalog, fecha_i, fecha_f):
    start_time = time.process_time()
    artwork_inrange = lt.newList('ARRAY_LIST')

    fecha_i_sep = (fecha_i).split('-')
    fecha_f_sep = (fecha_f).split('-')
    purchased = 0

    i = int(fecha_i_sep[0])

    while i >= int(fecha_i_sep[0]) and i <= int(fecha_f_sep[0]):
        di = d.datetime(int(fecha_i_sep[0]),int(fecha_i_sep[1]), int(fecha_i_sep[2]))
        df = d.datetime(int(fecha_f_sep[0]),int(fecha_f_sep[1]), int(fecha_f_sep[2]))
        pareja_year = mp.get(catalog['ArtworkDates'], i)
        i+=1

        if pareja_year:
            year_value = me.getValue(pareja_year)

            for artwork in lt.iterator(year_value['Artworks']):
                date_artwork = artwork['DateAcquired'].split("-")
                d1 = d.datetime(int(date_artwork[0]),int(date_artwork[1]), int(date_artwork[2]))

                if d1 >= di and d1 <= df:
                    lt.addLast(artwork_inrange, artwork)
                    if 'purchase' in artwork['CreditLine'].lower():
                        purchased += 1

    size = lt.size(artwork_inrange)
    sortArtwork(artwork_inrange)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return artwork_inrange, size, elapsed_time_mseg, purchased
```

IV. Requerimiento 3 (María Alejandra Moreno Bustillo) – Clasificar las obras de un artista por técnica

En este código se puede evidenciar que mediante un mp.get se obtiene la pareja llave-valor en el caso de que exista ese artista dentro del mapa ArtistTecnique. Si este existe, entonces se obtiene el valor asociado a ese artista el cual es un mapa que clasifica las obras del artista por técnica utilizada y con esto ya se obtiene la información solicitada por el requerimiento. Por otro lado, se obtuvo una lista de los valores del mapa de técnicas y se utiliza un for para recorrer la lista y de esta manera encontrar la técnica más utilizada por el artista. Para el análisis de complejidad, la única operación que aporta una complejidad no constante es el for que es utilizado para recorrer las técnicas de ese artista por lo que la complejidad temporal sería de O(n) n siendo el número de técnicas de ese artista, por lo general ese número es pequeño, entonces de igual manera el programa responderá de manera óptima y con buenos tiempos. En el reto anterior donde se hizo uso de TAD listas, la complejidad temporal de este requerimiento fue de O(n*k) + O(nlog(n)), por lo que se puede evidenciar que en este caso se redujo significativamente la complejidad del programa al ya no tener

que hacer recorridos para encontrar al artista, ni tener que ordenar las listas según la cantidad de obras en cada técnica.

```
def getArtistTecnique(catalog, artist_name):
    """
    Retorna las obras de arte de un artista clasificadas por medio/técnica
    """
    start_time = time.process_time()
    artist_map = mp.get(catalog['ArtistTecnique'], artist_name)
    mayor_num = 0
    mayor_elem = None

    if artist_map:
        tecnique_map = me.getValue(artist_map)
        tecnique_values = mp.valueSet(tecnique_map['Artworks'])
        tamano_tecs = mp.size(tecnique_map['Artworks'])
        total_obras = tecnique_map['TotalArtworks']


        for tecnique in lt.iterator(tecnique_values):

            if lt.size(tecnique['Artworks']) > mayor_num:
                mayor_num = lt.size(tecnique['Artworks'])
                mayor_elem = tecnique

    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return mayor_elem, tamano_tecs, total_obras, elapsed_time_mseg
```

V. Requerimiento 4 (Juliana Delgadillo Cheyne) – Clasificar las obras por las nacionalidades de sus creadores

Como se logra observar en la imagen a continuación, el código inicia llamando el mapa de nacionalidades y extrayendo de él todas sus llaves las cuales corresponden a los nombres de las nacionalidades que se cargaron del archivo .csv. Posteriormente, se realiza un bucle for (O(k)) en el cual se itera cada nacionalidad. Dentro de ciclo se encuentra la pareja llave_valor y se obtiene su valor asociado. Lo anterior se lleva a cabo porque dentro del valor de la pareja se encuentran las obras de arte (con su respectiva información) de todos los artistas que pertenezcan a dicha nacionalidad. Con esta información, se utiliza el comando lt.size sobre la lista de obras para determinar su tamaño y así conseguir el número de obras que contiene la nacionalidad con la que se está iterando. A continuación, se crea un diccionario de tres llaves, que contiene la nacionalidad, el número de obras asociada, y las obras de arte con su respectiva información; este diccionario es añadido una y otra vez a una lista de tipo ARRAY_LIST hasta que todas las nacionalidades cargadas hayan sido incluidas. Finalmente se ejecuta una función de ordenamiento a esta lista de diccionarios O(nlog(n)) de tal forma que queden posicionados de mayor a menor en cuanto a número de obras por nacionalidad. La complejidad total para este requerimiento fue de O(k) + O(nlog(n)) mientras que para el reto 1, la complejidad final fue de O(n*m) + O(nlog(n)). Luego de analizar cada una de estas complejidades, encontramos que es más eficiente el código de la función para el reto 2 ya que este solo necesita de un ciclo para recorrer las nacionalidades existentes en el archivo las cuales ya se encuentran agrupadas dentro de un mapa, reduciendo significativamente la complejidad para cumplir con el requerimiento.

