# **Observaciones del reto 2**

Ernesto Perez – 202112530 – ec.perez@uniandes.edu.co

Nicolás Saavedra – 20212963 - [n.saavedrag@uniandes.edu.co](mailto:n.saavedrag@uniandes.edu.co)

Requerimiento 3: Nicolás Saavedra

Requerimiento 4: Ernesto Perez

# **Análisis de complejidad**

# **Pruebas de rendimiento**

Especificaciones de la máquina de prueba

|  |  |
| --- | --- |
|  |  |
| Procesadores | AMD Ryzen 5 3600 6-Core Processor, 3593 Mhz, 6 Core(s), 12 Logical Processor(s) |
| Memoria RAM | 16GB |
| Sistema operativo | Microsoft Windows 10 pro 64-bits |

**Análisis de complejidad**

|  |  |  |
| --- | --- | --- |
| Operación | Complejidad temporal | Justificación |
| Cargar Datos | O(n) | Debido a que solo se recorre por cada uno de los archivos Artworks y Artists una sola vez. |
| Requerimiento 1 | O(klog(k))  Con k=log(n) | La búsqueda de artistas que están bajo el rango de fechas se hace en O(1) sin embargo, al aplicar merge sort sobre la lista de artistas que están bajo el rango, la complejidad temporal aumenta a O(klog(k)) donde k es el tamaño de la lista de artistas bajo el rango de fechas igual a log(n). |
| Requerimiento 2 | O(klog(k))  Con k=log(n) | Se recorre una vez la lista de las obras, sin embargo, se realiza un sort sobre las fechas en las obras por lo que O (n log(n)) toma prioridad. Al igual en el requerimiento 1 k es el tamaño de la lista de obras bajo el rango de fechas igual a log(n). En tiempos este requerimiento es mayor al 1 debido al tamaño del archivo(n). |
| Requerimiento 3 | O(n) | Solo se recorre una vez la lista de artistas y de obras y se utilizan diccionarios con el fin de evitar más iteraciones y optimizar el rendimiento. |
| Requerimiento 4 | O(log(n)) | La única iteración que se hace en el requerimiento es sobre la lista de obras únicas bajo la nacionalidad, una lista pequeña en comparación con todas las obras de arte. |
| Requerimiento 5 | O(n)+O(k log(k)) | Donde n es el tamaño de las obras totales y k es el tamaño del departamento el cual se está buscando debido a que solo se itera una vez las obras y se hace un sort en las obras del departamento elegido. Si el departamento de obras es muy grande se tendrá en cuenta el comportamiento linearitmico por sobre el lineal. |

**Tiempo de ejecución promedio por requerimiento y crecimiento temporal**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Tiempo de ejecución para cada función [ms]** | | | | | |
| **Porcentaje de datos** | **Cargar datos** | **Req1** | **Req2** | **Req3** | **Req4** | **Req5** |
| **10%** | 994.79 | 0.00 | 265.63 | 15.63 | 10.42 | 390.63 |
| **50%** | 4833.33 | 0.00 | 1552.08 | 52.08 | 31.25 | 2197.92 |
| **80%** | 7875.00 | 15.63 | 2557.29 | 78.13 | 41.67 | 3609.38 |
| **100%** | 9333.33 | 15.63 | 3270.83 | 93.75 | 46.88 | 4515.63 |

*En las funciones con entradas se usaron los mismos datos de entrada de los ejemplos.*

**Comparación Tiempo de ejecución promedio Reto 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Tiempo de ejecución para cada función [ms]** | | | | | |
| **Porcentaje de datos** | **Cargar datos** | **Req1** | **Req2** | **Req3** | **Req4** | **Req5** |
| **10%** | 171.88 | 146.88 | 640.625 | 0.00 | 31.25 | 432.29 |
| **50%** | 768.75 | 286.46 | 3531.25 | 42.97 | 93.75 | 2398.44 |
| **80%** | 1160.16 | 333.33 | 5753.91 | 62.5 | 128.91 | 3867.19 |
| **100%** | 1300 | 371.09 | 7278.13 | 72.92 | 171.88 | 4964.81 |

Si bien en el reto 1 se usaron diccionarios de Python para optimizar el tiempo de ejecución, si se puede observar diferencias de tiempo grande (no tan grande a como se vería sin el uso de dict en el reto1). Todos los requerimientos tienen un tiempo de ejecución menor en el reto 2 que en el reto 1 sin embargo, el tiempo de carga de datos si aumenta en el reto 2 debido a la creación de varios índices (mapas) en el catálogo que sirven para optimizar tiempos de búsqueda mas adelante.

**Graficas de tiempo de ejecución promedio por requerimiento y crecimiento temporal**