**Análisis de Complejidad**

**Nombre:** Nicolas Merchan Cuestas

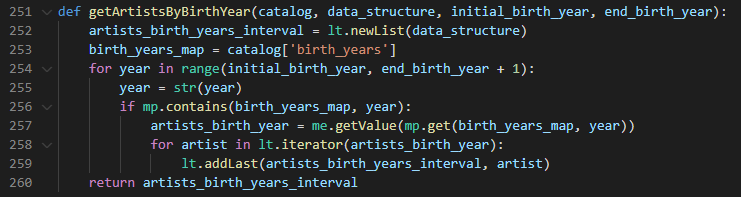
**Código:** 202112109

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

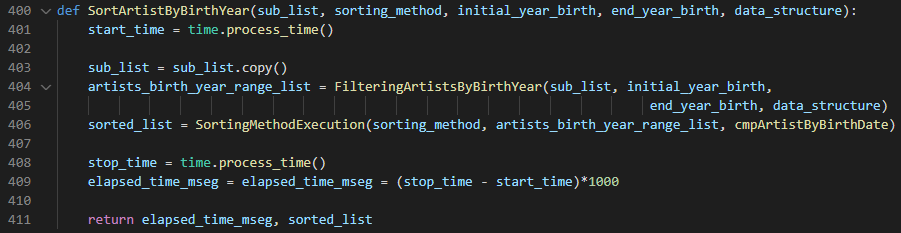
**Notas:**

* Los resultados de las pruebas de ejecución y las correspondientes gráficas de representación 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.
* 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**



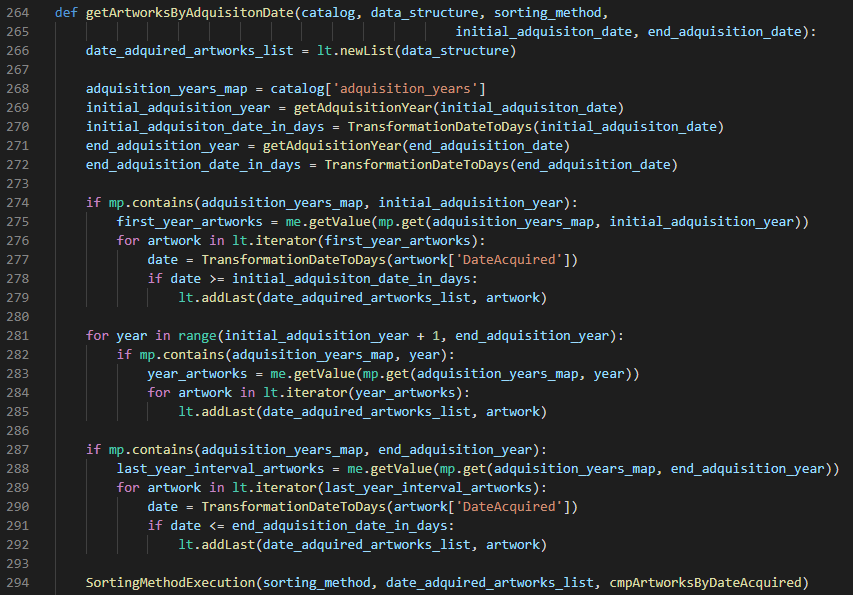
La complejidad asociada a **getArtistsByBirthYear()** en el Reto 2 es **O(1)**, dado que la función adquiere la información de los artistas nacidos en un rango de años específico por medio de la tabla de hash **catalog[‘birth\_years’]**, la cual fue creada en el cargue de datos. En dicha tabla de hash la llave es el año de nacimiento y el valor asociado a cada llave es una lista con los artistas nacidos en el año correspondiente. De ese modo, para obtener los artistas nacidos en un rango de años dado, solo es necesario consultar los respectivos valores de los años en el rango.



