Análisis de Reto 2 – Entrega Final

Integrantes:

- Juan David Vasquez Hernández - jd.vasquezh@uniandes.edu.co – 201914782

Complejidad de cada requerimiento:

Los códigos implementados para la solución de los requerimientos se muestran a continuación acompañados de sus respectivas complejidades.

En el caso de todos los algoritmos se usará n para representar el tamaño de artists y artworks asumiendo un valor similar en el caso del archivo –large.

Req1:

```
def ArtistsInRange(Artists,StartYear,EndYear,list_type,map_type):
    artistsInRange = map.newMap(maptype=map_type)
    posList = 0
    while posList < lt.size(Artists):
        Artist = lt.getElement(Artists,posList)
        Year = int(Artist['BeginDate'])
        if Year >= StartYear and Year <= EndYear:
            if map.contains(artistsInRange,Year):
                year_artists = map.get(artistsInRange,Year)['value']
                lt.addLast(year_artists,Artist)
            else:
                year_artists = lt.newList(datastructure=list_type)
                lt.addLast(year_artists,Artist)
                map.put(artistsInRange,Year,year_artists)
        posList += 1
    return artistsInRange


def SortChronologically(artistsInRange,StartYear,EndYear,list_type):
    sorted_artists = lt.newList(datastructure=list_type)
    for Year in range(StartYear,EndYear+1):
        if map.contains(artistsInRange,Year):
            year_artists = map.get(artistsInRange,Year)['value']
            for artist in lt.iterator(year_artists):
                lt.addLast(sorted_artists,artist)
    return sorted_artists
```

En este caso se optó por usar un único tipo de sort. El tipo de código aquí representado corresponde al mismo mecanismo de selection sort. A partir de lo anterior, haciendo el análisis para la complejidad en el reto 1, la segunda función posee una complejidad máxima

de $O(n^2)$. Por otro lado, la primera función recorre una única vez toda la lista de elementos, por ende, posee una complejidad O(n). Teniendo en cuenta lo anterior, la complejidad total de la solución es $O(n^2) + O(n)$, que aproximándola al mayor valor da $O(n^2)$. Esto sin embargo sería la complejidad en el caso de tener una lista como estructura de datos. Dado el uso de un TAD Map en el reto 2, la complejidad de búsqueda de cualquier elemento es O(1) u O(k), motivo por el cual la complejidad total de la solución es O(k*n) + O(n), que es posible aproximar a O(n).

Req 2:

```
def findArtist(artists,artist_IDs):
    artists_artworks = []
    for artist_ID in
(artist_IDs.replace('[','')).replace(']','').split(','):
        pos = 0
        while pos < lt.size(artists):
            artist = lt.getElement(artists,pos)
            if artist['ConstituentID'] == artist_ID:
                artists_artworks.append(artist['DisplayName'])
            pos += 1
    return artists_artworks

def ArtworksInRange(Artworks,StartDate,EndDate,list_type,map_type):
    artworksInRange = map.newMap(maptype=map_type)
    posList = 0
    while posList < lt.size(Artworks):
        Artwork = lt.getElement(Artworks,posList)
        Date = Artwork['DateAcquired']
        if Date >= StartDate and Date <= EndDate:
            if map.contains(artworksInRange,Date):
                date_artworks = map.get(artworksInRange,Date)['value']
                lt.addLast(date_artworks,Artwork)
            else:
                date_artworks = lt.newList(datastructure=list_type)
                lt.addLast(date_artworks,Artwork)
                map.put(artworksInRange,Date,date_artworks)
        posList += 1
    return artworksInRange

def SortArtworks(artworksInRange,sort_type,list_type):
    dates = map.keySet(artworksInRange)
    if sort_type == "QUICKSORT":
        sorted_dates = qs.sort(dates,cmpDates)
    elif sort_type == "INSERTION":
```

```
        sorted_dates = ins.sort(dates,cmpDates)
    elif sort_type == "SHELL":
        sorted_dates = ss.sort(dates,cmpDates)
    elif sort_type == "SELECTION":
        sorted_dates = scs.sort(dates,cmpDates)
    else:
        sorted_dates = ms.sort(dates,cmpDates)

    sorted_artworks = lt.newList(datastructure=list_type)
    for date in lt.iterator(sorted_dates):
        date_artworks = map.get(artworksInRange,date)['value']
        for artwork in lt.iterator(date_artworks):
            lt.addLast(sorted_artworks,artwork)
    return sorted_artworks
```

