# Documento de análisis del reto 1

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:*

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 que se 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)).

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-utf8-10pct.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 "Artworcks" 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-utf8-10pct.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 addArtworkArtist, 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. **addArtworkArtist:** el propósito de esta función es relacionar las obras de arte con su respectivo artista, para ello se utiliza isPresent que permite reconocer si el ID ingresado pertenece a algún artista del archivo, de tal forma que si lo anterior es "True" se retorna un valor menor a cero y se añade este dato a la lista artworks, la cual se encuentra almacenada en el diccionario de los artistas y es de tipo ARRAY_LIST.

v. **addArtistDate:** el objetivo de la función es añadir a la lista ArtistDate creada en el catálogo, un diccionario de cinco llaves con la información de un artista teniendo en cuenta el año encontrado en la columna BeginDate del archivo. La única condición para que esto se ejecute es que el año que ingresa sea diferente de cero. Esta función se ejecuta dentro de addArtist.

vi. **addArtworkDate:** la función consiste en añadir a la lista ArtworkDate (creada en el catálogo), un diccionario de seis llaves con la información de una obra de arte teniendo en cuenta la fecha encontrada en la columna DateAquire del archivo.LA condición para que lo anterior se ejecute es que la fecha que ingresa no corresponda a un dato string vacio. Esta función se ejecuta dentro de addArtwork.

*b.* *Funciones para la creación de datos:*

i. **newArtistDate:** Se encarga de crear un diccionario con la información necesaria para el requerimiento 1 del Reto, y esta luego es agregada a la lista final del requerimiento.

ii. **newArtworkDate:** Se encarga de crear un diccionario con la información necesaria para el requerimiento 2 del Reto, y esta luego es agregada a la lista final del requerimiento.

*c.* *Funciones utilizadas para comparar elementos dentro de una lista:*

i. **cmpartistyear:** Esta función de comparación se encarga de comparar las fechas de nacimiento de dos artistas y verifica si una es menor que la otra. Es utilizada por la función de ordenamiento del requerimiento.

ii. **cmpartworkyear:** Esta función de comparación se encarga de comparar las fechas de adquisición de dos obras de arte y verifica si una es menor que la otra. Es utilizada por la función de ordenamiento del requerimiento.

iii. **cmpartistID:** Esta función es utilizada para comparar los ArtistID de dos artistas, si son iguales arroja un 0, de lo contrario, arroja un -1. Esta función es utilizada como una cmpfunction de la lista Catalog['Artist'].

iv. **cmpArtistTecnique:** Esta función es utilizada para comparar las técnicas de dos obras de arte, si son iguales arroja un 0, de lo contrario, arroja un -1. Esta función es utilizada como una cmpfunction de la lista tecniques_list del requerimiento 3.

v. **cmpArtistNationality:** la función compara que dos datos que entran, los cuales corresponden a nacionalidades sean iguales, especificando que ambos strings se encuentran en minúscula. Si son iguales se retorna 0 y si son distintos el valor será de -1. Esta función es utilizada como una cmpfunction de la lista nationality_artworks para el requerimiento 4.

vi. **cmpTecniquesize:** Esta función es utilizada para comparar el tamaño de la lista de obras de arte asociadas a una técnica de dos técnicas diferentes, si son iguales arroja un 0, de lo contrario, arroja un -1. Es utilizada por la función de ordenamiento del requerimiento.

vii. **cmpNationalitysize:** La función hace una comparación del tamaño de la lista de obra de arte asociada a dos nacionalidades distintas de tal forma que si estas

son iguales el valor de retorno será un 0 mientras que si una es mayor que la otra retornara un -1. Es utilizada por la función de ordenamiento del requerimiento 4.

viii. **cmpTranspCost:** Esta función es utilizada para comparar el costo de transporte de dos obras de arte y pregunta si la primera es mayor a la otra para poder organizar de mayor a menor. Es utilizada por la función de ordenamiento según costos de la lista transp_cost del requerimiento 5.

ix. **cmpTranspOld:** Esta función es utilizada para comparar la fecha de dos obras de arte y pregunta si la primera es menor a la otra para poder organizar de menor a mayor. Es utilizada por la función de ordenamiento según fechas de la lista copy del requerimiento 5.