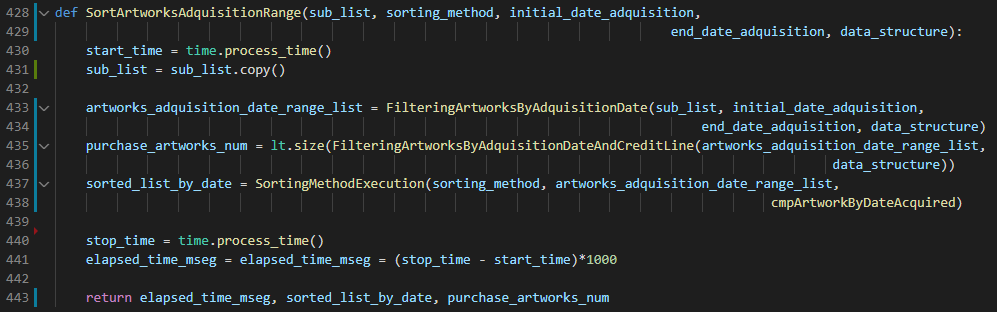
La complejidad asociada a **FilteringArtistsByBirthYear()** en el Reto 1 es **O(n)**, dado que separan los artistas nacidos en el rango de años indicado por medio de la comparación individual de cada elemento. De manera similar, la función **SortingMethodExecution()** realiza un ordenamiento en función del año de nacimiento de los artistas nacidos en el rango de años especificado. La complejidad de esta operación depende del tipo de algoritmo de ordenamiento utilizado. La complejidad es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Finalmente, la complejidad es modelada en su mayor parte por la complejidad en **SortingMethodExecution()**.

La complejidad del requerimiento 1 es mejor en el Reto 2 que en el Reto 1 por una diferencia de **O(1)** a **O(n)** al momento de buscar los artistas nacidos en un rango de años dados.

**Requerimiento 2**



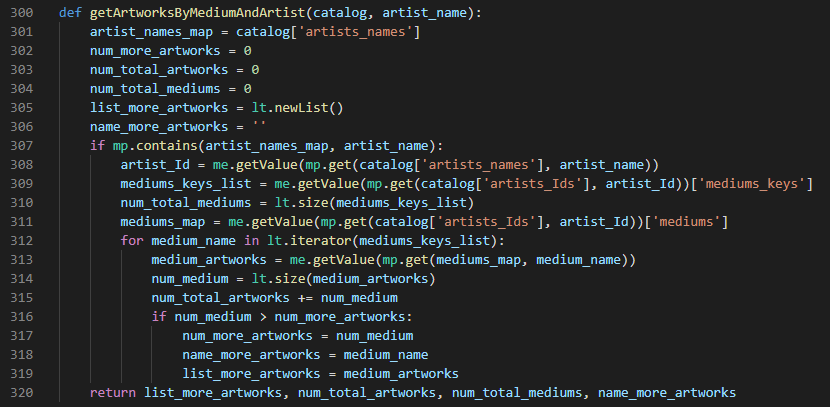
La complejidad asociada a **getArtworksByAdquisitionDate()** en el Reto 2 está determinada principalmente por el tipo de algoritmo de ordenamiento utilizado. Por una parte, la función hace uso de la tabla de hash **catalog[‘adquisition\_years’]** para encontrar todas las obras de arte en un rango de fechas. Las llaves de **catalog[‘adquisition\_years’]** son los años de adquisición y el valor asociado a dicha llave es una lista con todas las obras de arte adquiridas en el año en cuestión. La tabla de hash **catalog[‘adquisition\_years’]** fue creada en el cargue de datos. Por otra parte, la complejidad del proceso de ordenamiento depende del algoritmo utilizado. La complejidad es **O(n)**, **O(nlog(n)),** **O(nlog(n))** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.