```python
def getNationality(catalog):

    start_time = time.process_time()
    answer = lt.newList("ARRAY_LIST",cmpfunction=compareNationality)

    nation_map = catalog['Nationality']
    nationality =  mp.keySet(nation_map)

    for name in lt.iterator(nationality):
        nationality_entry = mp.get(nation_map, name)
        nationality_value = me.getValue(nationality_entry)

        artworks = nationality_value['Artworks']

        total_obras = lt.size(nationality_value['Artworks'])
        diccionario = {"Nationality": name,
                       "Total works": total_obras,
                       "Artwork": artworks
                       }
        lt.addLast(answer, diccionario)

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

VI. Requerimiento 5 (Grupal) – Transportar obras de un departamento

En el quinto requerimiento era necesario calcular el costo de transportar las obras de un departamento seleccionado por el usuario. Para este requerimiento se hizo uso de una lista auxiliar llamada transp_cost, la cual será la lista final que contiene a las obras de arte y el costo de transporte de cada una. Como se puede ver en la imagen, primero se hace uso de la función get y getValue para poder obtener del mapa de departamentos, las obras del departamento solicitado en O (1), luego esa lista de obras pertenecientes a ese departamento se recorre con un for loop que le agrega una complejidad de O(n) siendo n el número de obras de ese departamento. Luego se hacen varios condicionales para lograr tratar con excepciones en los datos del archivo.csv los cuales otorgan complejidades mínimas de O (1), así como las funciones utilizadas para calcular el costo del área y del volumen, donde únicamente se usan comparaciones y operaciones matemáticas. Al final se crea una copia de la lista de transp_cost para organizar esta según la antigüedad de las obras y la lista original se organiza según el costo de sus obras de mayor a menor. Ambos ordenamientos usan merge sort y la complejidad que otorgan es de O(nlog(n)). La complejidad final de este requerimiento es de O(n) + O(nlog(n)). Comparando con el reto pasado, la complejidad temporal promedio es menor, puesto que nos ahorramos un for loop para buscar las obras de arte del departamento solicitado, ya que lo tenemos todo previamente clasificado en un mapa.

```python
def getTranspCost(catalog, dpto):
    start_time = time.process_time()
    costo_total = 0
    peso_total = 0
    transp_cost = lt.newList('ARRAY_LIST')
    artwork_dpto_entry = mp.get(catalog['ArtworkDpto'], dpto)

    if artwork_dpto_entry:

        artworksBydpto = me.getValue(artwork_dpto_entry)

        for artwork in lt.iterator(artworksBydpto['Artworks']):

            artwork_filtrada = {'Title': artwork['Title'],
                                'Artist/s':artwork['Artists'],
                                'Classification': artwork['Classification'],
                                'Date':artwork['Date'],
                                'Medium':artwork['Medium'],
                                'Dimensions':artwork['Dimensions']}
            weight = artwork['Weight']

            if artwork['Weight'] == '':
                weight = 0
            else:
                weight = float(artwork['Weight'])

            cost_weight=round(((weight)*72),2)
            cost_a = round(((cost_Area(artwork))/10000),2)
            cost_vol = round(((cost_volume(artwork))/1000000),2)

            if cost_weight == 0 and cost_a == 0 and cost_vol == 0:
                costo_total  += 48.00
                cost = {'Artwork':artwork_filtrada,
                        'Cost':48.00}

                lt.addLast(transp_cost,cost)

            elif cost_weight > cost_vol and cost_weight > cost_a:

                costo_total  += cost_weight
                peso_total += weight
                cost = {'Artwork':artwork_filtrada,
                        'Cost':cost_weight}

                lt.addLast(transp_cost,cost)

            elif cost_a > cost_weight and cost_a > cost_vol:

                costo_total += cost_a
                peso_total += weight
                cost = {'Artwork':artwork_filtrada,
                        'Cost':cost_a}

                lt.addLast(transp_cost,cost)

            elif cost_vol > cost_a and cost_vol > cost_weight:

                costo_total  += cost_vol
                peso_total += weight
                cost = {'Artwork':artwork_filtrada,
                        'Cost':cost_vol}

                lt.addLast(transp_cost,cost)

        copy= lt.subList(transp_cost,1,lt.size(transp_cost))
        sortTransportation(transp_cost)
        sortTranspOld(copy)
        stop_time = time.process_time()
        elapsed_time_mseg = (stop_time - start_time)*1000

        return transp_cost,round(costo_total,2), copy, peso_total, elapsed_time_mseg
```

```python
def cost_Area(artwork):
    pi = math.pi
    length = artwork['Length']
    height = artwork['Height']
    width = artwork['Width']
    diameter =artwork['Diameter']

    #area de la forma largo por altura
    if artwork['Length'] == '':
        length = 0
    else:
        length = float(length)
    if artwork['Height'] == '':
        height = 0
    else:
        height = float(height)

    if artwork['Diameter'] == '':
        diameter = 0
    else:
        diameter = float(diameter)

    if artwork['Width'] == '':
        width = 0
    else:
        width = float(width)


    cost_area1 = (length*height)*72
    cost_area5 = (width*height)*72
    #area de circulo
    cost_area2 = (pi*((diameter)/2)**2)*72
    #area cilindro
    cost_area3 = (2*(pi*(diameter)/2)*height) + 2*((math.pi*((diameter)/2)**2))*72
    #area esfera
    cost_area4 = (4*(pi*(diameter)/2)**2)*72

    if cost_area1 > cost_area2 and cost_area1 >cost_area3 and cost_area1 > cost_area4 and cost_area1 > cost_area5:
        return cost_area1
    elif cost_area2 > cost_area1 and cost_area2 > cost_area3 and cost_area2 > cost_area4 and cost_area2 > cost_area5:
        return cost_area2
    elif cost_area3 > cost_area1 and cost_area3 > cost_area2 and cost_area3 > cost_area4 and cost_area3 > cost_area5 :
        return cost_area3
    elif cost_area4 > cost_area1 and cost_area4 > cost_area2 and cost_area4 > cost_area3 and cost_area4 > cost_area5 :
        return cost_area4
    else:
        return cost_area5
```

```python
def cost_Area(artwork):
    pi = math.pi
    length = artwork['Length']
    height = artwork['Height']
    width = artwork['Width']
    diameter =artwork['Diameter']

    #area de la forma largo por altura
    if artwork['Length'] == '':
        length = 0
    else:
        length = float(length)
    if artwork['Height'] == '':
        height = 0
    else:
        height = float(height)

    if artwork['Diameter'] == '':
        diameter = 0
    else:
        diameter = float(diameter)

    if artwork['Width'] == '':
        width = 0
    else:
        width = float(width)


    cost_area1 = (length*height)*72
    cost_area5 = (width*height)*72
    #area de circulo
    cost_area2 = (pi*((diameter)/2)**2)*72
    #area cilindro
    cost_area3 = (2*(pi*(diameter)/2)*height) + 2*((math.pi*((diameter)/2)**2))*72
    #area esfera
    cost_area4 = (4*(pi*(diameter)/2)**2)*72

    if cost_area1 > cost_area2 and cost_area1 >cost_area3 and cost_area1 > cost_area4 and cost_area1 > cost_area5:
        return cost_area1
    elif cost_area2 > cost_area1 and cost_area2 > cost_area3 and cost_area2 > cost_area4 and cost_area2 > cost_area5:
        return cost_area2
    elif cost_area3 > cost_area1 and cost_area3 > cost_area2 and cost_area3 > cost_area4 and cost_area3 > cost_area5 :
        return cost_area3
    elif cost_area4 > cost_area1 and cost_area4 > cost_area2 and cost_area4 > cost_area3 and cost_area4 > cost_area5 :
        return cost_area4
    else:
        return cost_area5
```

```python
def cost_volume(artwork):

    pi = math.pi

    length = artwork['Length']
    height = artwork['Height']
    diameter = artwork['Diameter']
    width = artwork['Width']
    depth = artwork['Depth']

    if artwork['Width'] == '':
        width = 0
    else:
        width = float(width)
    if artwork['Length'] == '':
        length = 0
    else:
        length = float(length)
    if artwork['Height'] == '':
        height = 0
    else:
        height = float(height)
    if artwork['Diameter'] == '':
        diameter = 0
    else:
        diameter = float(diameter)
    if artwork['Depth'] == '':
        depth = 0
    else:
        depth = float(depth)

    #volumen de la forma longitud por altura por ancho
    cost_vol1 = (length*height*width)*72
    cost_vol2 = (length*height*depth)*72
    #volumen esfera
    cost_vol3 = ((4/3)*((pi*(diameter)/2)**3))*72
    #volumen cilindro
    cost_vol4 = ((pi*((diameter)/2)**2)*height)*72

    if cost_vol1 > cost_vol2 and cost_vol1 > cost_vol3 and cost_vol1 > cost_vol4:
        return cost_vol1
    elif cost_vol2 > cost_vol1 and cost_vol2 > cost_vol3 and cost_vol2 > cost_vol4:
        return cost_vol2
    elif cost_vol3 > cost_vol1 and cost_vol3 > cost_vol2 and cost_vol3 > cost_vol4:
        return cost_vol3
    else:
        return cost_vol4
```

