**Análisis de Complejidad**

**Nombre:** Nicolas Merchan Cuestas

**Código:** 202112109

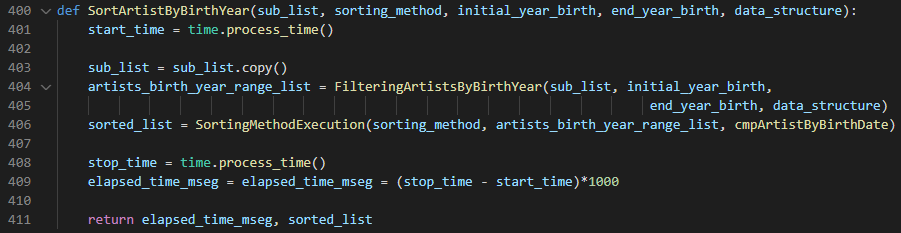
**Correo:** [n.merchan@uniandes.edu.co](mailto:n.merchan@uniandes.edu.co)

**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’.

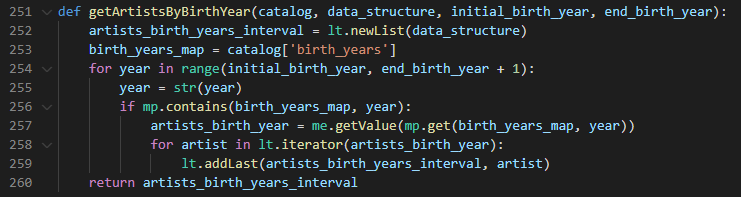
**Requerimiento 1**

* **Reto 1**



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**

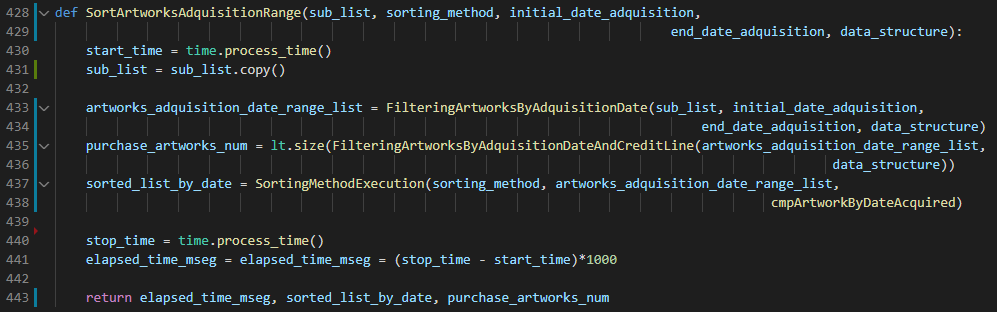


La complejidad asociada al requerimiento 1 del Reto 2 está dada por **O(1)**. El requerimiento 1 se lleva a cabo por medio de la función **getArtistsByBirthYear()**. En esta función se recurre al uso del TAD map **catalog[‘birth\_years’]**, el cual fue creado en el cargue de datos. Las llaves de **catalog[‘birth\_years’]** 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)**.