d. Funciones de ordenamiento:

```
# Funciones de ordenamiento

def sortYear_Artist(artist_inrange):

    ms.sort(artist_inrange, cmpartistyear)


def sortYear_Artwork(artwork_inrange):

    ms.sort(artwork_inrange, cmpartworkyear)

def sortTecnique_size(tecnique_list):

    ms.sort(tecnique_list, cmpTecniquesize)
def sortNationalitysize(nationalities):

    ms.sort(nationalities, cmpNationalitysize)
def sortTransportation(transp_cost):

    ms.sort(transp_cost, cmpTranspCost)

def sortTranspOld(copy):
    ms.sort(copy, cmpTranspOld)
```

Como es posible observar en la imagen anterior, todas las funciones de ordenamiento tienen la misma estructura y usan el algoritmo de ordenamiento “merge sort”, este fue el algoritmo que elegimos para todos los ordenamientos, pues la complejidad de este es constante sin importar los datos que le sean suministrados, su complejidad siempre es de O(nlog(n)). En las pruebas del laboratorio 4, fue uno de los que mejores tiempos registró junto con “Insertion Sort”. La razón por la que no escogimos “Insertion Sort” fue porque a pesar de que también es estable y es inplace, su peor caso y caso promedio es $O(n^2)$, una complejidad mucho más alta que la que ofrece merge sort en su peor caso.

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

```python
# Funciones de consulta
def getArtistYear(catalog,año_inicial,año_final):

    artist_inrange = lt.newList("ARRAY_LIST")

    for artist in lt.iterator(catalog['ArtistDate']):

        if int(artist['BeginDate']) >= año_inicial and int(artist['BeginDate']) <= año_final:

            lt.addLast(artist_inrange, artist)

    sortYear_Artist(artist_inrange)
    return artist_inrange
```

En el código de arriba 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. Por un lado, al contar con un for loop que recorre toda la lista de catalog[‘ArtistDate’], este da una complejidad de O(n), mientras que la función de sortYear_Artist utilizada para ordenar la lista de los artistas dentro del rango especificado (como utiliza la función de ordenamiento merge), tiene una complejidad constante de O(nlog(n)). En total, la complejidad de este requerimiento sería de O(n) + O(nlog(n)).

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

```python
def getArtworkYear(catalog,año_inicial,año_final):
    start_time = time.process_time()

    artwork_inrange = lt.newList("ARRAY_LIST")

    #Fechas ingresadas
    año_i = año_inicial.split("-")
    di = d.datetime(int(año_i[0]),int(año_i[1]), int(año_i[2]))
    año_f = año_final.split("-")
    df = d.datetime(int(año_f[0]),int(año_f[1]), int(año_f[2]))

    #Fechas del csv
    for artwork in lt.iterator(catalog['ArtworkDate']):
        date = artwork['DateAcquired'].split("-")
        d1 = d.datetime(int(date[0]),int(date[1]), int(date[2]))

        if d1 >= di and d1 <= df:

            lt.addLast(artwork_inrange, artwork )
    sortYear_Artwork(artwork_inrange)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return artwork_inrange, elapsed_time_mseg
```

En este requerimiento fue necesario listar las obras de arte que fueron adquiridas dentro de un rango de tiempo especificado por el usuario. Para la estimación de su complejidad, es posible notar que hay un único for loop que recorre catalog[‘ArtworkDate’] el cual tiene un tamaño de n y esto significa que su complejidad es de O(n), mientras que la función de ordenamiento de la lista resultante (como utiliza la función de ordenamiento merge) tiene una complejidad temporal de O(nlog(n)). En total, la complejidad temporal de esta función es O(n) + O(nlog(n)).

