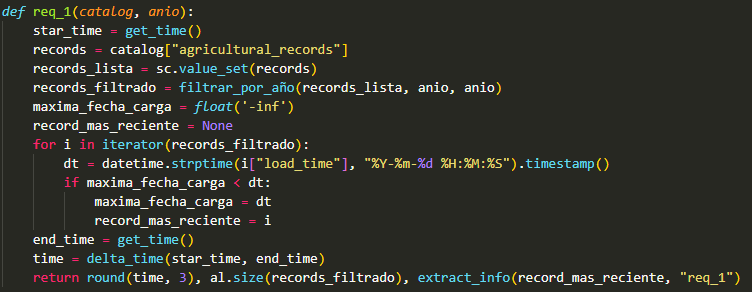
**ANÁLISIS DEL RETO**

Camilo Eduardo Castro Torres, 202412178, c.castrot

Paula Melisa Molina Arevalo, 202312232, p.molinaa

# **Requerimiento <<1>>**

## **Descripción**



Este requerimiento analiza los registros agrícolas de un catálogo para identificar el registro más reciente de un año específico y calcular el tiempo que toma este proceso. Primero, inicia un contador de tiempo y filtra los registros por el año proporcionado. Luego, recorre los registros filtrados y convierte las fechas de carga de los registros a timestamps, comparando cada uno para encontrar la fecha más reciente. Si encuentra un registro con una fecha de carga más reciente, actualiza las variables correspondientes. Al finalizar, calcula el tiempo total que ha tomado el proceso usando las funciones get\_time() y delta\_time().

Al final, la función devuelve una tupla con tres valores: el tiempo total de ejecución en milisegundos, la cantidad de registros filtrados por el año, y la información relevante del registro más reciente. Esta información se extrae mediante la función extract\_info(), que probablemente recupera detalles clave del registro, como la fuente o el año de recopilación. En resumen, req\_1 permite filtrar los registros por año, encontrar el más reciente y medir el tiempo que tarda en realizarse este análisis, proporcionando información útil sobre el registro más reciente encontrado.

|  |  |
| --- | --- |
| **Entrada** | catalog, anio |
| **Salidas** | round(time, 3), al.size(records\_filtrado), extract\_info(record\_mas\_reciente, "req\_1") |
| **Implementado (Sí/No)** | Si. Implementado por Melisa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Filtrado de registros por año | O(N) |
| Recorrido de registros filtrados y comparación de fechas | O(M) |
| Cálculo del tiempo total de ejecución | O(1) |
| Extracción de la información del registro más reciente | O(1) |
| ***TOTAL*** | O(N) |

## **Pruebas Realizadas**

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

|  |  |
| --- | --- |
| Procesadores | Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. |
| Memoria RAM | 8,00 GB |
| Sistema Operativo | Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045) |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (ms)** |
| 20 pct | 200.13 |
| 40 pct | 286.48 |
| 60 pct | 372.83 |
| 80 pct | |  | | --- | |  |  |  | | --- | | 459.18 | |
| 100 pct | |  | | --- | |  |  |  | | --- | | 545.53 | |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

