Análisis de Complejidad

Nombre: Nicolas Merchan Cuestas

Código: 202112109

Correo: n.merchan@uniandes.edu.co

Notas:

- Los resultados de las pruebas de tiempos de ejecución y las correspondientes gráficas de los mismos se encuentran en '*Datos - Análisis de Complejidad.xslx*'.
- El programa 'information.py' proporciona información sobre el número de artistas, obras de arte, años de nacimiento, años de adquisición, nacionalidades y departamentos del museo. La información mencionada anteriormente es esencial para la correcta implementación de los TAD maps en el programa principal.
- El programa 'Test_Function.py' realiza las pruebas de tiempos de ejecución de manera automática e ingresa los resultados en un archivo EXCEL llamado 'Test_Data.xlsx'.

Requerimiento 1

- **Reto 1**

```
400 def SortArtistByBirthYear(sub_list, sorting_method, initial_year_birth, end_year_birth, data_structure):
401     start_time = time.process_time()
402 
403     sub_list = sub_list.copy()
404     artists_birth_year_range_list = FilteringArtistsByBirthYear(sub_list, initial_year_birth,
405                                                                  end_year_birth, data_structure)
406     sorted_list = SortingMethodExecution(sorting_method, artists_birth_year_range_list, cmpArtistByBirthDate)
407 
408     stop_time = time.process_time()
409     elapsed_time_mseg = elapsed_time_mseg = (stop_time - start_time)*1000
410 
411     return elapsed_time_mseg, sorted_list
```

La complejidad asociada al requerimiento 1 del Reto 1 está dada principalmente por el algoritmo de ordenamiento utilizado. El requerimiento 1 se lleva a cabo por medio de la función **sortArtistByBirthYear()**. En esta función se hace uso de la función **FilteringArtistsByBirthYear()** para obtener un TAD lista con los artistas nacidos dentro del rango de años dado, por medio de una búsqueda lineal sobre el TAD lista de los artistas. De esa forma, la complejidad asociada a **FilteringArtistsByBirthYear()** es **O(n)**. Posteriormente, en la función **SortingMethodExecution()** se procede a ordenar el TAD lista de los artistas nacidos dentro del rango de años dado por medio del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 1 es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 1 es, en el mejor caso, **O(n) + O(n)**, **O(nlog(n)) + O(n)**, **O(nlog(n)) + O(n)** y **O(nlog(n)) + O(n)** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

- **Reto 2**

```
251 def getArtistsByBirthYear(catalog, data_structure, initial_birth_year, end_birth_year):
252     artists_birth_years_interval = lt.newList(data_structure)
253     birth_years_map = catalog['birth_years']
254     for year in range(initial_birth_year, end_birth_year + 1):
255         year = str(year)
256         if mp.contains(birth_years_map, year):
257             artists_birth_year = me.getValue(mp.get(birth_years_map, year))
258             for artist in lt.iterator(artists_birth_year):
259                 lt.addLast(artists_birth_years_interval, artist)
260     return artists_birth_years_interval
```

La complejidad asociada al requerimiento 1 del Reto 2 está dada por **O(1)**. El requerimiento 1 se lleva a cabo por medio de la función **getArtistsByBirthYear()**. En esta función se recurre al uso del TAD map **catalog['birth_years']**, el cual fue creado en el cargue de datos. Las llaves de **catalog['birth_years']** son los años de nacimiento de los artistas cargados y los valores asociados a dichas llaves son las TAD listas de los artistas nacidos en el año de la llave. De ese modo, para encontrar los artistas nacidos en un rango de años dado, es preciso consultar **catalog['birth_years']** para todos los años dentro del intervalo de años. Así, la complejidad del requerimiento 1 se reduce a consultar llaves-valor en el TAD map **catalog['birth_years']** para una cantidad de llaves (años de nacimiento) siempre menor a 236*.

La complejidad del requerimiento 1 es más favorable en el Reto 2 que en el Reto 1, porque la complejidad del requerimiento en cuestión en el Reto 2 es **O(1)** a comparación de aquella del Reto 1 superior a **O(n)**.