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

```
def getArtistTecnique(catalog,name):

    '''
    Crea una lista nueva donde se van a ir clasificando las obras de arte de un artista según la técnica empleada.
    '''
    start_time = time.process_time()
    tecniques_list = lt.newList('ARRAY_LIST', cmpfunction=cmpArtistTecnique)

    for artist in lt.iterator(catalog['Artist']):
        if name.lower() in artist['DisplayName'].lower():
            total_obras = lt.size(artist['Artworks'])
            for artwork in lt.iterator(artist['Artworks']):
                medium = artwork['Medium']
                postechnique = lt.isPresent(tecniques_list, medium)
                artwork_filtrada = {'Title': artwork['Title'],
                                    'Date': artwork['Date'],
                                    'Medium': artwork['Medium'],
                                    'Dimensions': artwork['Dimensions']}

                if postechnique > 0:
                    tecnique = lt.getElement(tecniques_list,postechnique)
                    lt.addLast(tecnique['Artworks'], artwork_filtrada)
                else:

                    tec = {'Tecnique': medium,
                           'Artworks': lt.newList('ARRAY_LIST')}

                    lt.addLast(tec['Artworks'], artwork_filtrada)
                    lt.addLast(tecniques_list, tec)

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

En este requerimiento era necesario clasificar las obras de un artista por las técnicas empleadas. Para esto fue necesario emplear dos for loops anidados, en el primer for loop se hace búsqueda del artista que se está consultando y luego, una vez este sea encontrado, se recorren las obras de arte asociadas a este, para esto es que se realiza el segundo for loop. En este segundo for se realiza la clasificación por técnicas y se va agregando a una lista auxiliar las técnicas con el listado de sus obras asociadas. Para el cálculo de las complejidades tendremos en cuenta el peor caso, en el que el artista que buscamos se encuentre al final de la lista de artistas y toque recorrerla por completo, esto sería de complejidad O(n), luego con el for loop anidado, es muy poco probable que un artista sea dueño de todas las obras del museo, por lo que sería prácticamente imposible que en este segundo for loop se tuviera una complejidad de O(m) (m siendo el total de obras), por ende, asumiremos que es O(k), siendo una cantidad más plausible de obras de arte asociadas a un artista. Por último, para facilitar la búsqueda de las técnicas con mayor número de obras asociadas, se realizó un sort según el tamaño de la lista de artworks de cada técnica, el cual tiene una complejidad de O(nlog(n)). La complejidad final del algoritmo sería de O(n*k) + O(nlog(n)).

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

```python
def getArtistNationality(catalog):
    '''
    Crea una lista nueva donde se  clasifican las obras de arte  según la nacionalidad del artista.
    '''
    start_time = time.process_time()

    nationality_artworks = lt.newList('ARRAY_LIST', cmpfunction=cmpArtistNationality)

    for artist in lt.iterator(catalog['Artist']):

        nationality = artist['Nationality']
        if nationality == "" or nationality == "nationalityunknown":
            nationality = "Unknown"

        nation = lt.isPresent(nationality_artworks, nationality)
        artist_artworks = artist['Artworks']
        if artist_artworks != 0:
            if nation > 0:
                nation_works = lt.getElement(nationality_artworks,nation)
                #lt.addLast(nationality_list, nationality)
            else:
                nation_works = {'Nationality': nationality,
                                'Artworks': lt.newList('ARRAY_LIST') }
                lt.addLast(nationality_artworks, nation_works)

            for work in lt.iterator(artist_artworks):
                lt.addLast(nation_works["Artworks"], work)

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

```

Para la solución del requerimiento 4 se creó una nueva lista llamada nationality_artworks de tipo ARRAY_LIST en la cual se busca añadir paulatinamente por medio de dos bucles for las obras de arte creadas por un artista de una nacionalidad en específico. De esta forma, todas las obras cargadas quedaran clasificadas por la nacionalidad de su respectivo artista. En el primer ciclo for se recorre el catálogo de artistas y se extrae de cada uno de ellos su respectiva nacionalidad y la lista de obras de arte creadas por el/ella. A continuación, se utiliza isPresent para determinar si la nacionalidad ya existe en la lista creada lo cual asegura que no abran datos repetidos y que todas las obras de una nacionalidad serán añadidas a dicha nacionalidad. En este orden de ideas, si la nacionalidad aun no pertenece a la lista, esta será añadida al final de nationality_artworks con las obras asociadas, de lo contrario, si la nacionalidad ya había sido añadida simplemente se añadirán obras, todo si y solo si artista_arworks (donde se encuentran las obras de una artista) es diferente de cero (complejidad O(n)). Adicionalmente, el segundo for se encarga de recorrer artista_artworks e ir añadiendo en la última posición todas las obras del artista en cada iteración (complejidad O(m) siendo m<n). Cabe resaltar que dentro de la función se consideró excepción donde la nacionalidad se presenta como un string vacío o con el nombre “nationality unknown” para estos casos la variable nationality toma el nombre de Unknown. Finalmente, se realiza un sort con el tamaño de la lista de artworks para cada nacionalidad y así determinar cual de ellas tiene una mayor cantidad de obras asociadas (O(nlog(n))). La complejidad final del algoritmo es de O(n*m) + O(nlog(n)).

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