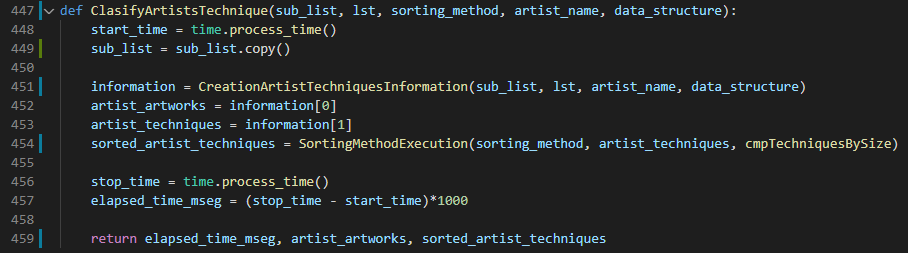
La complejidad asociada a **FilteringArtworksByAduisitionDate()** en el Reto 1 es **O(n)**, dado que separan las obras de arte adquiridas en el rango de fechas indicado por medio de la comparación individual de cada elemento. Igualmente, la complejidad de **FilteringArtworksByAduisitionDate()** es **O(n)**, porque separa las obras de arte adquiridas por compra revisando individualmente cada obra de arte. De manera similar, la función **SortingMethodExecution()** realiza un ordenamiento en función de la fecha de adquisición de las obras adquiridas en el rango de fechas especificado. La complejidad de esta operación depende del tipo de algoritmo de ordenamiento utilizado. La complejidad es **O(n)**, **O(nlog(n)),** **O(nlog(n))** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Ello se debe a que las obras de arte fueron agregadas a la lista de clasificación a medida que fueron adquiridas por el museo. De ese modo, existe un orden dentro de la lista a ordenar. Finalmente, la complejidad es modelada en su mayor parte por la complejidad en **SortingMethodExecution()**.

La complejidad del requerimiento 2 es más favorable en el Reto 2 que en el Reto 1. La diferencia en complejidades de los retos se da en la búsqueda de las obras de arte adquiridas en un rango de fechas dado. Así la complejidad de búsqueda es **O(n)** y **O(1)** para el Reto 1 y el Reto 2, respectivamente. El Reto 2 se hace uso de una tabla de hash para encontrar las obras de arte.

**Requerimiento 3**



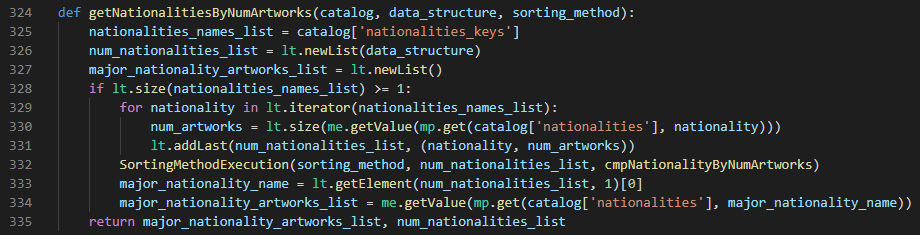
La complejidad asociada a **getArtworksByMediumAndArtist()** en el Reto 2 es **O(1)**. Ello se debe a que se hace uso de la tabla de hash **catalog[‘artists\_Ids’]** y se simplifica la complejidad del proceso de búsqueda de las obras de arte asociadas al artista a **O(1)**. Las llaves de **catalog[‘artists\_Ids’]** son los códigos de identificación únicos de los artistas, estos son obtenidos por medio de la tabla de hash **catalog[‘artists\_names’]**, la cual a su vez tiene como llaves los nombres de los artistas y como valor el código de identificación asociado a los artistas. Los valores asociados a las llaves de **catalog[‘artists\_Ids’]**, son diccionarios que contienen la información del artista y las obras de arte ordenadas por técnica. Tanto **catalog[‘artists\_Ids’]**, como **catalog[‘artists\_names’]** fueron creadas en el cargue de datos. Finalmente, solo es necesario comparar la cantidad de obras asociadas a una técnica de un artista utilizando la información del valor de la llave en **catalog[‘artists\_Ids’]**.



