

## Análisis de Reto 1 – Entrega Final

Integrantes:

- Juan David Vasquez Hernández - jd.vasquezh@uniandes.edu.co - 201914782
- Santiago Sierra Rodríguez - s.sierrar@uniandes.edu.co – 201911116

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):
    artistsInRange = lt.newList(datastructure=list_type)
    posList = 0
    while posList < lt.size(Artists):
        Artist = lt.getElement(Artists, posList)
        Year = int(Artist['BeginDate'])
        if Year >= StartYear and Year <= EndYear:
            lt.addLast(artistsInRange, Artist)
        posList += 1
    return artistsInRange

def SortChronologically(artistsInRange):
    for pos1 in range(lt.size(artistsInRange)):
        minPos = pos1
        for pos2 in range(pos1+1, lt.size(artistsInRange)):
            YearMin = lt.getElement(artistsInRange, minPos)['BeginDate']
            Year2 = lt.getElement(artistsInRange, pos2)['BeginDate']
            if Year2 < YearMin:
                minPos = pos2

        lt.exchange(artistsInRange, minPos, pos1)
    sortedArtists = artistsInRange
    return sortedArtists
```

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, 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  $O(n^2)$ .

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,StartYear,EndYear,list_type):
    artworksInRange = lt.newList(datastructure=list_type)
    posList = 0
    while posList < lt.size(Artworks):
        Artwork = lt.getElement(Artworks,posList)
        Year = Artwork['DateAcquired']
        if Year >= StartYear and Year <= EndYear:
            lt.addLast(artworksInRange,Artwork)
        posList += 1
    return artworksInRange

def SortArtworks(artworksInRange,sort_type):
    if sort_type == "QUICKSORT":
        sortedList = qs.sort(artworksInRange,cmpArtworkByDateAcquired)
    elif sort_type == "INSERTION":
        sortedList = ins.sort(artworksInRange,cmpArtworkByDateAcquired)
    elif sort_type == "SHELL":
        sortedList = ss.sort(artworksInRange,cmpArtworkByDateAcquired)
    elif sort_type == "SELECTION":
        sortedList = scs.sort(artworksInRange,cmpArtworkByDateAcquired)
    else:
        sortedList = ms.sort(artworksInRange,cmpArtworkByDateAcquired)
    return sortedList
```

Para este requerimiento, 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  $O(n\log(n))$ . Para la función findArtist 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.

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):
    mediums = {}
    maxIteUses = 0
    artist_mediums = 0
    artist_artworks = 0
    for artwork in lt.iterator(artworks):
        if artwork['ConstituentID'] == '[' + artist_ID + ']':
            artist_artworks += 1
            medium = artwork['Medium']
            if medium not in mediums:
                artist_mediums += 1
                mediums[medium] = lt.newList(datastructure=list_type)
                lt.addLast(mediums[medium],artwork)
                if maxIteUses == 0:
                    maxIteUses = 1
                    mostUsed = medium
            else:
                lt.addLast(mediums[medium],artwork)
                IteUses = lt.size(mediums[medium])
                if IteUses > maxIteUses:
                    maxIteUses = IteUses
                    mostUsed = medium
    return artist_artworks, artist_mediums, mostUsed, mediums[mostUsed]
```

Para el 3er requerimiento, 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  $O(n)$ .

Req 4:

```
def findArtistNationality(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['Nationality'])
            pos += 1
    return artists_artworks

def nationalityArtworks(artworks,artists,list_type):
    artworksNationality = lt.newList(datastructure=list_type)
    nations = {}
    for artwork in lt.iterator(artworks):
        artists_ID = artwork['ConstituentID']
        artists_nationality = findArtistNationality(artists,artists_ID)
        nations_ite = []
        for nation in artists_nationality:
            if nation == '':
                nation = 'Unknown'
            if nation not in nations_ite:
                nations_ite.append(nation)
            if nation not in nations:
                nations[nation] = lt.newList(datastructure=list_type)
                lt.addLast(nations[nation], artwork)
            else:
                lt.addLast(nations[nation], artwork)

    for nation in nations:
        num_artworks = lt.size(nations[nation])
        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 = nations[art_nation]
    return sortedList,art_nation,artworks_nation

```

Para este requerimiento, 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  $O(n\log(n))$ . 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.

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,list_type):
    art2trans = 0
    est_price = 0
    est_weight = 0
    artworks_dep = lt.newList(datastructure=list_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
            lt.addLast(artworks_dep,artwork)
        pos += 1
    return est_price, art2trans, est_weight, artworks_dep

def artworksWithDate(artworks_dep,list_type):
    artworksWithDate = lt.newList(datastructure=list_type)
    for artwork in lt.iterator(artworks_dep):
        if artwork['Date'] != '':
            lt.addLast(artworksWithDate,artwork)
    return artworksWithDate

def SortArtworksByDate(artworks_dep,sort_type):
    if sort_type == "QUICKSORT":
        sortedList = qs.sort(artworks_dep,cmpArtworkByDate)
    elif sort_type == "INSERTION":
        sortedList = ins.sort(artworks_dep,cmpArtworkByDate)
    elif sort_type == "SHELL":
        sortedList = ss.sort(artworks_dep,cmpArtworkByDate)
    elif sort_type == "SELECTION":
        sortedList = scs.sort(artworks_dep,cmpArtworkByDate)
    else:
        sortedList = ms.sort(artworks_dep,cmpArtworkByDate)
    return sortedList

def SortArtworksByPrice(artworks_dep,sort_type):
    if sort_type == "QUICKSORT":
        sortedList = qs.sort(artworks_dep,cmpArtworkByEstPrice)
    elif sort_type == "INSERTION":
        sortedList = ins.sort(artworks_dep,cmpArtworkByEstPrice)
    elif sort_type == "SHELL":
        sortedList = ss.sort(artworks_dep,cmpArtworkByEstPrice)
    elif sort_type == "SELECTION":
        sortedList = scs.sort(artworks_dep,cmpArtworkByEstPrice)
    else:
        sortedList = ms.sort(artworks_dep,cmpArtworkByEstPrice)
    return sortedList

```

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  $O(n \log(n))$ . Las funciones artworksWithDate, 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  $O(n^3/2) + O(n^3/2) + O(n) + O(n) + O(n)$ ). Si se aproximaran estas complejidades a la de aquella función que más complejidad posee, se podría aproximar la complejidad máxima de la solución a  $O(n^3/2)$ .

Pruebas de Rendimiento:

Req 1:

Para la prueba se hallaron los artistas entre los años 1900 y 1950, variando en cada prueba el tamaño de la muestra.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Tiempo [ms]
5,00%	500,00	1031,25
10,00%	1000,00	5843,75
20,00%	2000,00	54984,38
35,00%	3500,00	252265,63
50,00%	5000,00	783078,13

Porcentaje de la muestra [pct]	Tamaño de la muestra (LINKED_LIST)	Tiempo [ms]
5,00%	500,00	687,50
10,00%	1000,00	6500,00
20,00%	2000,00	50468,75
35,00%	3500,00	252468,75
50,00%	5000,00	724375,00





Como se observa en las figuras, la complejidad temporal del algoritmo es tal como se esperaba a partir del análisis de complejidad. Al usarse el método de ordenamiento selection, el problema posee una complejidad de  $O(n^2)$ .

Req 2:

Para la prueba se ordenaron las obras entre las fechas de 1970-01-01 y 1970-01-15, variando en cada prueba el tamaño de la muestra.