```
def getTransportationCost(catalog, dpto):
    start_time = time.process_time()
    costo_total = 0
    transp_cost = lt.newList('ARRAY_LIST')
    artworksBydpto = lt.newList('ARRAY_LIST')

    for artwork in lt.iterator(catalog['Artwork']):

        if artwork['Department'].lower() == dpto.lower():

            lt.addLast(artworksBydpto, artwork)

    for artwork in lt.iterator(artworksBydpto):
        artwork_filtrada = {'Title': artwork['Title'],
                            'Artist/s':artwork['ConstituentID'],
                            '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= transp_cost.copy()
    sortTranspOld(copy)
    sortTransportation(transp_cost)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000
    return transp_cost, elapsed_time_mseg,round(costo_total,2), copy, round(peso_total, 2)
```

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 dos listas auxiliares, artworksBydpto es utilizada para guardar las obras de arte de ese departamento y transp_cost 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, se realizan 2 for loops no anidados, en el primero se recorren todas las obras de arte lo cual genera una complejidad de O(n) en el peor de los casos, mientras que el otro loop recorre únicamente las obras del departamento generando una complejidad de O(m) siendo m el total 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(m) + O(nlog(n)).

## VII. Pruebas Temporales de cada requerimiento

### Ambientes de pruebas

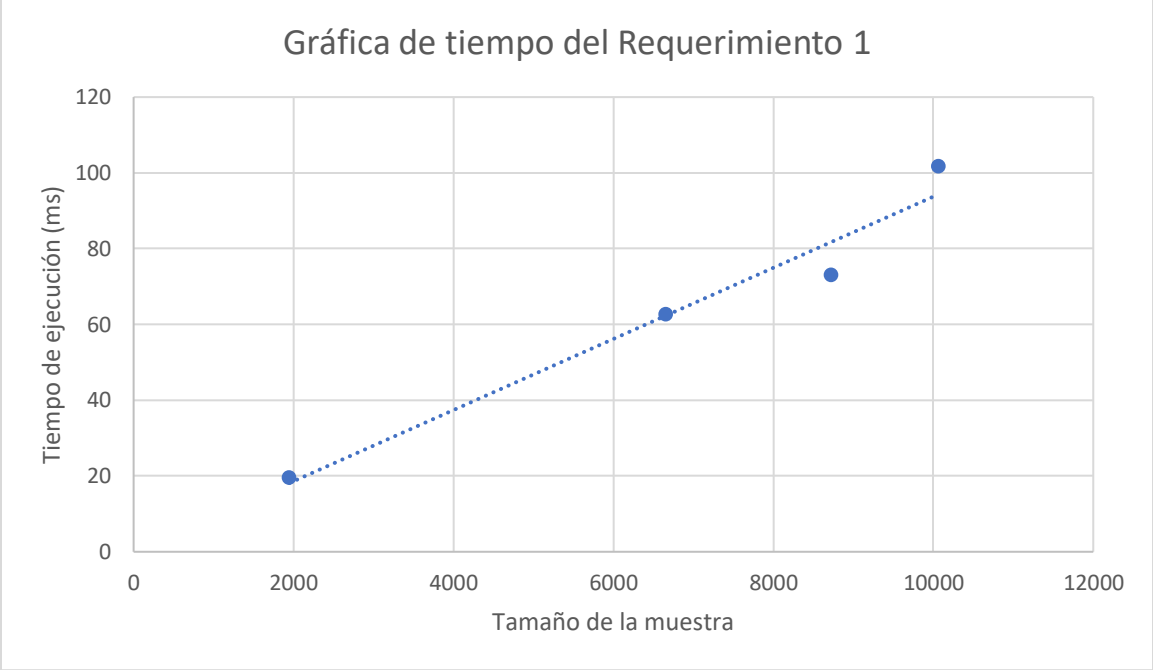
| | **Máquina** |
|---|---|
| **Procesadores** | AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz |
| **Memoria RAM (GB)** | 12,0 GB (9,95 GB usable) |
| **Sistema Operativo** | Windows 10 64-bit operating system |

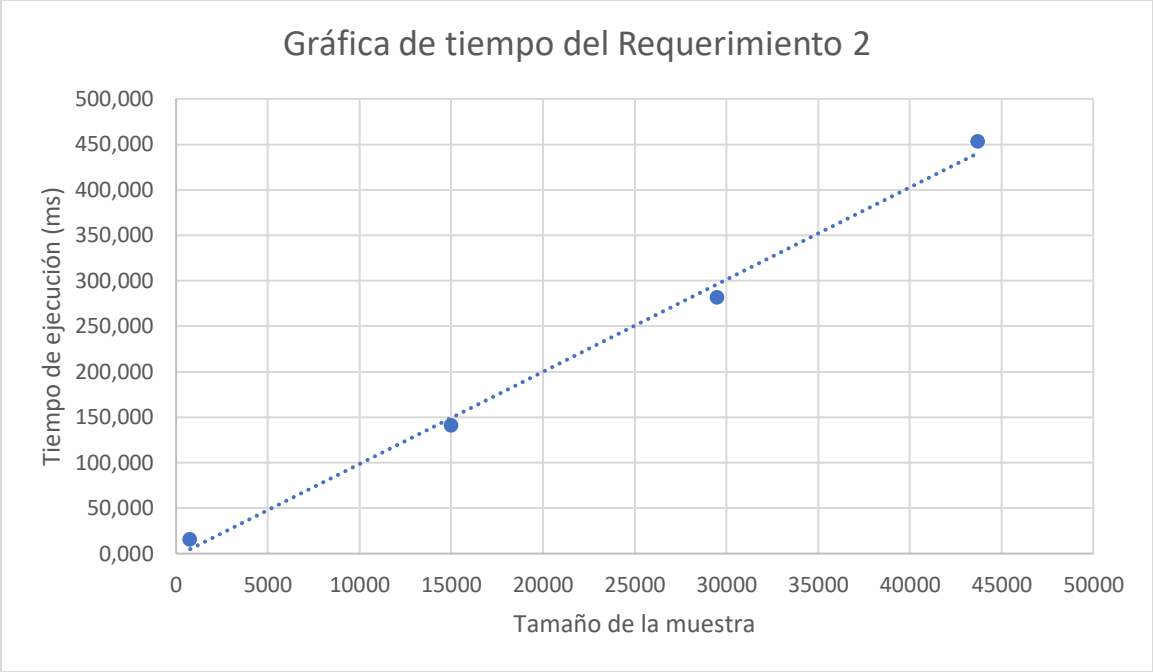
*Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.*

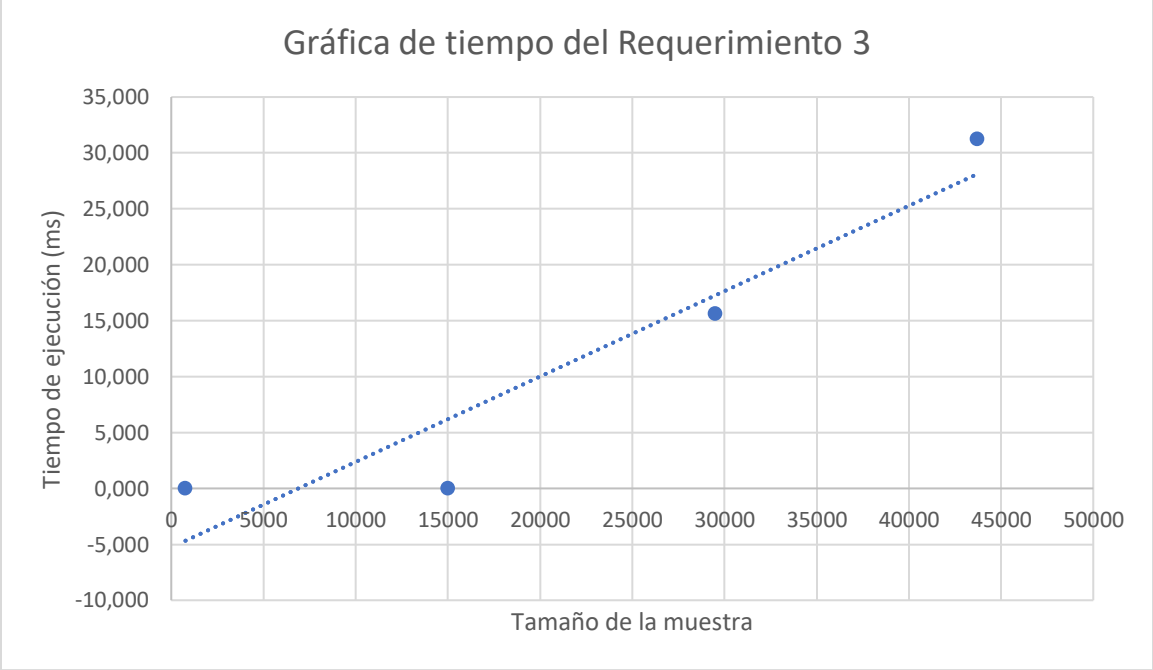
**Máquina: Resultados**
**En el req-1 se usan rangos de 1920-1985 y 1920-1930**
**En el req-2 se usan rangos de 1944-06-06 hasta 1989-11-09 y rangos de un año de diferencia**
**En el req-3 se está usando al artista: Ivan Kliun**
**En el req-5 se usa drawings & prints**

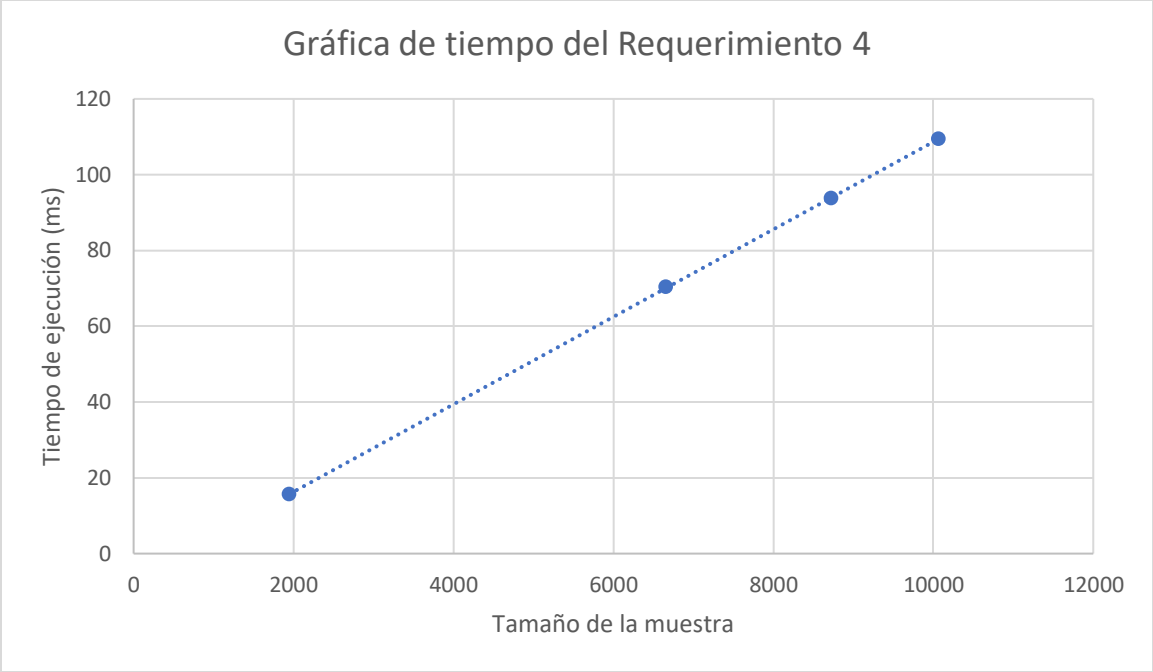
| Porcentaje de la muestra [pct] | Tamaño de la muestra | Req-1 [ms] | Req-2 [ms] | Req-3 [ms] | Req-4 [ms] | Req-5 [ms] |