Requerimiento 2

- **Reto 1**

```
428 def SortArtworksAdquisitionRange(sub_list, sorting_method, initial_date_adquisition,
429                                                                 end_date_adquisition, data_structure):
430     start_time = time.process_time()
431     sub_list = sub_list.copy()
432
433     artworks_adquisition_date_range_list = FilteringArtworksByAdquisitionDate(sub_list, initial_date_adquisition,
434                                                                 end_date_adquisition, data_structure)
435     purchase_artworks_num = lt.size(FilteringArtworksByAdquisitionDateAndCreditLine(artworks_adquisition_date_range_list,
436                                                                 data_structure))
437     sorted_list_by_date = SortingMethodExecution(sorting_method, artworks_adquisition_date_range_list,
438                                                                 cmpArtworkByDateAcquired)
439
440     stop_time = time.process_time()
441     elapsed_time_mseg = elapsed_time_mseg = (stop_time - start_time)*1000
442
443     return elapsed_time_mseg, sorted_list_by_date, purchase_artworks_num
```

La complejidad asociada al requerimiento 2 del Reto 1 está dada principalmente por el algoritmo de ordenamiento utilizado. El requerimiento 2 se lleva a cabo por medio de la función **sortArtworksAdquisitionRange()**. En esta función se hace uso de la función **FilteringArtworksByAdquisitionDate()** para obtener un TAD lista de las obras adquiridas dentro del rango de fechas dado, por medio de una búsqueda lineal sobre el TAD lista de las obras. De esa forma, la complejidad asociada a **FilteringArtistsByBirthYear()** es **O(n)**.

Posteriormente, en la función **SortingMethodExecution()** se procede a ordenar el TAD lista de las obras adquiridas dentro del rango de fechas dado por medio del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 2 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 2 es, en el mejor caso, **O(n) + O(n)**, **O(nlog(n)) + O(n)**, **O(nlog(n)) + O(n)** y **O(nlog(n)) + O(n)** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

- **Reto 2**

```
264  def getArtworksByAdquisitonDate(catalog, data_structure, sorting_method,
265                                                  initial_adquisiton_date, end_adquisition_date):
266      date_adquired_artworks_list = lt.newList(data_structure)
267
268      adquisition_years_map = catalog['adquisition_years']
269      initial_adquisition_year = getAdquisitionYear(initial_adquisiton_date)
270      initial_adquisiton_date_in_days = TransformationDateToDays(initial_adquisiton_date)
271      end_adquisition_year = getAdquisitionYear(end_adquisition_date)
272      end_adquisition_date_in_days = TransformationDateToDays(end_adquisition_date)
273
274      if mp.contains(adquisition_years_map, initial_adquisition_year):
275          first_year_artworks = me.getValue(mp.get(adquisition_years_map, initial_adquisition_year))
276          for artwork in lt.iterator(first_year_artworks):
277              date = TransformationDateToDays(artwork['DateAcquired'])
278              if date >= initial_adquisiton_date_in_days:
279                  lt.addLast(date_adquired_artworks_list, artwork)
280
281      for year in range(initial_adquisition_year + 1, end_adquisition_year):
282          if mp.contains(adquisition_years_map, year):
283              year_artworks = me.getValue(mp.get(adquisition_years_map, year))
284              for artwork in lt.iterator(year_artworks):
285                  lt.addLast(date_adquired_artworks_list, artwork)
286
287      if mp.contains(adquisition_years_map, end_adquisition_year):
288          last_year_interval_artworks = me.getValue(mp.get(adquisition_years_map, end_adquisition_year))
289          for artwork in lt.iterator(last_year_interval_artworks):
290              date = TransformationDateToDays(artwork['DateAcquired'])
291              if date <= end_adquisition_date_in_days:
292                  lt.addLast(date_adquired_artworks_list, artwork)
293
294      SortingMethodExecution(sorting_method, date_adquired_artworks_list, cmpArtworksByDateAcquired)
```

