OBSERVACIONES & ANÁLISIS – RETO 1

Req3 - María Alejandra Estrada García Cod. 202021060

Req4 - Santiago Martínez Novoa Cod. 202112020

Estimación de complejidad para cada requerimiento datos:

1. Se hacen por medio de la notación big O, la cual presenta el límite superior para cada algoritmo.
2. La complejidad de las funciones de la librería DISClib se desestimó, pues ninguna de las usadas con mayor frecuencia (NewList, addLast, size, getElement) tiene una complejidad diferente a O(1).
3. La carga de datos fue implementada con una estructura de datos 'ARRAY_LIST', debido a que su complejidad es inferior que una lista, y la que menos tiempo tarda en ordenar los datos.
4. Se utilizó para ordenar los datos 'merge sort', debido a que su complejidad es menor y constante.

Requerimientos:

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

El primer requerimiento se reparte en tres funciones, donde la primera es getArtistByDate(catalog, BeginDate, EndDate) la cual 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, En esta función se hace un ciclo para cada artista, y se llama a la función SortDates, el cual organiza las fechas, y tiene una complejidad O(NlogN). Por lo tanto, esta función tiene una complejidad de O(N + log(N)). Por otro lado, esta función tiene sus datos a partir de las funciones newArtistDate y addArtistDate(catalog, listArtist), las cuales crean las listas con las llaves para cada uno, estas tienen una complejidad de O(1) y O(1) respectivamente. Por lo que la complejidad temporal final será: O(N + N*log(N)).

Funciones principales del requerimiento:

- getArtistByDate(catalog, BeginDate, EndDate):

```
def getArtistByDate(catalog, BeginDate, EndDate):


    DatesArtist = lt.newList('ARRAY_LIST')

    for a in lt.iterator(catalog['ArtistsDate']):
        if int(a['BeginDate']) >= BeginDate and int(a['BeginDate']) <= EndDate and a['BeginDate'] != "" and a['BeginDate'] != 0:
            lt.addLast(DatesArtist, a)

    Dates_Artist = SortDates(DatesArtist)

    return Dates_Artist
```

- newArtistDate(artist, BeginDate, EndDate, nationality, gender):

```
def newArtistDate(artist, BeginDate, EndDate, nationality, gender):
    artistDate = {'Name': '', 'BeginDate': '', 'EndDate': '', 'Nationality': '', 'Gender': ''}
    artistDate['Name'] = artist
    artistDate['BeginDate'] = BeginDate
    artistDate['EndDate'] = EndDate
    artistDate['Nationality'] = nationality
    artistDate['Gender'] = gender


    return artistDate
```

- addArtistDate(catalog, listArtist):

```
def addArtistDate(catalog, listArtist):
    addDate = newArtistDate(listArtist['DisplayName'], listArtist['BeginDate'], listArtist['EndDate'],
                            listArtist['Nationality'], listArtist['Gender'])
    lt.addLast(catalog['ArtistsDate'], addDate)
```

Complejidad temporal final = O(1) + O(logN) + O(N) = O(N) + O(log(N))

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 artworksByDate(catalog, inicial, final). En esta función se hace un ciclo con dos condicionales, para verificar en el primero que la llave de 'DateAcquired' no sea vació o '0', el otro para verificar si dicho valor este dentro del rango establecido por el usuario, posteriormente se llama a la función sortDateAcquired para organizar los valores en la lista de menor a mayor. Esta función tiene complejidad de O(N) + O(Nlog(N)) = O(N Log(N)). Asimismo, se crearon funciones para cada acción que podía ocurrir con una obra de arte (si es comprada y una función para crear los datos requeridos en el view.py), para estas funciones la complejidad será de O(1), excepto para la función artworksPurchased(sort_DateAcquired) donde es de O(N) pues hay un ciclo que se ejecuta completamente cada vez que se llama a esa función. Por lo tanto la complejidad final es:

Complejidad temporal final = O(N Log(N)

Funciones:

- AddArtworkDAcquired(catalog, listArtwork):

```
def addArtworkDAcquired(catalog, listArtwork):
    addDateAcquired = newArtworksDateAcquired(listArtwork['ObjectID'], listArtwork['Title'], listArtwork['Medium'],
                                              listArtwork['Dimensions'], listArtwork['Date'], listArtwork['DateAcquired'],
                                              listArtwork['CreditLine'])
    lt.addLast(catalog['ArtworksDateAcquired'], addDateAcquired)
```