Para este requerimiento, haciendo el análisis a partir de la complejidad en el reto 1, la función usada para organizar las obras de arte por fecha de adquisición va a variar en complejidad dependiendo del tipo de sort usado. Si se tomara el peor caso, con insertion y selection tendríamos una complejidad $O(n^2)$. Con shell habría una complejidad $O(n^3/2)$. Con quicksort tendríamos complejidad $O(n^2)$ y con merge O(nlog(n)). Para la función findArtist – no mostrada – habría una complejidad de O(k*n), donde k es el número de IDs de artistas. La función ArtworksInRange también tendría una complejidad O(n). La complejidad máxima en el peor de los casos es de $O(n^3/2)$ + O(k*n) + O(n). Esta es estimable a la función de mayor complejidad, que vendría siendo $O(n^3/2)$. Es importante aclarar que esta máxima complejidad solo sucede al escoger la función shell sort como algoritmo de ordenamiento en el peor de los casos. Esto sin embargo sería la complejidad en el caso de tener una lista como estructura de datos. Dado el uso de un TAD Map en el reto 2, la complejidad de búsqueda de cualquier elemento es O(1) u O(k), motivo por el cual la complejidad máxima en el peor de los casos es de $O(n^2/2)$ + O(k) + O(n) – la misma complejidad de los algoritmos mencionados reduciendo la complejidad de aquellos donde se usa el TAD Map en 1/n.

Req 3:

```
def encounterArtist(artists,artist_name):
    for artist in lt.iterator(artists):
        if artist['DisplayName'] == artist_name:
            return artist['ConstituentID']
    return 'NotFound'

def artistMediumInfo(artworks,artist_ID,list_type,map_type):
    mediums = map.newMap(maptype=map_type)
    maxIteUses = 0
    artist_mediums = 0
```

```
    artist_artworks = 0
    for artwork in lt.iterator(artworks):
        if artist_ID in
(artwork['ConstituentID'].replace('[','')).replace(']','').split(','):
            artist_artworks += 1
            medium = artwork['Medium']
            if not (map.contains(mediums,medium)):
                artist_mediums += 1
                medium_artworks = lt.newList(datastructure=list_type)
                lt.addLast(medium_artworks,artwork)
                map.put(mediums,medium,medium_artworks)
                if maxIteUses == 0:
                    maxIteUses = 1
                    mostUsed = medium
            else:
                medium_artworks = map.get(mediums,medium)['value']
                lt.addLast(medium_artworks,artwork)
                IteUses = lt.size(medium_artworks)
                if IteUses > maxIteUses:
                    maxIteUses = IteUses
                    mostUsed = medium
    return artist_artworks, artist_mediums, mostUsed,
map.get(mediums,mostUsed)['value']
```

Para el 3er requerimiento, haciendo el análisis a partir de la complejidad en el reto 1, la función artistMediumInfo obtiene la información pertinente al recorrer todos los elementos de la lista una única vez. Su complejidad es O(n). Adicionalmente, la función encounterArtist tiene como máxima complejidad O(n). Por ende, la máxima complejidad de la solución es O(n) + O(n). Si se aproxima a la complejidad mayor, esta complejidad es de O(n). Aún con el uso del TAD Map en el reto 2, la complejidad permanece igual, pero se vuelve más eficiente la impresión de la respuesta gracias al uso de esta estructura de datos.