La complejidad asociada al requerimiento 2 del Reto 2 está dada por el algoritmo de ordenamiento utilizado. El requerimiento 2 se lleva a cabo por medio de la función **getArtworksByAdquisitionDate()**. En esta función se recurre al uso del TAD map **catalog['adquisition_years']**, el cual fue creado en el cargue de datos. Las llaves de **catalog['adquisition_years']** son los años de adquisición de las obras cargadas y los valores asociados a dichas llaves son las TAD listas de las obras adquiridas en el año de la llave. De ese modo, para encontrar las obras adquiridas dentro de un rango de rango de fechas dado, es preciso consultar **catalog['adquisition_years']** para todos los años dentro del intervalo de años y separar las obras presentes en los años de los extremos del rango según las fechas de los extremos del rango. Así, la complejidad de la búsqueda de obras en adquiridas en el rango de fechas dado se reduce a consultar llaves-valor en el TAD map **catalog['adquisition_years']** para una cantidad de llaves (años de adquisición) siempre menor a 93*. Sin embargo, la función **SortingMethodExecution()** tiene asociada una

complejidad en función del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 2 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 2 es, en el mejor caso, **O(n)**, **O(nlog(n))**, **O(nlog(n))** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

La complejidad del requerimiento 2 es más favorable en el Reto 2 que en el Reto 1, porque la complejidad de búsqueda de obras adquiridas dentro del rango de fechas indicado en el Reto 2 es **O(1)** a comparación de aquella del Reto 1 igual a **O(n)**.

Requerimiento 3

- **Reto 1**

```
447 def ClasifyArtistsTechnique(sub_list, lst, sorting_method, artist_name, data_structure):
448     start_time = time.process_time()
449     sub_list = sub_list.copy()
450 
451     information = CreationArtistTechniquesInformation(sub_list, lst, artist_name, data_structure)
452     artist_artworks = information[0]
453     artist_techniques = information[1]
454     sorted_artist_techniques = SortingMethodExecution(sorting_method, artist_techniques, cmpTechniquesBySize)
455 
456     stop_time = time.process_time()
457     elapsed_time_mseg = (stop_time - start_time)*1000
458 
459     return elapsed_time_mseg, artist_artworks, sorted_artist_techniques
```

La complejidad asociada al requerimiento 3 del Reto 1 está dada principalmente por el algoritmo de ordenamiento utilizado. El requerimiento 3 se lleva a cabo por medio de la función **ClasifyArtistsTechnique()**. En esta función se hace uso de la función **CreationArtistTechniquesInformation()** para obtener un TAD lista de las obras del artista dado y un diccionario donde las llaves son las técnicas y el valor asociado a dicha llaves es un TAD lista de la obras creadas en la técnica de la llave. La ejecución de **CreationArtistTechniquesInformation()** se da por medio de una búsqueda lineal sobre el TAD lista de las obras y la complejidad asociada a este proceso es **O(n)**. Posteriormente, se procede a convertir el diccionario de técnicas en un TAD lista y en la función **SortingMethodExecution()** se ordena el TAD lista de las técnicas utilizadas por el artista, por medio del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 3 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 3 es, en el mejor caso, **O(n) + O(n)**, **O(nlog(n)) + O(n)**, **O(nlog(n)) + O(n)** y **O(nlog(n)) + O(n)** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

- **Reto 2**

```
300   def getArtworksByMediumAndArtist(catalog, artist_name):
301       artist_names_map = catalog['artists_names']
302       num_more_artworks = 0
303       num_total_artworks = 0
304       num_total_mediums = 0
305       list_more_artworks = lt.newList()
306       name_more_artworks = ''
307       if mp.contains(artist_names_map, artist_name):
308           artist_Id = me.getValue(mp.get(artist_names_map, artist_name))
309           mediums_keys_list = me.getValue(mp.get(catalog['artists_Ids'], artist_Id))['mediums_keys']
310           num_total_mediums = lt.size(mediums_keys_list)
311           mediums_map = me.getValue(mp.get(catalog['artists_Ids'], artist_Id))['mediums']
312           for medium_name in lt.iterator(mediums_keys_list):
313               medium_artworks = me.getValue(mp.get(mediums_map, medium_name))
314               num_medium = lt.size(medium_artworks)
315               num_total_artworks += num_medium
316               if num_medium > num_more_artworks:
317                   num_more_artworks = num_medium
318                   name_more_artworks = medium_name
319                   list_more_artworks = medium_artworks
320       return list_more_artworks, num_total_artworks, num_total_mediums, name_more_artworks
```