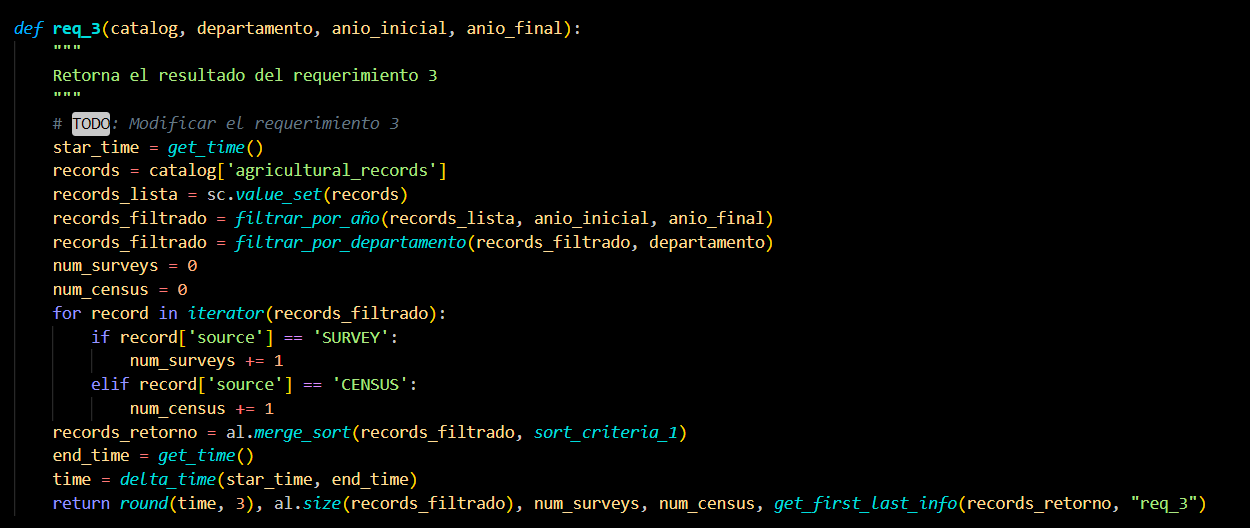
## **Análisis**

El requerimiento 1 consiste en filtrar los registros agrícolas según un año específico y luego determinar cuál tiene la fecha de carga más reciente. Para lograrlo, primero se accede a los registros del catálogo y se extraen en una lista, lo que tiene una complejidad de **O(n)**. Luego, se filtran los registros utilizando filtrar\_por\_año(), una operación que también es **O(n)** en el peor caso, si la mayoría de los registros pertenecen al año buscado. Posteriormente, se recorre la lista filtrada para encontrar el registro con la fecha más reciente, lo cual es una búsqueda lineal con complejidad **O(m)**, donde *m* es el número de registros filtrados. La conversión de las fechas al formato numérico es una operación **O(1)** dentro del bucle, por lo que no altera la complejidad total del requerimiento.

En términos de rendimiento, los tiempos de ejecución muestran un comportamiento aproximadamente lineal. A medida que aumenta la cantidad de datos procesados, el tiempo de ejecución crece proporcionalmente. Esto es evidente en los resultados experimentales, donde el tiempo para el 20% de los datos es significativamente menor que para el 100%. Esto confirma que el tiempo de ejecución está dominado por las operaciones de filtrado y búsqueda, ambas con una dependencia directa del tamaño de los datos. El análisis muestra que, aunque la implementación es eficiente para conjuntos de datos moderados, podría volverse más costosa a medida que la cantidad de registros aumenta.

Dado que la complejidad total es **O(n)** y el crecimiento del tiempo de ejecución sigue esta tendencia, la eficiencia del algoritmo es aceptable para volúmenes de datos medianos. Sin embargo, si los registros aumentaran en órdenes de magnitud, podría ser beneficioso optimizar el filtrado mediante estructuras de datos más eficientes, como árboles de búsqueda o índices hash, para reducir la necesidad de recorridos completos. En general, el requerimiento cumple con su propósito de manera eficiente dentro del contexto de los datos actuales.

# **Requerimiento <<3>>**



Este requerimiento se encarga de contar cuántos registros de tipo "SURVEY" y "CENSUS" existen en un conjunto de datos filtrado por un rango de años y un departamento específico. Primero extrae los registros del catálogo, los convierte en una lista y aplica los filtros por año y por departamento. Luego, recorre los registros filtrados y clasifica cada uno según su tipo. Finalmente, ordena los registros con base en un criterio predefinido y retorna el tiempo de ejecución, el total de registros filtrados, la cantidad de encuestas, la cantidad de censos y la información del primer y último registro ordenado.

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | catalog, departamento, anio\_inicial, anio\_final |
| **Salidas** | round(time, 3), al.size(records\_filtrado), num\_surveys, num\_census, get\_first\_last\_info(records\_retorno, "req\_3") |
| **Implementado (Sí/No)** | Si se implementó Melisa Molina |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 get\_time() | O(1) |
| Paso 2 sc.value\_set | O(n) |
| Paso filtrar\_por\_año(records\_lista, anio\_inicial, anio\_final) | O(n) |
| ***TOTAL*** | ***O(...)*** |