- newArtworksDateAcquired(ObjectID, artwork, ConstituentID, Medium, Dimensions, Date, DateAcquired, CreditLine):

```
def newArtworksDateAcquired(ObjectID, artwork, ConstituentID, Medium, Dimensions, Date, DateAcquired, CreditLine):
    ArtworkDateAcquired = {'ObjectID': '', 'Title': '', 'ConstituentID': '', 'Medium': '', 'Dimensions': '',
    'Date':'', 'DateAcquired': ''}
    ArtworkDateAcquired['ObjectID'] = ObjectID
    ArtworkDateAcquired['Title'] = artwork
    ArtworkDateAcquired['ConstituentID'] = ConstituentID
    ArtworkDateAcquired['Medium'] = Medium
    ArtworkDateAcquired['Dimensions'] = Dimensions
    ArtworkDateAcquired['Date'] = Date
    ArtworkDateAcquired['DateAcquired'] = DateAcquired
    ArtworkDateAcquired['CreditLine'] = CreditLine

    return ArtworkDateAcquired
```

- artworksPurchased(sort_DateAcquired):

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

    return count
```

- artworksByDate(catalog, inicial, final):

```
def artworksByDate(catalog, inicial, final):

    artworksDate = lt.newList('ARRAY_LIST')

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

    for a in lt.iterator(catalog['ArtworksDateAcquired']):
        if a['DateAcquired'] != '' and a['DateAcquired']!='0':
            a1 = date.fromisoformat(a['DateAcquired'])
            if a1 >= inicialDate and a1 <= finalDate and a1 != '' and a1!='0':
                lt.addLast(artworksDate, a)


    sort_DateAcquired = sortDateAcquired(artworksDate)

    return sort_DateAcquired
```

- sortDateAcquired(artworksDate):

```
def sortDateAcquired(artworksDate):

    start_time = time.process_time()
    sorted_list = mergesort.sort(artworksDate, compDateAcquired)
    stop_time = time.process_time()
    elapsed_time_mseg = (stop_time - start_time)*1000

    return sorted_list, elapsed_time_mseg
```

- CompDateAcquired(a1, a2):

```
def compDateAcquired(Date1, Date2):
    if Date1['DateAcquired'] != '' and Date1['DateAcquired'] != '0' and Date2['DateAcquired'] != '0' and Date2['DateAcquired'] != '':
        return (date.fromisoformat(Date1['DateAcquired']) < date.fromisoformat(Date2['DateAcquired']))
```

Complejidad temporal final =

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 getArtistByTecnique(catalog, Artistname). En esta función se hace dos ciclos for y se crea una lista vació, donde se van a

meter las técnicas utilizadas por los artistas con sus obras. Primero se revisa si el artista está en el catálogo de artistas, luego por medio de un condicional if se revisa si el Artistname que me entró como parámetro (fue dado por el usuario), es igual a unos de los nombres de los artistas O(N). Posteriormente, se hace otro ciclo, donde se revisa si una obra está dentro del catálogo de obras, O(X), en el siguiente paso buscamos por medios de condicionales si la tecnica está en la lista hecha al principio y se le va agregando la técnica que el artista utilizó con su(s) obra(s). La lista se organiza por medio de comparaciones, compATecnique, el cual tiene complejidad O(1) y por medio de un sort, ArtworkTecniqueSort(ArtworkTecnique), el cual se usa otra comparación, compTec(tec1, tec2); estas funciones tienen complejidad O(logN) y O(1). En consecuencia, la complejidad temporal final del algoritmo es: O(N*X)

Complejidad temporal final = O(N*X) + O(N*logN)

Funciones:

- getArtistByTecnique(catalog, Artistname):

```
#Req 3:
def getArtistByTecnique(catalog, Artistname):
    ArtistTecnique = lt.newList('ARRAY_LIST', cmpfunction=compATecnique)
    for artists in lt.iterator(catalog['Artists']):
        if artists['DisplayName'] == Artistname:
            for a in lt.iterator(artists['Artworks']):
                tecnique = a['Medium']
                pos = lt.isPresent(ArtistTecnique,tecnique)
                if pos > 0:
                    tec = lt.getElement(ArtistTecnique, pos)
                    lt.addLast(tec['Artworks'], a)
                else:
                    tec = newTecnique(a['Medium'])
                    lt.addLast(ArtistTecnique, tec)
                lt.addLast(tec['Artworks'], a)

    return ArtistTecnique
```

- newTecnique(tecnique):

```
def newTecnique(tecnique):
    artec = {'MediumName': '',
             'Artworks': lt.newList('ARRAY_LIST')}
    artec['MediumName'] = tecnique
    return artec
```

- CompATecnique(tec, artistTecnique):