VII. Pruebas temporales y de memoria para cada requerimiento

Ambientes de pruebas

	Máquina
Procesadores	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz
Memoria RAM (GB)	12,0 GB (9,95 GB usable)
Sistema Operativo	Windows 10 64-bit operating system

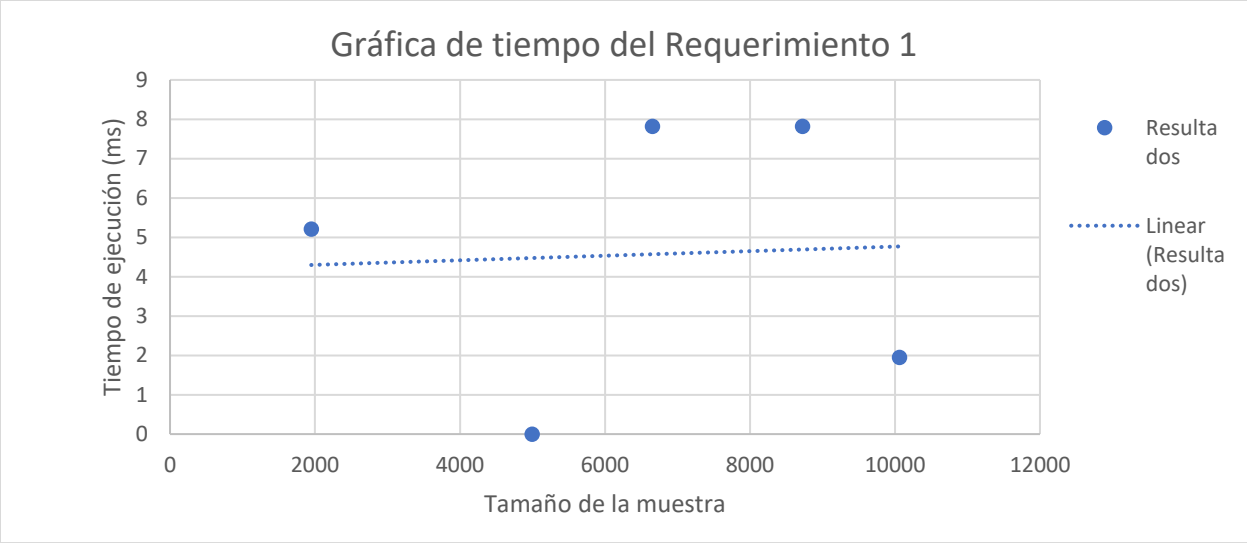
Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