La complejidad asociada al requerimiento 3 del Reto 2 está dada por **O(1)**. El requerimiento 3 se lleva a cabo por medio de la función **getArtworksByMediumAndArtist()**. En esta función se recurre al uso de los TAD map **catalog[‘artists_names’]** y **catalog[‘artists_Ids’]**, los cuales fueron creados en el cargue de datos. Las llaves de **catalog[‘artists_names’]** son los nombres de los artistas y los valores asociados a dichas llaves son los códigos únicos de identificación de cada artista. Luego, las llaves de **catalog[‘artists_Ids’]** son los códigos únicos de identificación de cada artista y los valores asociados a dichas llaves son diccionarios que contienen la información de los artistas y las obras ordenadas por técnica del artista en un TAD map. De ese modo, para encontrar el número de obras creadas por el artista, la técnica más utilizada por el artista y las obras de dicha técnica, es preciso encontrar dicha información de manera directa comparando la información presente en **catalog[‘artists_Ids’]** asociada al artista dado. Así, la complejidad del requerimiento 1 se reduce a consultar llaves-valor en los TAD maps **catalog[‘artists_names’]** y **catalog[‘artists_Ids’]** para un artista dado.

La complejidad del requerimiento 3 es más favorable en el Reto 2 que en el Reto 1, porque la complejidad del requerimiento en cuestión en el Reto 2 es **O(1)** a comparación de aquella del Reto 1 superior a **O(n)**.

Requerimiento 4

- **Reto 1**

```
463 def ClasifyArtworksByNationality(sub_list, sorting_method, artists_ID_dict, data_structure):
464     start_time = time.process_time()
465     sub_list = sub_list.copy()
466 
467     num_artworks_nationalities = CreateDictNumPerNationality(sub_list, artists_ID_dict)
468     artworks_nationalities_list = CreateNationalityNumList(num_artworks_nationalities, data_structure)
469     sorted_list = SortingMethodExecution(sorting_method, artworks_nationalities_list, cmpNationalitiesBySize)
470 
471     stop_time = time.process_time()
472     elapsed_time_mseg = (stop_time - start_time)*1000
473 
474     return elapsed_time_mseg, sorted_list
```

La complejidad asociada al requerimiento 4 del Reto 1 está dada principalmente por el algoritmo de ordenamiento utilizado. El requerimiento 4 se lleva a cabo por medio de la función **ClasifyArtistsByNationality()**. En esta función se hace uso de las funciones **CreateDictNumPerNationality()** y **CreateNationalityNumList()** para obtener un diccionario con las nacionalidades como llaves y el número de obras asociadas como valor y un TAD lista con la información del diccionario, por medio de una búsqueda lineal sobre el TAD lista de las obras. De esa forma, la complejidad asociada a **CreateDictNumPerNationality()** y **CreateNationalityNumList()** es **O(n)**. Posteriormente, en la función **SortingMethodExecution()** se procede a ordena el TAD lista de las nacionalidades, por medio del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 4 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 4 es, en el mejor caso, **O(n) + O(n)**, **O(nlog(n)) + O(n)**, **O(nlog(n)) + O(n)** y **O(nlog(n)) + O(n)** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

- **Reto 2**

```
324 def getNationalitiesByNumArtworks(catalog, data_structure, sorting_method):
325     nationalities_names_list = catalog['nationalities_keys']
326     num_nationalities_list = lt.newList(data_structure)
327     major_nationality_artworks_list = lt.newList()
328     if lt.size(nationalities_names_list) >= 1:
329         for nationality in lt.iterator(nationalities_names_list):
330             num_artworks = lt.size(me.getValue(mp.get(catalog['nationalities'], nationality)))
331             lt.addLast(num_nationalities_list, (nationality, num_artworks))
332         SortingMethodExecution(sorting_method, num_nationalities_list, cmpNationalityByNumArtworks)
333         major_nationality_name = lt.getElement(num_nationalities_list, 1)[0]
334         major_nationality_artworks_list = me.getValue(mp.get(catalog['nationalities'], major_nationality_name))
335     return major_nationality_artworks_list, num_nationalities_list
```