Req 4:

```
def findArtistNationality(artistsIDs_map,artist_IDs,list_type):
    artwork_nationalities = lt.newList(datastructure=list_type)
    for artist_ID in
(artist_IDs.replace('[','')).replace(']','').split(','):
        if map.contains(artistsIDs_map,artist_ID):
            artist = map.get(artistsIDs_map,artist_ID)['value']
            lt.addLast(artwork_nationalities,artist['Nationality'])
    return artwork_nationalities
```

```
def nationalityArtworks(artworks,catalog,list_type,map_type):
    artworksNationality = lt.newList(datastructure=list_type)
    nations = map.newMap(maptype=map_type)
    for artwork in lt.iterator(artworks):
        artists_ID = artwork['ConstituentID']
        artists_nationality =
findArtistNationality(catalog['ArtistID'],artists_ID,list_type)
        for nation in lt.iterator(artists_nationality):
            if nation == '':
                nation = 'Unknown'
            if not map.contains(nations,nation):
                nationality_artworks = lt.newList(datastructure=list_type)
                lt.addLast(nationality_artworks,artwork)
                map.put(nations,nation,nationality_artworks)
            else:
                nationality_artworks = map.get(nations,nation)['value']
                lt.addLast(nationality_artworks,artwork)

    nations_list = map.keySet(nations)
    for nation in lt.iterator(nations_list):
        num_artworks = lt.size(map.get(nations,nation)['value'])
        nationDict = {'Nation':nation,'NumbArtworks':num_artworks}
        lt.addLast(artworksNationality,nationDict)
    return artworksNationality, nations

def sortNations(artworksNationality,nations,sort_type):
    if sort_type == "QUICKSORT":
        sortedList = qs.sort(artworksNationality,cmpArtworkByNumbWorks)
    elif sort_type == "INSERTION":
        sortedList = ins.sort(artworksNationality,cmpArtworkByNumbWorks)
    elif sort_type == "SHELL":
        sortedList = ss.sort(artworksNationality,cmpArtworkByNumbWorks)
    elif sort_type == "SELECTION":
        sortedList = scs.sort(artworksNationality,cmpArtworkByNumbWorks)
    else:
        sortedList = ms.sort(artworksNationality,cmpArtworkByNumbWorks)
    art_nation = lt.getElement(sortedList,0)['Nation']
    artworks_nation = map.get(nations,art_nation)['value']
    return sortedList,art_nation,artworks_nation
```

Para este requerimiento, haciendo el análisis a partir de la complejidad en el reto 1, la función usada para organizar las nacionalidades por el número de obras va a variar en complejidad dependiendo del tipo de sort usado. Si se tomara el peor caso, con insertion y selection tendríamos una complejidad $O(n^2)$. Con shell habría una complejidad $O(n^3/2)$. Con quicksort

tendríamos complejidad $O(n^2)$ y con merge O(nlog(n)). Para la función findArtistNationality habría una complejidad de O(k*n), donde k es el número de IDs de artistas. Finalmente, la función nationalityArtworks posee una complejidad aproximada de O(n). La complejidad máxima en el peor de los casos es de $O(n^3/2) + O(k*n) + O(n)$. Esta es estimable a la función de mayor complejidad, que vendría siendo $O(n^3/2)$, así como en el caso del requerimiento 2.

Esto sin embargo sería la complejidad en el caso de tener una lista como estructura de datos. Dado el uso de un TAD Map en el reto 2, la complejidad de búsqueda de cualquier elemento es O(1) u O(k), motivo por el cual la complejidad máxima en el peor de los casos es de $O(n^2/2) + O(k) + O(n)$ – la misma complejidad de los algoritmos mencionados reduciendo la complejidad de aquellos donde se usa el TAD Map en 1/n. Estimando respecto a la función de mayor complejidad, este requerimiento posee un valor de $O(n^2/2)$.

