Ivan Camilo Ballén Méndez – 202011440 - i.ballen@uniandes.edu.co
María José Sáenz Rodríguez – 202013542 – m.saenzr@uniandes.edu.co

Análisis de complejidad Reto #1

Implementación catálogo de videos:

Para el TAD lista que tiene como función almacenar todo el catálogo de videos que entra como archivo .csv se decidió implementar una estructura de datos de tipo "ARRAY LIST". Nuestro grupo se decantó por esta opción debido a la implementación de las funciones de carga de datos las cuales giran en torno a la función addLast(), la cual tiene una complejidad de O(n) en "SINGLE LINKED" (en caso de que no exista referencia al último) y O(1) "ARRAY LIST". Este tipo de implementación no es tan relevante dentro de las demás funciones debido a que únicamente se itera para poder crear una serie de listas auxiliares y la función lt.iterator() parece tener un orden de complejidad independiente a la estructura de datos del TAD lista..

Requerimiento 1 y 4:

En la realización de estos dos requerimientos se decide implementar la función "sort_sublist". Sus características se basan en la utilización de una estructura de datos SINGLED LINKED, debido a su eficiencia implementando funciones como "addFirst", en la cual posee un orden de crecimiento O (1). El ordenamiento utilizado para la ejecución de los requerimientos es Merge Sort, donde su complejidad se determina teóricamente, es decir, corresponde a O (n log (n)) para todos los casos. Además, se implementa el tratamiento de errores Try- except, con el fin de obtener un código más eficaz, porque solo debe evaluar la longitud de la sublista una vez.

Requerimiento 2 y 3:

Requerimiento #2: Ivan Camilo Ballén Méndez

Requerimiento #3: María José Sáenz Rodríguez

Para el desarrollo de estos dos requerimientos se implementó la función "mostTrendingVideo", donde de acuerdo con la arquitectura MVC, en el model se implementa una estructura de datos SINGLED LINKED, para realizar una lista auxiliar con el fin de trabajar los datos de una manera más cómoda. La razón por la cual se escoge este tipo de estructura es por su facilidad en la indexación, pues al utilizar la función "addFirst" el orden de complejidad es de O (1), por lo que la eficiencia en la ejecución del código es notoria. Finalmente, el ordenamiento utilizado es Merge Sort, ya que es eficiente y, por lo tanto, se determina que el orden de complejidad es O (n log (n)).

De acuerdo con todos los requerimientos analizados, se evidencia que el tipo de ordenamiento para cada uno corresponde a Merge Sort. En efecto, se decide utilizar este tipo de ordenamiento, ya que mantiene su orden de complejidad constante para cualquier caso. Además, su determinación se basa en las pruebas experimentales de los laboratorios pasados, donde conforme a las características de las dos maquinas y los tiempos de ejecución obtenidos, el mejor ordenamiento fue este.