La complejidad asociada al requerimiento 4 del Reto 2 está dada por **O(1)**. El requerimiento 4 se lleva a cabo por medio de la función **getNationalitiesByNumArtworks()**. En esta función se recurre al uso del TAD map **catalog['nationalities']**, el cual fue creado en el cargue de datos. Las llaves de **catalog['nationalities']** son las nacionalidades de los artistas y los valores asociados a dichas llaves son los TAD listas de las obras asociadas a dicha nacionalidad. De ese modo, para encontrar las nacionalidades con mayor número de obras, es preciso comparar los

elementos del TAD lista **catalog['nationalities_keys']**, el cual contiene los nombres de las nacionalidades como elementos. Así, la complejidad del requerimiento 4 se reduce a consultar llaves-valor en el TAD map **catalog['nationalities']** y organizarlas por medio de **SortingMethodExecution()** para una cantidad de llaves (nacionalidades) siempre menor a 119*.

La complejidad del requerimiento 4 es más favorable en el Reto 2 que en el Reto 1, porque la complejidad del requerimiento en cuestión en el Reto 2 es **O(1)** a comparación de aquella del Reto 1 superior a **O(n)**.

Requerimiento 5

- **Reto 1**

```
478 def TransportArtworksDepartment(sub_list, sorting_method, department, data_structure):
479     start_time = time.process_time()
480     information = CreateArtworkTransportationCostList(sub_list, department, data_structure)
481     artworks_by_date = information[0]
482     artworks_by_cost = information[1]
483     total_cost = information[2]
484     total_weight = information[3]
485     oldest_artworks = CreationOrderedListByDate(artworks_by_date, sorting_method)
486     most_expensive_artworks = CreationOrderedListByCost(artworks_by_cost, sorting_method)
487 
488     stop_time = time.process_time()
489     elapsed_time_mseg = (stop_time - start_time)*1000
490 
491     return elapsed_time_mseg, artworks_by_date, total_cost, total_weight, most_expensive_artworks, oldest_artworks
```

La complejidad asociada al requerimiento 5 del Reto 1 está dada principalmente por el algoritmo de ordenamiento utilizado. El requerimiento 5 se lleva a cabo por medio de la función **TransportArtworkDepartment()**. En esta función se hace uso de la función **CreationArtworkTransportationCostList()** para obtener un TAD lista de las obras del departamento dado. La ejecución de **CreationArtworkTransportationCostList()** se da por medio de una búsqueda lineal sobre el TAD lista de las obras y la complejidad asociada a este proceso es **O(n)**. Posteriormente, se procede a ordenar el TAD lista de las obras del departamento en la función del costo y el año de creación. Aquel proceso se lleva a cabo por medio de las funciones **CreationOrderedListByDate() y CreationOrderedListByCost()** haciendo uso del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 5 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 5 es, en el mejor caso, **O(n) + O(n)**, **O(nlog(n)) + O(n)**, **O(nlog(n)) + O(n)** y **O(nlog(n)) + O(n)** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

- **Reto 2**

```
339  def getTransportationCostByDepartment(catalog, data_structure, sorting_method, department):
340      departments_map = catalog['departments']
341      requirement_list_by_date = lt.newList()
342      requirement_list_by_price = lt.newList()
343      total_cost = 0
344      total_weight = 0
345      if mp.contains(departments_map, department):
346          department_artworks_list = me.getValue(mp.get(departments_map, department))
347          requirement_info = CreateArtworkTransportationCostList(department_artworks_list, data_structure)
348          requirement_list_by_date = requirement_info[0]
349          requirement_list_by_price = requirement_info[1]
350          total_cost = requirement_info[2]
351          total_weight = requirement_info[3]
352
353          SortingMethodExecution(sorting_method, requirement_list_by_price, cmpArtworkBycost )
354          SortingMethodExecution(sorting_method, requirement_list_by_date, cmpArtworkByCreationDate)
355
356      return requirement_list_by_date, requirement_list_by_price, total_cost, total_weight
```