Req 5:

```
def checkDeparment(artworks,department):
    encountered = False
    pos = 0
    while not encountered and pos < lt.size(artworks):
        artwork = lt.getElement(artworks,pos)
        if artwork['Department'] == department:
            encountered = True
        pos += 1
    return encountered

def moveDepartment(artworks,department,map_type):
    art2trans = 0
    est_price = 0
    est_weight = 0
    price_map = map.newMap(maptype=map_type)
    date_map = map.newMap(maptype=map_type)

    pos = 0
    while pos < lt.size(artworks):
        artwork = lt.getElement(artworks,pos)
        if artwork['Department'] == department:
            price = estimatePrice(artwork)
            if artwork['Weight (kg)'] != '':
                est_weight += float(artwork['Weight (kg)'])
            est_price += price
            art2trans += 1
            artwork['EstPrice'] = price
            map.put(price_map,str(round(price,2)),artwork)
            if artwork['Date'] != '':
```

```
                map.put(date_map,artwork['Date'],artwork)
        pos += 1
    return est_price, art2trans, est_weight, price_map, date_map

def SortArtworksByDate(date_map,sort_type,list_type):
    dates = map.keySet(date_map)
    if sort_type == "QUICKSORT":
        sortedList = qs.sort(dates,cmpDates)
    elif sort_type == "INSERTION":
        sortedList = ins.sort(dates,cmpDates)
    elif sort_type == "SHELL":
        sortedList = ss.sort(dates,cmpDates)
    elif sort_type == "SELECTION":
        sortedList = scs.sort(dates,cmpDates)
    else:
        sortedList = ms.sort(dates,cmpDates)
    ordered_dates = lt.newList(list_type)
    for date in lt.iterator(sortedList):
        lt.addLast(ordered_dates,map.get(date_map,date)['value'])
    return ordered_dates

def SortArtworksByPrice(price_map,sort_type,list_type):
    prices = map.keySet(price_map)
    if sort_type == "QUICKSORT":
        sortedList = qs.sort(prices,cmpPrices)
    elif sort_type == "INSERTION":
        sortedList = ins.sort(prices,cmpPrices)
    elif sort_type == "SHELL":
        sortedList = ss.sort(prices,cmpPrices)
    elif sort_type == "SELECTION":
        sortedList = scs.sort(prices,cmpPrices)
    else:
        sortedList = ms.sort(prices,cmpPrices)
    ordered_prices = lt.newList(list_type)
    for price in lt.iterator(sortedList):
        lt.addLast(ordered_prices,map.get(price_map,price)['value'])
    return ordered_prices
```

Con antelación, se presentan las funciones principales de la solución para el requerimiento 5. Las funciones usadas para organizar las obras de arte del departamento por año de creación y por costo van a variar en complejidad dependiendo del tipo de sort usado. Si se tomara el peor caso, con insertion y selection tendríamos una complejidad $O(n^2)$. Con shell habría una complejidad $O(n^3/2)$. Con quicksort tendríamos complejidad $O(n^2)$ y con merge $O(nlog(n))$. Las funciones moveDepartment y checkDepartment llevan a cabo sus respectivas labores con

una misma complejidad, aproximables a O(n). Teniendo en cuenta lo anterior, la complejidad máxima para el algoritmo vendría siendo (asumiendo el peor de los casos en todas las funciones $O(n^3/2) + O(n^3/2) + O(n) + O(n)$. Si se aproximaran estas complejidades a la de aquella función que más complejidad posee, se podría aproximar la complejidad máxima de la solución a $O(n^3/2)$.

Esto sin embargo sería la complejidad en el caso de tener una lista como estructura de datos. Dado el uso de un TAD Map en el reto 2, la complejidad de búsqueda de cualquier elemento es O(1) u O(k), motivo por el cual la complejidad máxima en el peor de los casos es de $O(n^2/2) + O(k) + O(n) + O(n)$ – la misma complejidad de los algoritmos mencionados reduciendo la complejidad de aquellos donde se usa el TAD Map en 1/n. Estimando respecto a la función de mayor complejidad, este requerimiento posee un valor de $O(n^2/2)$.

Pruebas de Rendimiento:

Req 1:

Para la prueba se usaron valores de 1920 y 1950.

Requerimiento 1		
Lista/Map	Separate Chaining	Linear Probing
Tiempo (mseg)	171,875	156,250

Req 2:

Para la prueba se ordenaron las obras entre las fechas de 1970-01-01 y 1975-01-01 (usando merge sort).

Requerimiento 2		
Lista/Map	Separate Chaining	Linear Probing
Tiempo (mseg)	828,125	609,375

Req 3:

Para la prueba se usó la artista Louise Bourgeois.

Requerimiento 3		
Lista/Map	Separate Chaining	Linear Probing
Tiempo (mseg)	359,375	296,875

Req 4:

Requerimiento 4		
Lista/Map	Separate Chaining	Linear Probing
Tiempo (mseg)	7.468,750	7.734,375

Req 5:

Para la prueba se usó el departamento de Drawings & Prints.

Requerimiento 5		
Lista/Map	Separate Chaining	Linear Probing
Tiempo (mseg)	34.546,875	34.325,475