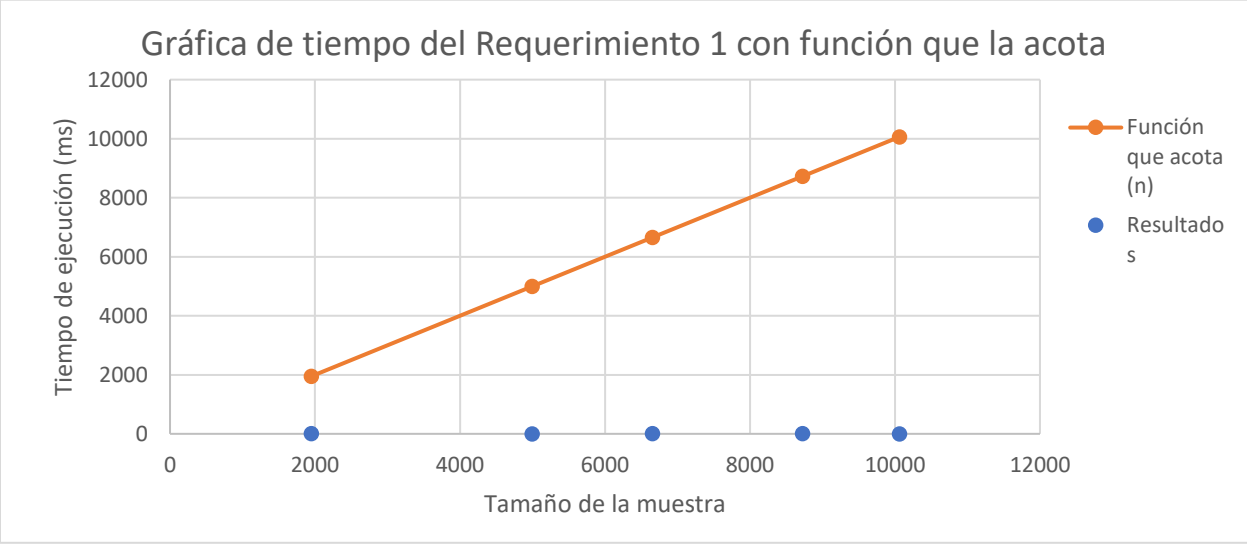
Resultados

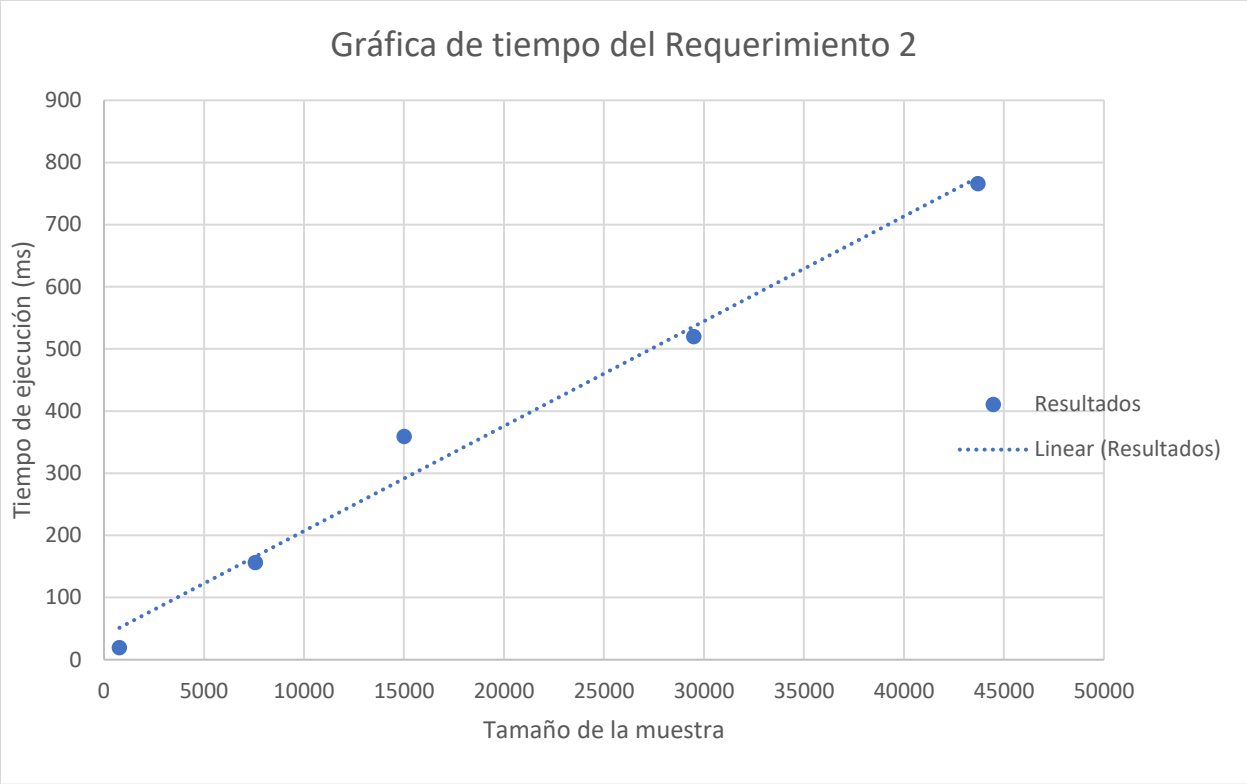
Req 1 – 1920-1985
Req 2 – 1944-06-06 hasta 1989-11-09
Req 3 – Louise Bourgeois
Req 5 – drawings & prints