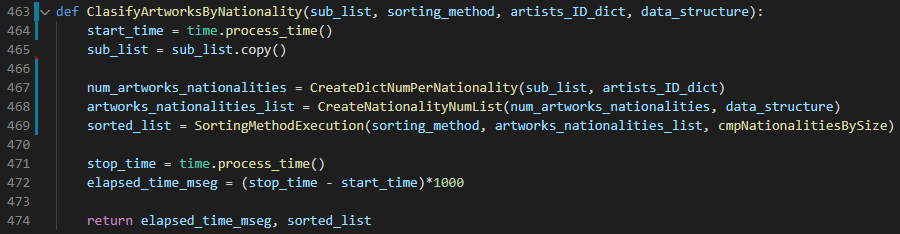
La función **CreationArtistsTechniquesInformation()** en el Reto 1 crea una TAD lista que contiene como elementos una lista con el nombre de la técnica y una TAD lista de todas la obras del autor que hacen uso de dicha técnica. La complejidad de esta función es **O(n)**, porque la función compara todas las obras respecto al autor y técnica utilizada en la mismas. De manera similar, la función **SortingMethodExecution()** realiza un ordenamiento en función de la cantidad de obras del artista en cuestión por técnica utilizada . La complejidad de esta operación depende del tipo de algoritmo de ordenamiento utilizado. La complejidad es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Finalmente, la complejidad es modelada en su mayor parte por la complejidad en **SortingMethodExecution()**.

La complejidad del requerimiento 3 es mejor en el Reto 2 que en el Reto 1, dado que la complejidad asociada a la búsqueda de datos es **O(1)**  y **O(n)** en los casos del Reto 2 y Reto 1, respectivamente.

**Requerimiento 4**



La complejidad asociada a **getNationalitiesByNumArtworks()** en el Reto 2 es aproximadamente **O(1)**. Ello se debe a que existe una cantidad fija de nacionalidades en la tabla de hash **catalog[‘nationalities’]**, la cual tiene como llaves el nombre de las nacionalidades y como valores asociados las obras de arte de la nacionalidad asociada. Así, la clasificación de nacionalidades por número de obras de arte se hace recorriendo la lista de nacionalidades **catalog[‘nationalities\_keys’]¸** la cual tiene tamaño constante a medida que se agregan datos, y comparando el tamaño de las listas de obras de arte asociadas. Tanto **catalog[‘nationalities’]**, como **catalog[‘nationalities’]** fueron creadas en el cargue de datos.

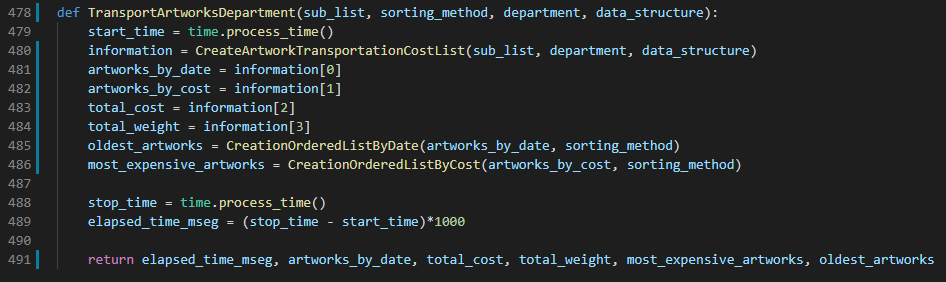


La función **CreateDictNumPerNationality()** en el Reto 1 cuenta la cantidad de obras de arte de cada nacionalidad y guarda dicha información en un diccionario donde la nacionalidad es la llave y el número de obras de arte es el valor de dicha llave. La función **CreateNationalityNumList()** convierte el diccionario generado en **CreateDictNumPerNationality()** en un TAD lista donde los elementos es son lista que contienen las nacionalidades y sus respectivos números de obras. La complejidad del proceso anteriormente mencionado es **O(n)**, dado que para completarlo es necesario recorrer la lista exactamente una vez comparando todas las obras de arte. De manera similar, la función **SortingMethodExecution()** realiza un ordenamiento en función de la cantidad de obras del artista en cuestión por nacionalidad . La complejidad de esta operación depende del tipo de algoritmo de ordenamiento utilizado. La complejidad es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Finalmente, la complejidad es modelada en su mayor parte por la complejidad en **SortingMethodExecution()**.

La complejidad del requerimiento 4 es mejor en el Reto 2 que en el Reto 1, dado que la complejidad en el Reto 2 es constante y aquella del Reto 1 depende del algoritmo de ordenamiento.