```
def compATecnique(tec, artistTecnique):
    if tec.lower() == artistTecnique['MediumName'].lower():
        return 0
    else:
        return -1
```

- ArtworkTecniqueSort(ArtworkTecnique):

```
def ArtworkTecniqueSort(ArtworkTecnique):
    #start_time = time.process_time()
    sort_list = mergesort.sort(ArtworkTecnique, cmpfunction=compTec)
    #stop_time = time.process_time()
    #elapsed_time_mseg = (stop_time - start_time)*1000
    return sort_list
```

- compTec(tec1, tec2):

```
def compTec(tec1, tec2):
    return lt.size(tec1['Artworks']) > lt.size(tec2['Artworks'])
```

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

- El cuarto requerimiento se reparte en 4 funciones. Estas se encargan de iterar sobre todos los artistas y sus respectivas obras de artes para organizarlas por nacionalidades. La función principal del requerimiento es getArtworksByNationality(catalog), la cual llama a las demás a su debido tiempo. La complejidad de esta función es de O(n^2), pues cuenta con dos ciclos entrelazados entre sí. Como son ciclos de tipo *for* significa que tanto en el mejor caso como el peor caso la complejidad será de O(n^2). Si observamos las siguientes funciones nos damos cuenta de que, para NewNationality(Nationality), Comparenationality(ListNationality, Nationality) y compArtworkNation(N1, N2), la complejidad es de O(1). Finalmente, para la función de ordenamiento la complejidad es de O(N Log(N)).

Complejidad temporal final = O(n^2)

Funciones:

- getArtworksByNationality(catalog):

```
def getArtworksByNationality(catalog):
    ArtistNationality = lt.newList('ARRAY_LIST', cmpfunction=comparenationality)
    for artists in lt.iterator(catalog['Artists']):
        for a in lt.iterator(artists['Artworks']):
            Nation = artists['Nationality']
            position = lt.isPresent(ArtistNationality,Nation)
            if position > 0:
                Nation = lt.getElement(ArtistNationality, position)
                trabajo = lt.newList("ARRAY_LIST")
                lt.addLast(trabajo,a)
                lt.addLast(trabajo,artists["DisplayName"])
                lt.addLast(Nation['Artworks'], trabajo)
            else:
                newnation = newNationality(artists['Nationality'])
                trabajo = lt.newList("ARRAY_LIST")
                lt.addLast(trabajo,a)
                lt.addLast(trabajo,artists["DisplayName"])
                lt.addLast(ArtistNationality, newnation)
                lt.addLast(newnation['Artworks'], trabajo)
    sorted_list = sortArtworkNationality(ArtistNationality)
    return sorted_list
```

- NewNationality(Nationality):

```
def newNationality(Nationality):
    artnat = {'Nationality': '',
              'Artworks': lt.newList('ARRAY_LIST')}
    artnat['Nationality'] = Nationality
    return artnat
```

- Comparenationality(ListNationality, Nationality):

```
def comparenationality(Nation,Nations):
    if Nation.lower() == Nations['Nationality'].lower():
        return 0
    else:
        return -1
```

- compArtworkNation(N1, N2):

```
def compArtworkNation(N1, N2):
    return int(lt.size(N1['Artworks'])) > int(lt.size(N2['Artworks']))
```

- sortArtworkNationality(ArtistNationality):

```
def sortArtworkNationality(ArtistNationality):
    sort_list = mergesort.sort(ArtistNationality, compArtworkNation)
    return sort_list
```

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

El quinto requerimiento se reparte en 5 funciones. Las funciones consiguen organizar la información por un departamento y organizarlo de diferentes maneras para encontrar varios tipos de información. Las complejidades encontradas para cada una de las funciones son las siguientes:

- GetArtworksByDepartment(catalog, departament)(incluyendo el ordenamiento): O(N+ N Log(N)) = O(N Log(N))
- precioest(ArtworkinCategory): O(N)
- pesoest(ArtworkinCateogory, category): O(N)
- compareprice(p1,p2): O(1)
- Compareage(a1,a2): O(1)

Como se puede observar, ya se han simplificado las complejidades a su expresión general.

Complejidad temporal final = O(N Log(N))

Funciones=

- GetArtworksByDepartment(catalog, departament):