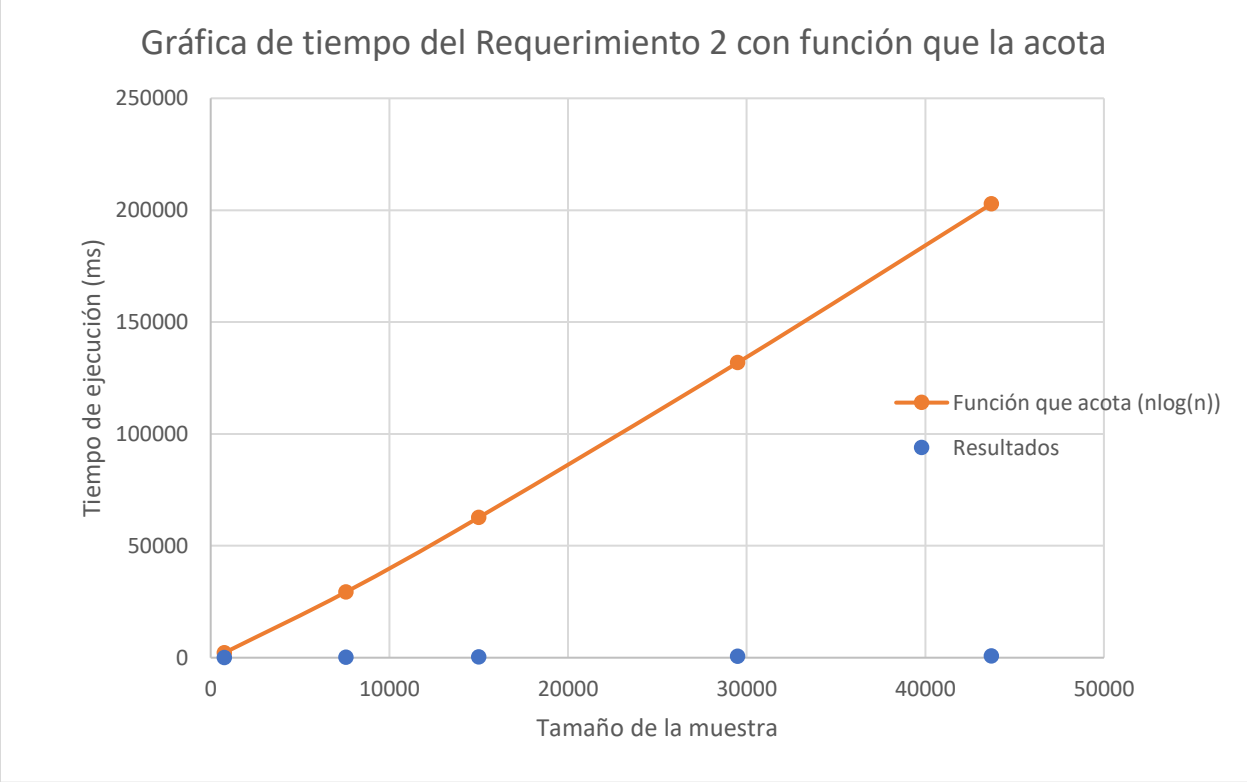
Porcentaje de la muestra [pct]	Tamaño de la muestra	Req-1 [ms]	Req-2 [ms]	Req-3 [ms]	Req-4 [ms]	Req-5 [ms]
0.50%	Artistas = 1948 obras = 768	5.21	19.53	0.0	0.0	31.25
5.00%	Artistas = 4996 obras = 7572	0.0	156.25	0.0	0.0	437.5
10.00%	Artistas = 6656 obras = 15008	7.82	359.37	0.0	3.91	984.37
20.00%	Artistas = 8724 obras = 29489	7.82	520.04	0.0	0.0	1359.375
30.00%	Artistas = 10063 obras =43704	1.95	765.625	3.91	3.91	2109.375

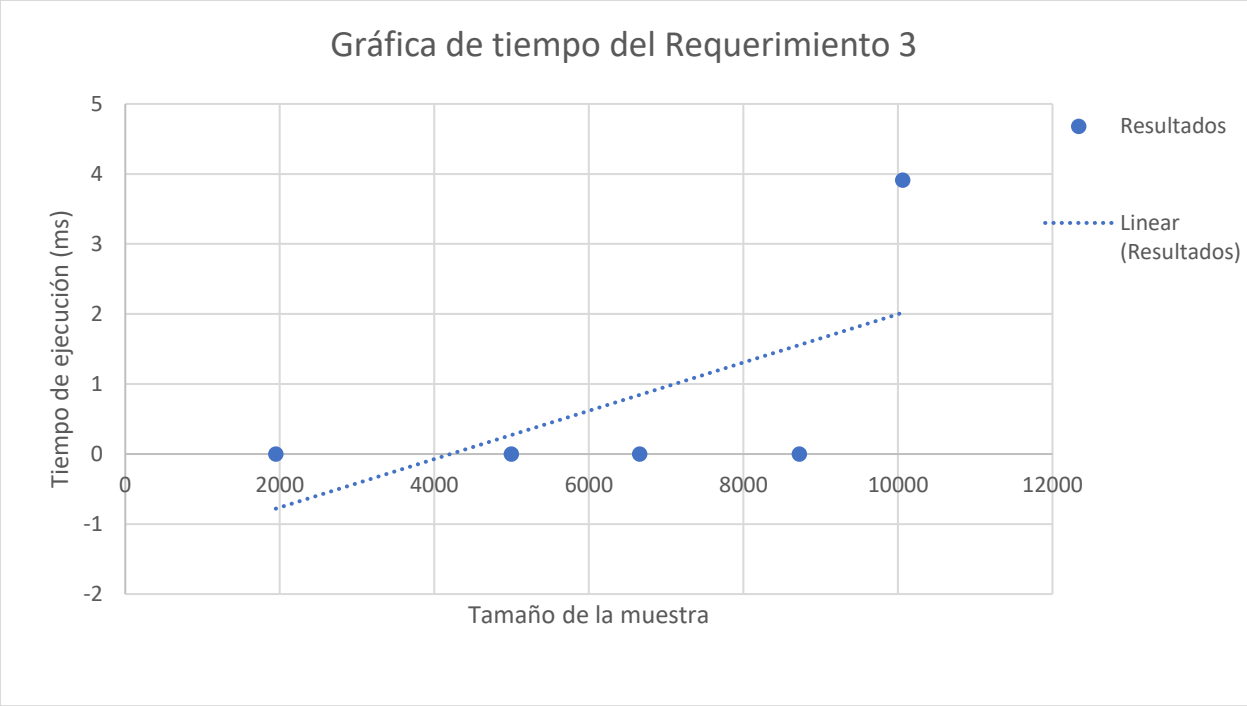
A continuación, se presentarán las gráficas con los resultados temporales obtenidos. Hay dos gráficas por cada requerimiento, una en la que se pueda apreciar los resultados obtenidos únicamente con su correspondiente línea de tendencia y otro con esos resultados y con la función que acota los resultados obtenidos según el análisis de complejidades.