**Requerimiento 5**

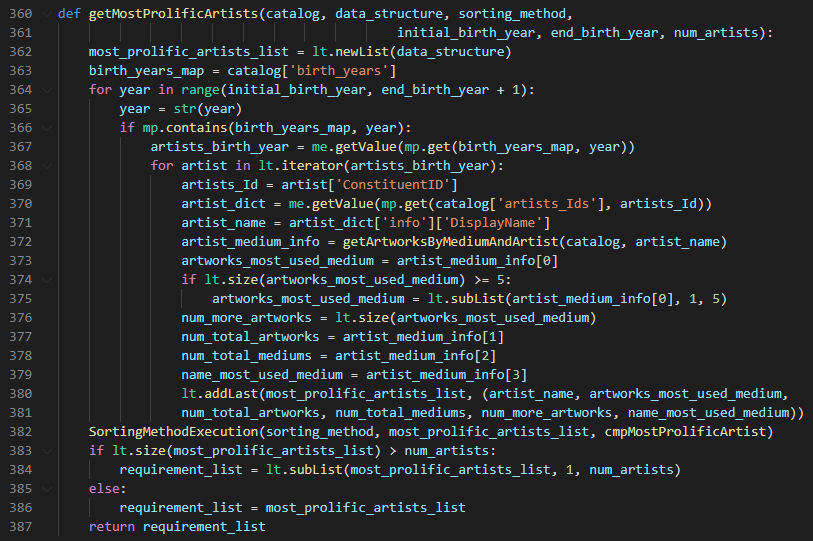
La complejidad de la función **getTransportationByDepartment()** en el Reto 2 está determinada por el algoritmo de ordenamiento utilizado. Por un parte, obtener las obras de arte del departamento tiene una complejidad de **O(1)**, dado que este proceso se hace mediante la lista de hash **catalog[‘departments’]**. Las llaves de **catalog[‘departments’]** son los nombres de los departamentos del museo y los valores asociados a dichas llaves son las listas de las respectivas obras de arte del departamento. Por otra parte, es necesario ordenar la lista de las obras de arte del departamento. De ese modo, la complejidad es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.



La función **CreateArtworkTransportationCostList()** en el Reto 1 calcula el valor de transporte de cada obra de arte del departamento ingresado por el usuario. La complejidad de esta función es **O(n)**, dado que la misma recorre todas las obras de arte y verifica si pertenecen al departamento indicado y calcula el costo de transporte simultáneamente. Posteriormente, las funciones **CreationOrderedListByDate()**  y **CreationOrderedListByCost**  ordenan la lista generada en **CreateArtworkTransportationCostList()** en base a la fecha de creación y costo de transporte, respectivamente. La complejidad de esta operación depende del tipo de algoritmo de ordenamiento utilizado. La complejidad es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente. Finalmente, la complejidad es modelada en su mayor parte por la complejidad en **SortingMethodExecution()**.

La complejidad del requerimiento 5 es mejor en el Reto 2 que en el Reto 1, dado que el proceso inicial de búsqueda de las obras pertenecientes al departamento ingresado por el usuario tiene un complejidad de **O(1)** y **O(n)** para el Reto 2 y el Reto 1, respectivamente.

**Requerimiento 6**



La complejidad de la función **getMostProlificArtists()** está determinada por el algoritmo de ordenamiento utilizado. Por una parte, obtener los artistas nacidos en un rango de fechas por medio del uso de la tabla de hash **catalog[‘birth\_years ]** tiene una complejidad de **O(1)**. Así mismo, la obtención de la cantidad de obras y medios utilizados por cada artista nacido en el rango de fechas por medio de la función **getArtworkByMediumAndArtist()** tiene una complejidad de **O(1)**. Por otra parte, el proceso de ordenamiento de los artistas en función de la cantidad de obras y medios registrados tiene una complejidad acorde al algoritmo de ordenamiento utilizado. De ese modo, la complejidad es **O(n^2)**, **O(n^3/2)**, **O(n^2)** y **O(nlog(n))** para los algoritmos Insertion Sort, Shell Sort, Quick Sort y Merge Sort, respectivamente.