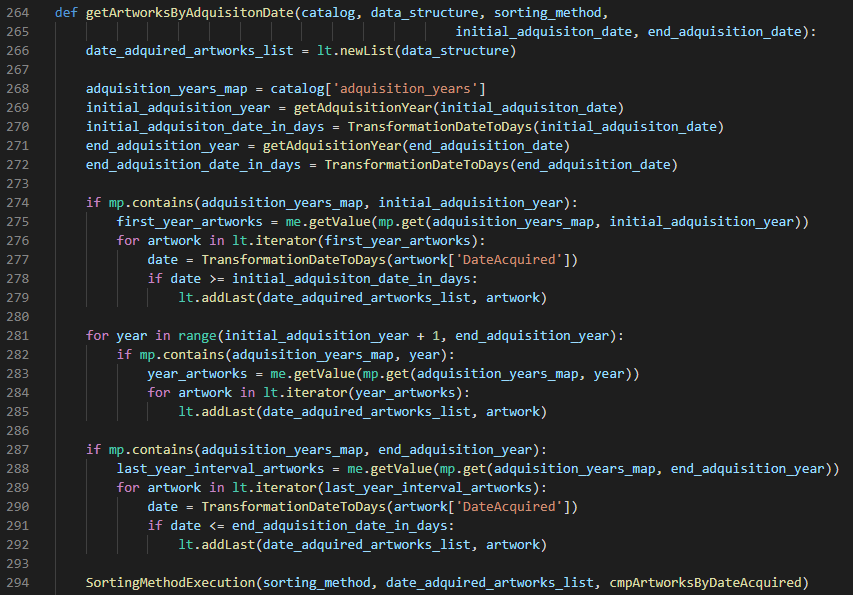
**Requerimiento 2**

* **Reto 1**



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**

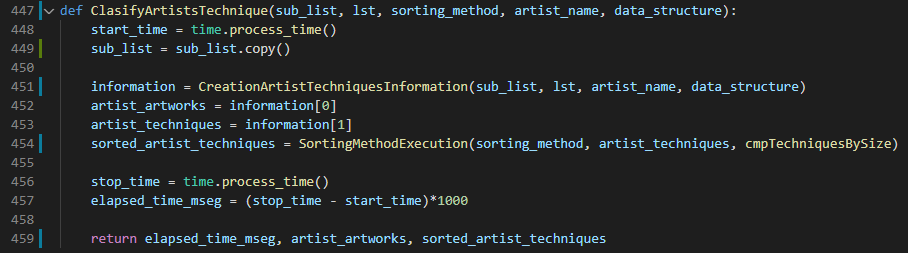


La complejidad asociada al requerimiento 2 del Reto 2 está dada por el algoritmo de ordenamiento utilizado. El requerimiento 2 se lleva a cabo por medio de la función **getArtworksByAdquisitionDate()**. En esta función se recurre al uso del TAD map **catalog[‘adquisition\_years’]**, el cual fue creado 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)**.

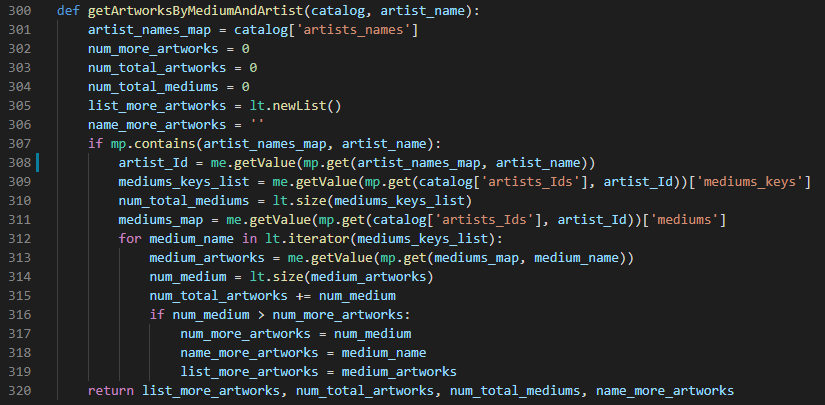
**Requerimiento 3**

* **Reto 1**



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**

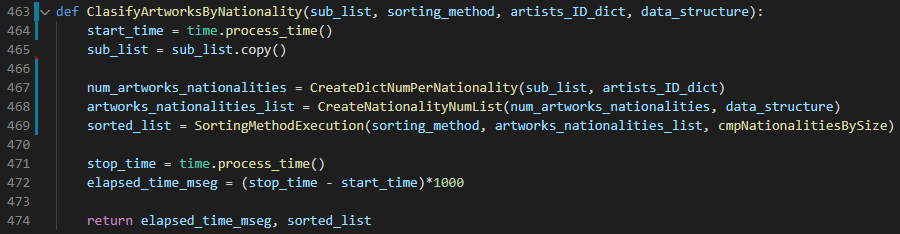


La complejidad asociada al requerimiento 3 del Reto 2 está dada por **O(1)**. El requerimiento 3 se lleva a cabo por medio de la función **getArtworksByMediumAndArtist()**. En esta función se recurre al uso de los TAD map **catalog[‘artists\_names’]** y **catalog[‘artists\_Ids’]**, los cuales fueron 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)**.