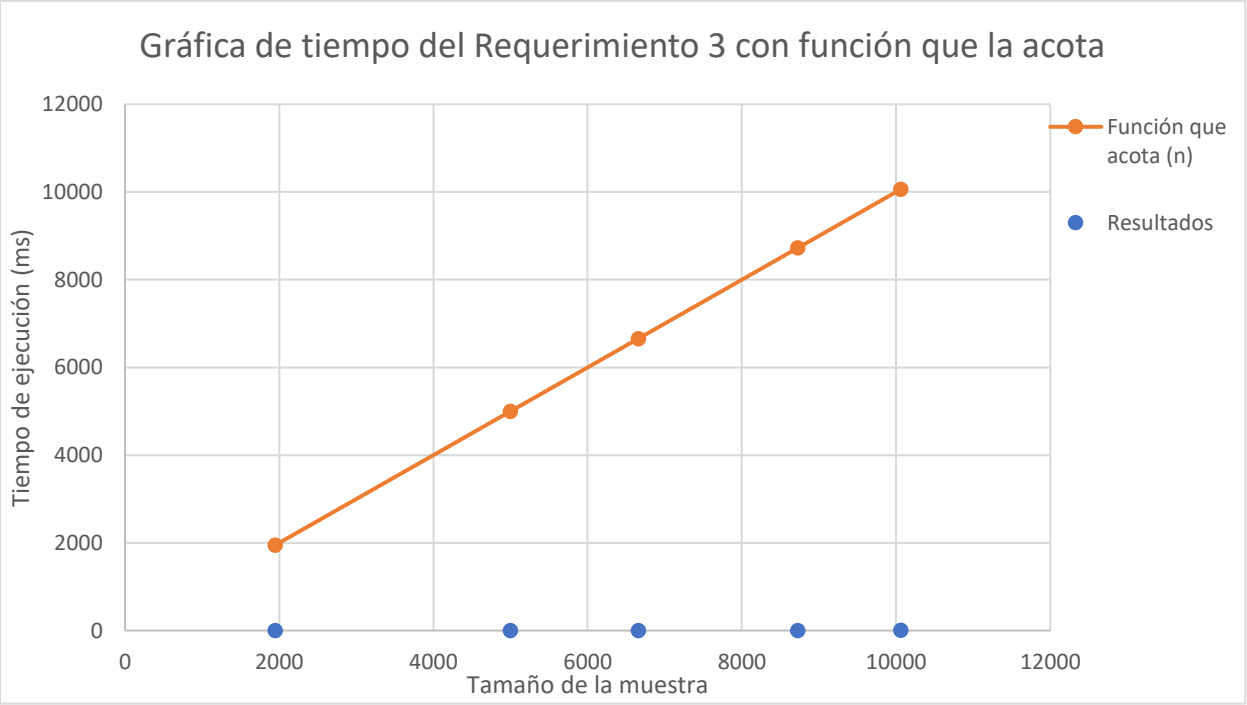

Gráfica de tiempo del Requerimiento 1
Tiempo de ejecución (ms)
Tamaño de la muestra
0
1
2
3
4
5
6
7
8
9
0
2000
4000
6000
8000
10000
12000
Resultados
Linear (Resultados)

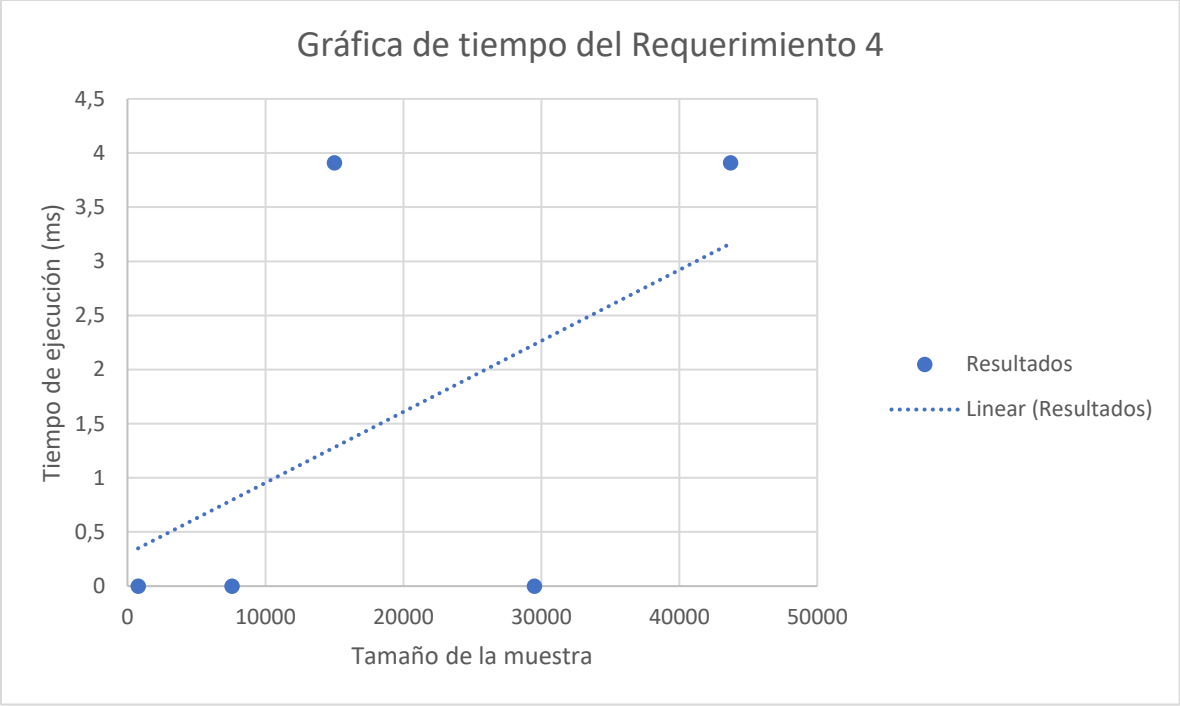

Gráfica de tiempo del Requerimiento 1 con función que la acota
Tiempo de ejecución (ms)
Tamaño de la muestra
0
2000
4000
6000
8000
10000
12000
0
2000
4000
6000
8000
10000
12000
Función que acota (n)
Resultados