La complejidad asociada al requerimiento 5 del Reto 2 está dada por el algoritmo de ordenamiento utilizado. El requerimiento 5 se lleva a cabo por medio de la función **getTransportationCostByDepartment()**. En esta función se recurre al uso del TAD map **catalog[‘departments]**, el cual fue creado en el cargue de datos. Las llaves de **catalog[‘departments]** son los departamentos de las obras y los valores asociados a dichas llaves son las TAD listas de las obras del departamento de la llave. Así, la complejidad de la búsqueda de obras de un departamento dado se reduce a consultar llaves-valor en el TAD map **catalog[‘departments’]** para una cantidad de llaves (departamentos) siempre menor a 8*. Luego, se procede a utilizar la función **SortingMethodExecution()** para ordenar el TAD lista de las obras del departamento en función del costo y el año de creación. Sin embargo, la función **SortingMethodExecution()** tiene asociada una complejidad en función del algoritmo de ordenamiento ingresado por el usuario. En pocas palabras, la complejidad del requerimiento 5 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 5 es, en el mejor caso, **O(n)**, **O(nlog(n))**, **O(nlog(n))** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

La complejidad del requerimiento 5 es más favorable en el Reto 2 que en el Reto 1, porque la complejidad de búsqueda de obras en el rango de fechas indicado en el Reto 2 es **O(1)** a comparación de aquella del Reto 1 igual a **O(n)**.

Requerimiento 6

```
def getMostProlificArtists(catalog, data_structure, sorting_method,
                                                initial_birth_year, end_birth_year, num_artists):
    most_prolific_artists_list = lt.newList(data_structure)
    birth_years_map = catalog['birth_years']
    for year in range(initial_birth_year, end_birth_year + 1):
        year = str(year)
        if mp.contains(birth_years_map, year):
            artists_birth_year = me.getValue(mp.get(birth_years_map, year))
            for artist in lt.iterator(artists_birth_year):
                artists_Id = artist['ConstituentID']
                artist_dict = me.getValue(mp.get(catalog['artists_Ids'], artists_Id))
                artist_name = artist_dict['info']['DisplayName']
                artist_medium_info = getArtworksByMediumAndArtist(catalog, artist_name)
                artworks_most_used_medium = artist_medium_info[0]
                if lt.size(artworks_most_used_medium) >= 5:
                    artworks_most_used_medium = lt.subList(artist_medium_info[0], 1, 5)
                num_more_artworks = lt.size(artworks_most_used_medium)
                num_total_artworks = artist_medium_info[1]
                num_total_mediums = artist_medium_info[2]
                name_most_used_medium = artist_medium_info[3]
                lt.addLast(most_prolific_artists_list, (artist_name, artworks_most_used_medium,
                num_total_artworks, num_total_mediums, num_more_artworks, name_most_used_medium))
    SortingMethodExecution(sorting_method, most_prolific_artists_list, cmpMostProlificArtist)
    if lt.size(most_prolific_artists_list) > num_artists:
        requirement_list = lt.subList(most_prolific_artists_list, 1, num_artists)
    else:
        requirement_list = most_prolific_artists_list
    return requirement_list
```

La complejidad asociada al requerimiento 6 del Reto 2 está dada por el algoritmo de ordenamiento utilizado. El requerimiento 6 se lleva a cabo por medio de la función **getMostProlificArtists()**. En esta función se recurre al uso del TAD map **catalog['birth_years']** para encontrar los artistas nacidos dentro del rango de años dado con una complejidad asociada de **O(1)**. Posteriormente, se procede a crear un TAD lista por medio de **getArtworksByMediumAndArtists()** en el cual los elementos contienen la información referente al nombre del artista, el número de obras creadas, la cantidad de técnicas utilizadas y el TAD lista de las obras de la técnica más utilizada por el artista. La complejidad de asociada al proceso descrito anteriormente es **O(n)**. Finalmente, se procede a ordenar el TAD lista de los artistas nacidos dentro del rango de años dado en base a los criterios descritos en el enunciado del Reto 2 y por medio de **SortingMethodExecution()**. En pocas palabras, la complejidad del requerimiento 6 es, en el peor caso, **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Similarmente, la complejidad del requerimiento 6 es, en el mejor caso **O(n) + O(n)**, **O(nlog(n)) + O(n)**, **O(nlog(n)) + O(n)** y **O(nlog(n)) + O(n)** para los algoritmos de ordenamiento Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.

Notas:

*La información fue obtenida por medio del programa 'information.py'.