```python
def getArtworksByDepartment(catalog, department):
    start_time = time.process_time()
    ArtworkinCategory = lt.newList('ARRAY_LIST',cmpfunction=compACategory)

    for artworks in lt.iterator(catalog["Artwork"]):
        if artworks['Department'] == department:
            lt.addLast(ArtworkinCategory,artworks)

    listaconprecio = precioest(ArtworkinCategory)
    pesoestim = pesoest(ArtworkinCategory, "Weight")
    precioestim = pesoest(ArtworkinCategory, "Price")

    sorted_listbyprice = mergesort.sort(listaconprecio, compareprice)
    artworkingsub = lt.subList(listaconprecio,1, lt.size(ArtworkinCategory))

    sorted_listbyage = mergesort.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
```

- precioest(ArtworkinCategory):

```python
def precioest(ArtworkinCategory):

    for artworks in lt.iterator(ArtworkinCategory):
        if artworks["Weight"] == '':
            porPeso = 0
        else:
            porPeso = round(72 * float(artworks["Weight"]),4)
        if (artworks["Height"] == '' or artworks["Width"] == '') and artworks["Diameter"] == '':
            porArea = 0
        elif artworks["Diameter"] != '':
            radius = float(artworks["Diameter"])/200
            porArea = round((radius)**2*(3.1415)*72, 4)
        else:
            porArea = round(((float(artworks["Height"])*float(artworks["Width"]))/ 10000)*72,4)
        if (artworks["Height"] == '' or artworks["Width"] == '' or artworks["Length"] == ''):
            porVol = 0
        else:
            porVol = round(((float(artworks["Height"])*float(artworks["Width"])*float(artworks["Length"]))/ 1000000)*72,4)

        if porVol == 0 and porArea == 0 and porPeso == 0:
            precio_final = 48
        else:
            precio_final = max(porPeso,porArea,porVol)
        artworks['Price'] = precio_final
    return ArtworkinCategory
```

- pesoest(ArtworkinCateogory, category):

```python
def pesoest(ArtworkinCategory, category):
    suma = 0
    for artworks in lt.iterator(ArtworkinCategory):
        if artworks[category] != '':
            suma += float(artworks[category])
    return round(suma,4)
```

- compareprice(p1,p2):

```python
def compareprice(p1,p2):
    return (float(p1['Price']) > float(p2['Price']))
```

- Compareage(a1,a2):

```python
def compareage(a1,a2):
    if a1['Date'] != '' and a1['Date'] != '0' and a2['Date'] != '' and a2['Date'] != '0':
        return (int(a1['Date']) < int(a2['Date']))
```

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.

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

Máquina	*Tamaño de muestra*	*Tiempo promedio*
1	2716	15.625
1	12658	140.625
1	21664	156.25
1	38213	171.875

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

Máquina	*Tamaño de muestra*	*Tiempo promedio*
1	2716	31.25
1	12658	156.25
1	21664	296.88
1	38213	515.63

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

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

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

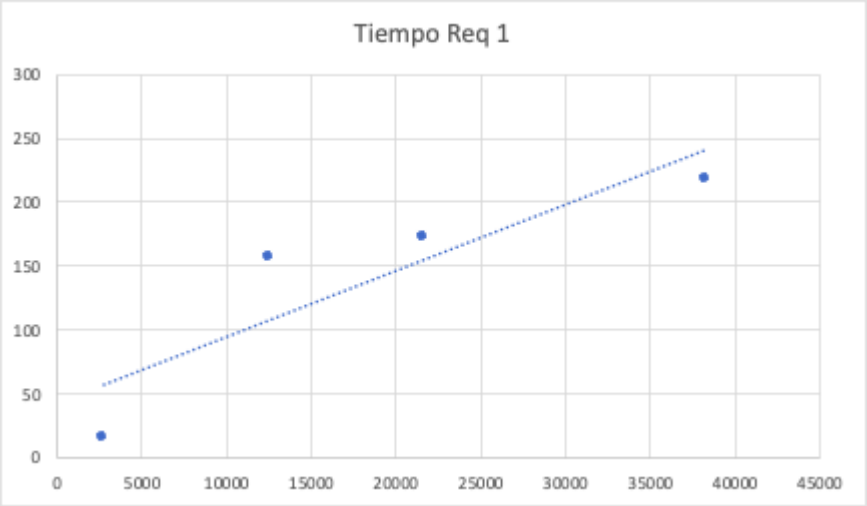
Máquina	*Tamaño de muestra*	*Tiempo promedio*
1	2716	15.63
1	12658	78.13
1	21664	156.25
1	38213	312.25

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

Máquina	*Tamaño de muestra*	*Tiempo promedio*
1	2716	15.63
1	12658	187.5
1	21664	421.88
1	38213	859.38

Gráficas de complejidad:

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.

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