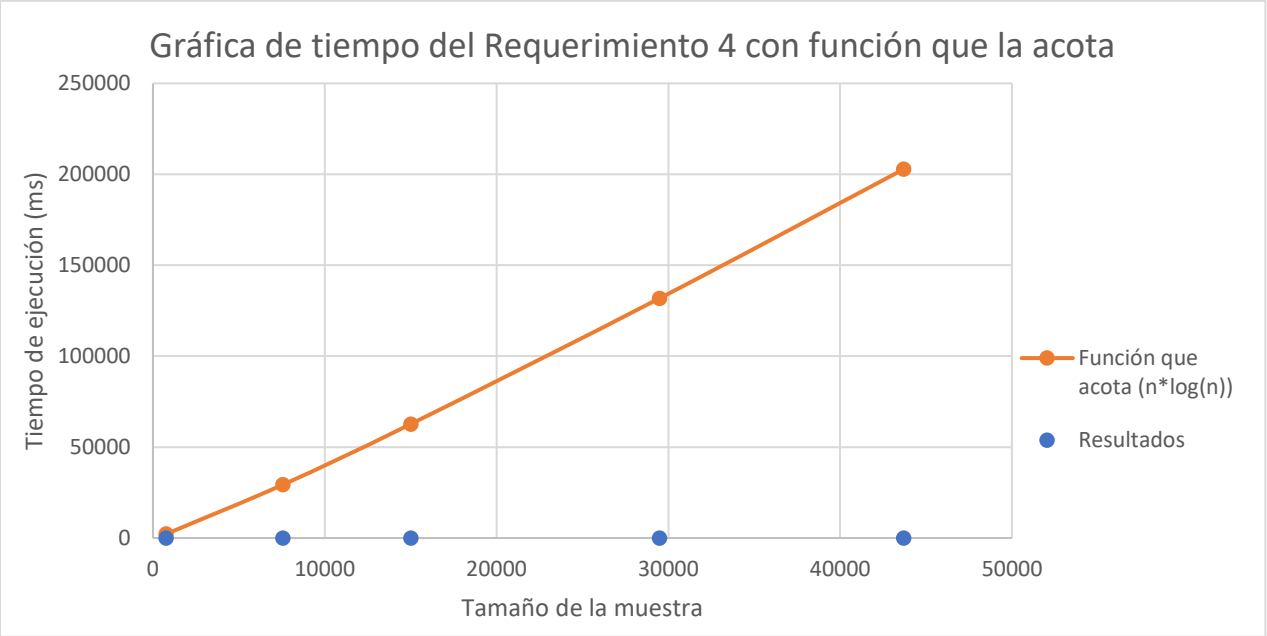

Gráfica de tiempo del Requerimiento 2
Tiempo de ejecución (ms)
900
800
700
600
500
400
300
200
100
0
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
Tamaño de la muestra
Resultados
Linear (Resultados)

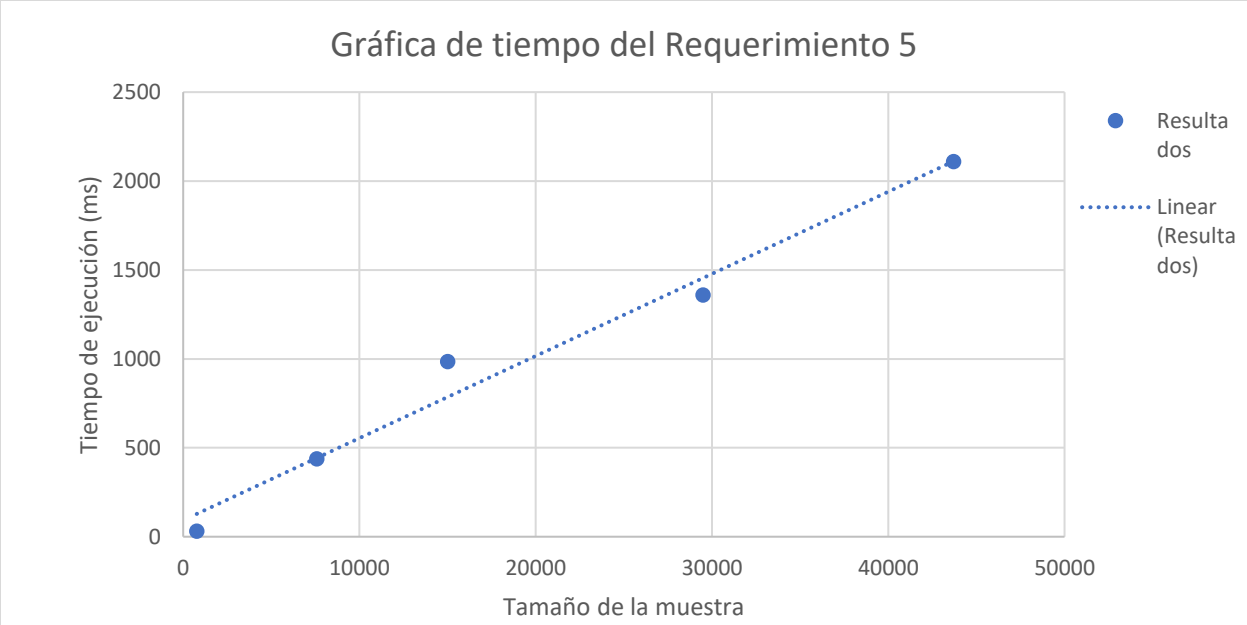

Gráfica de tiempo del Requerimiento 2 con función que la acota
Tiempo de ejecución (ms)
250000
200000
150000
100000
50000
0
0
10000
20000
30000
40000
50000
Tamaño de la muestra
Función que acota (nlog(n))
Resultados