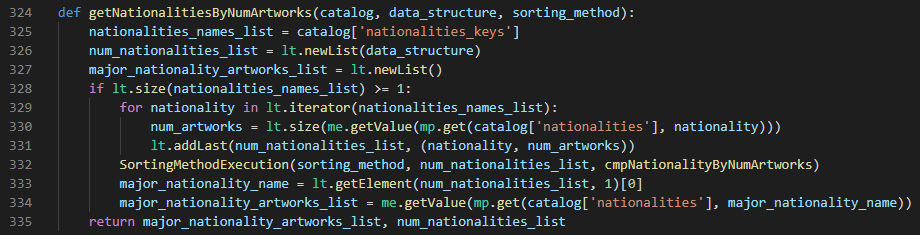
**Requerimiento 4**

* **Reto 1**



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**

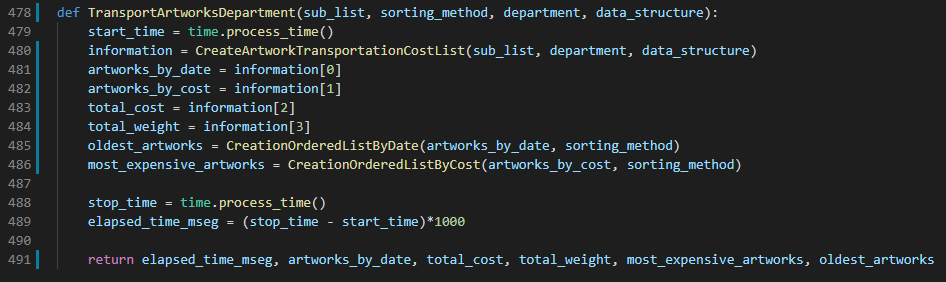


La complejidad asociada al requerimiento 4 del Reto 2 está dada por **O(1)**. El requerimiento 4 se lleva a cabo por medio de la función **getNationalitiesByNumArtworks()**. En esta función se recurre al uso del TAD map **catalog[‘nationalities’]**, el cual fue creado en el cargue de datos. Las llaves 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)**.

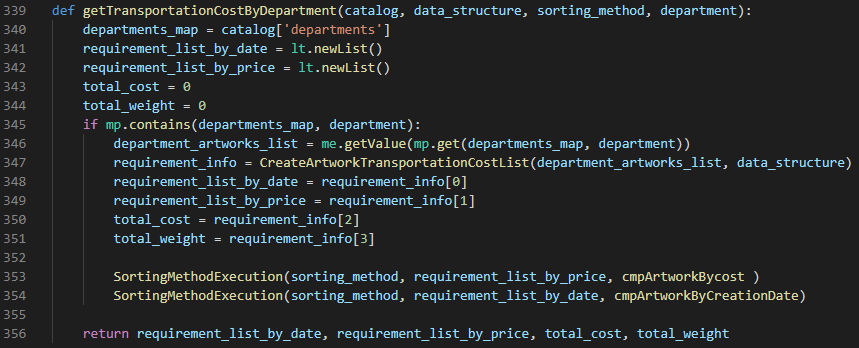
**Requerimiento 5**

* **Reto 1**



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**

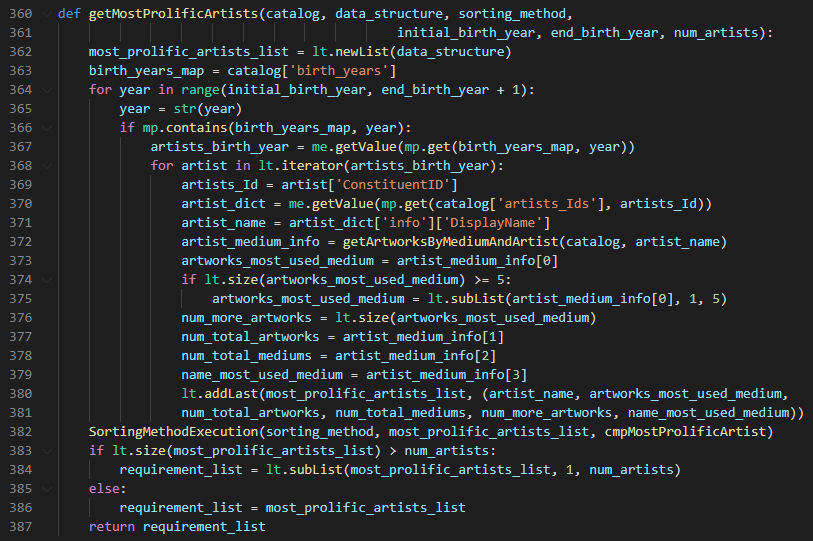


La complejidad asociada al requerimiento 5 del Reto 2 está dada por el algoritmo de ordenamiento utilizado. El requerimiento 5 se lleva a cabo por medio de la función **getTransportationCostByDepartment()**. En esta función se recurre al uso del TAD map **catalog[‘departments]**, el cual fue creado en 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**



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’.