|---|---|---|---|---|---|---|
| 0.50% | Artistas = 1948 obras = 768 | 19.53 | 15.625 | 0.0 | 15.625 | 31.25 |
| 10.00% | Artistas = 6656 obras = 15008 | 62.50 | 140.62 | 0.0 | 70.3125 | 656.2 |
| 20.00% | Artistas = 8724 obras = 29489 | 72.91 | 281.23 | 15.625 | 93.75 | 1296.875 |
| 30.00% | Artistas = 10063 obras = 43704 | 101.6 | 453.1 | 31.25 | 109.37 | 2140.625 |

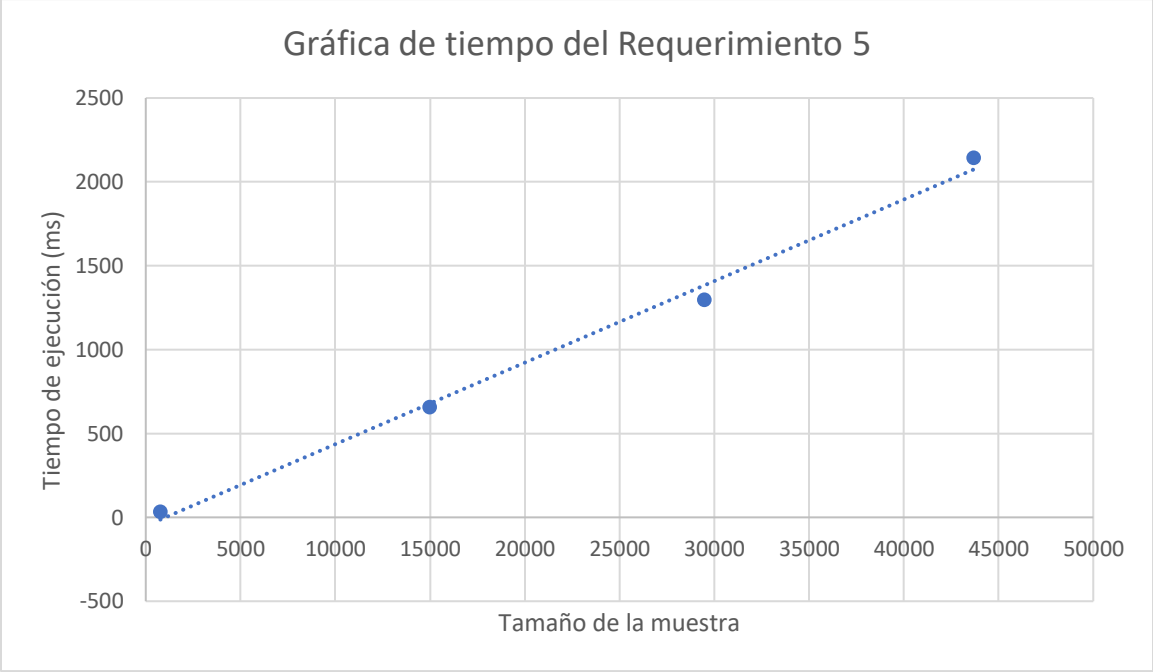
*Tabla 2. Comparación de tiempos de ejecución para los ordenamientos en la representación arreglo.*




Gráfica de tiempo del Requerimiento 2
Tiempo de ejecución (ms)
500,000
450,000
400,000
350,000
300,000
250,000
200,000
150,000
100,000
50,000
0,000
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
Tamaño de la muestra


Gráfica de tiempo del Requerimiento 3
Tiempo de ejecución (ms)
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0,000
-5,000
-10,000
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
Tamaño de la muestra




En las gráficas es posible evidenciar que las complejidades de la mayoría de los requerimientos siguen una tendencia lineal, en un aspecto global, pues de vez en cuando hay valores que se salen de la línea de tendencia. Por ejemplo, en los requerimientos 1 y 3, no se sigue tanto este comportamiento lineal que si se puede ver en las gráficas de los requerimientos 2,4 y 5. Esto nos puede dar un indicio de que puede

que haya una inconsistencia generada al momento de tomar las pruebas con la máquina, ya sea porque había algún programa en el fondo que estaba retardando el proceso en ese momento, o simplemente problemas con las velocidades alcanzadas por el procesador de la máquina. En cuanto a las complejidades que se pueden observar, se puede ver que el orden de crecimiento temporal de las funciones 1,2,4 y 5 parecen ser del orden de entre O(nlog(n)), que de cierta manera coincide con el análisis de complejidades realizado anteriormente. La única gráfica que se sale de ese modelo es el de la gráfica 3, la cual parece describir más un orden de O(n^2). Esto se debe a que hay un error en la carga de las obras de arte de los artistas a cada artista y esto afecta enormemente al requerimiento 3, pues, aunque su código funcione, no se ejecutará de la manera esperada.