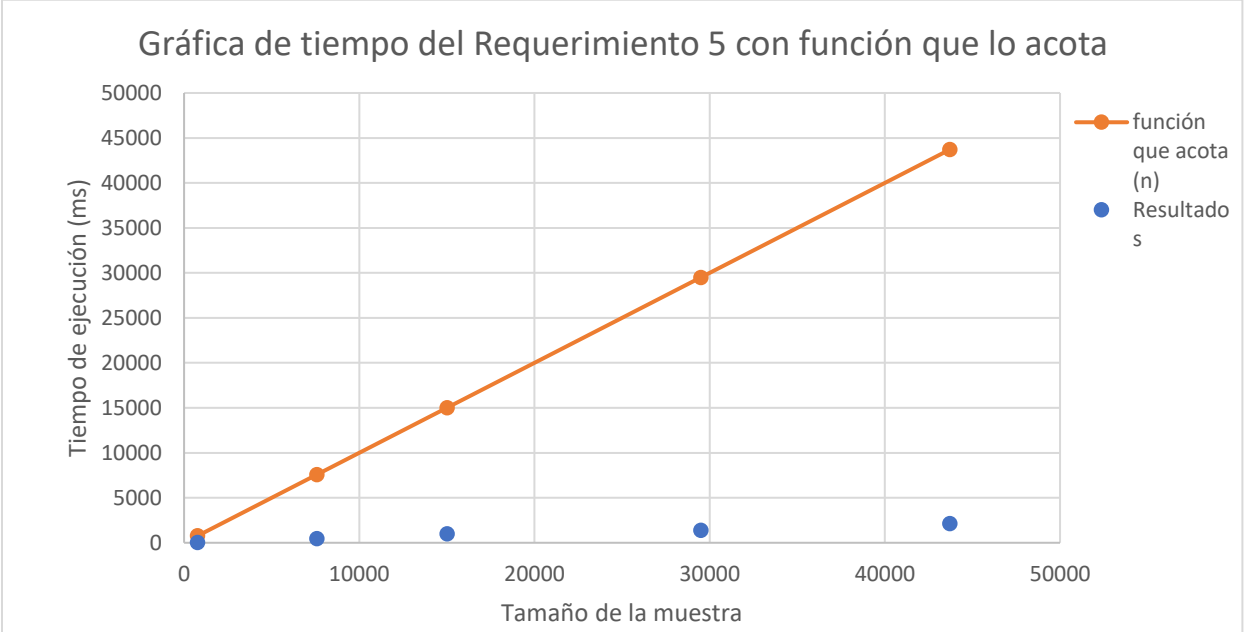

Gráfica de tiempo del Requerimiento 3
Tiempo de ejecución (ms)
5
4
3
2
1
0
-1
-2
0
2000
4000
6000
8000
10000
12000
Tamaño de la muestra
Resultados
Linear (Resultados)


Gráfica de tiempo del Requerimiento 3 con función que la acota
Tiempo de ejecución (ms)
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000
Tamaño de la muestra
Función que acota (n)
Resultados


Gráfica de tiempo del Requerimiento 4
Tiempo de ejecución (ms)
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
0
10000
20000
30000
40000
50000
Tamaño de la muestra
Resultados
Linear (Resultados)


Gráfica de tiempo del Requerimiento 4 con función que la acota
Tiempo de ejecución (ms)
250000
200000
150000
100000
50000
0
0
10000
20000
30000
40000
50000
Tamaño de la muestra
Función que acota (n*log(n))
Resultados





Análisis de los resultados obtenidos en las pruebas

Luego de realizar las pruebas de rendimiento temporal de todos los requerimientos pudimos notas que las complejidades temporales de muchos mejoraron significativamente en los casos promedios comparándolos con los resultados del reto 1. En el caso del requerimiento 1, se registró una amplia mejoría en los tiempos de ejecución respecto a la solución mediante TAD listas implementada en el primer requerimiento. El peor caso de ambas soluciones es el mismo O(n), cuando tenga que recorrer todos los años junto a todas las obras, pero en el caso promedio, la solución mediante TAD maps probó ser mucho más eficiente en el caso

promedio y también en el mejor caso, pues en las listas era necesario recorrer todas las obras de arte siempre, mientras que con TAD maps solo se extraían aquellas pertenecientes al rango dado, ahorrando mucho tiempo y disminuyendo la complejidad. En cuanto al requerimiento 2, la implementación del reto 2 muestra un menor tiempo de ejecución porque solo es necesario hacer un recorrido de los datos que se encuentran delimitados en un rango de fechas establecido por el usuario, mientras que en la implementación realizada para el reto 1, para lograr cumplir con el requerimiento, se hacía un ciclo que itera con todos los datos existentes en el archivo independientemente de si estos hacían parte del rango solicitado o no. Cabe mencionar que para el peor caso de ambos códigos la complejidad será de O(n), debido a que un rango de tiempo que abarque todos los datos obligará a un recorrido completo de los mismos, sin embargo, dado que el código implementado para el reto 2 solo itera ciclos si los datos pertenecen al rango de fechas, el mejor tiempo y el tiempo promedio en esta implementación siempre será mucho más eficiente que la solución planteada para el reto 1. Por otro lado, en el requerimiento 3, los tiempos de ejecución en el primer requerimiento fueron buenos, pero en este fueron mucho mejores, se pudo evidenciar que la solución dada mediante TAD maps mejoró muchísimo los tiempos de ejecución, en la mayoría de las pruebas se mostró prácticamente constante y la función que la acotó fue log(n), mientras que en la solución dada en el reto 1 se podía ver un comportamiento más lineal, que iba creciendo de manera proporcional al aumento de los datos. En el requerimiento 4, es evidente la gran diferencia en términos de tiempo que se logra observar al comparar la implementación de TAD listas y la de TAD maps, pues para el reto 1 era necesario iterar cada uno de los artistas existentes en los datos al igual que sus obras asociadas, mientras que en la nueva implementación del reto 2, el ciclo se limita a recorrer las nacionalidades existentes que ya se encuentran agrupadas, lo cuan reduce los tiempos sustancialmente, generando una respuesta casi inmediata y un comportamiento temporal que tiende a ser constante independientemente de la cantidad de dato. Por último, en el requerimiento 5 se vio una mejoría en los resultados obtenidos, pero la mejoría no se vio tan significativa respecto a la solución planteada en el reto 1. El único cambio implementado fue que ya no era necesario recorrer la lista buscando todas las obras de arte pertenecientes al departamento dado, ya que con los mapas ya las obras estaban clasificadas por departamento y con la función get y getValue fue posible extraer las obras de ese departamento en O(1) en vez de O(n), de igual manera el proceso para determinar el costo de transporte se mantuvo y al ser algo complejo que se debe calcular para todas las obras de arte sin excepciones, es un requerimiento que está sujeto a la cantidad de datos obligatoriamente, entonces a medida que la cantidad de datos aumenta, los tiempos de ejecución también lo hicieron. De igual manera, es importante tener en cuenta que comparar los únicamente los resultados temporales de ambos requerimientos es una prueba que está sujeta a muchos factores que pueden afectar los tiempos obtenidos, entonces es posible que algunas pruebas dieran tiempos mayores a pesar de que lógicamente los TAD maps agilizaron el proceso, por lo que esas diferencias temporales se le pueden atribuir al comportamiento de la máquina utilizada al momento de ejecutar las pruebas.