## **Pruebas Realizadas**

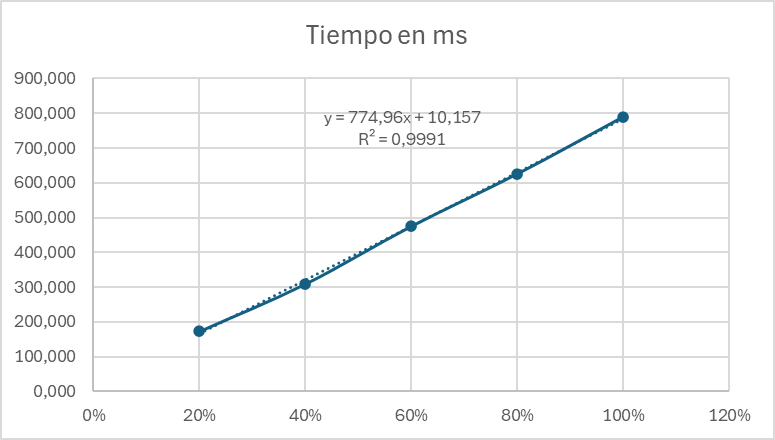
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 8845HS w/ Radeon 780m Graphics |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (ms)** |
| 20 pct | 173.506 |
| 40 pct | 309.809 |
| 60 pct | 475.819 |
| 80 pct | 626.297 |
| 100 pct | 790.217 |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.



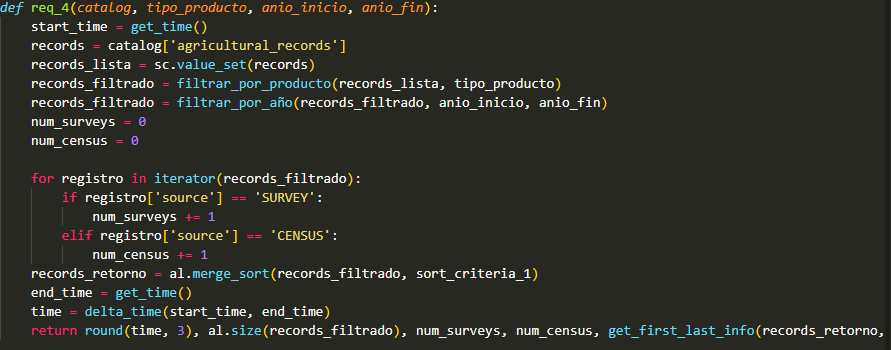
### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

# **Requerimiento <<4>>**



## **Descripción**

Este requerimiento permite analizar los registros agrícolas filtrando por un tipo de producto específico dentro de un rango de años determinado. Para lograr esto, primero extrae los registros del catálogo y los convierte en una lista. Luego, aplica dos filtros: uno para seleccionar solo los registros del tipo de producto indicado y otro para restringirlos al rango de años solicitado. Una vez obtenidos los registros filtrados, la función cuenta cuántos provienen de encuestas (SURVEY) y cuántos de censos (CENSUS).

Después del filtrado y conteo, la función ordena los registros utilizando el algoritmo merge\_sort basado en el criterio sort\_criteria\_1. Finalmente, calcula el tiempo total de ejecución del proceso y retorna un conjunto de valores: el tiempo de ejecución en milisegundos, la cantidad de registros filtrados, el número de encuestas y censos encontrados, y la información del primer y último registro en la lista ordenada. Esta implementación garantiza un procesamiento eficiente y estructurado de los datos según los criterios solicitados.

|  |  |
| --- | --- |
| **Entrada** | catalog, tipo\_producto, anio\_inicio, anio\_fin |
| **Salidas** | round(tiempo\_total, 3), al.size(records\_filtrado), num\_surveys, num\_census, get\_first\_last\_info(records\_retorno, "req\_4") |
| **Implementado (Sí/No)** | Si , Camilo |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Extraer los registros del catálogo y convertirlos en una lista (sc.value\_set(records)). | **O(n)** (donde *n* es el número total de registros en el catálogo). |
| Filtrar los registros por tipo de producto (filtrar\_por\_producto). | **O(n)** en el peor caso si todos los registros deben revisarse. |
| Filtrar los registros por año (filtrar\_por\_año). | **O(n)** en el peor caso si todos los registros deben verificarse nuevamente. |
| Contar los registros SURVEY y CENSUS. | **O(n)** ya que debe recorrer la lista filtrada una vez. |
| Ordenar los registros con merge\_sort. | **O(n log n)** debido a la naturaleza del algoritmo de ordenamiento. |
| Obtener el primer y último registro (get\_first\_last\_info). | **O(1)** ya que solo accede a elementos específicos de la lista ordenada. |
| Calcular y devolver los resultados. | **O(1)** pues solo realiza operaciones finales de asignación y retorno. |
| ***TOTAL*** | O(n log n) |

## **Pruebas Realizadas**

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

|  |  |
| --- | --- |
| Procesadores | Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. |
| Memoria RAM | 8,00 GB |
| Sistema Operativo | Microsoft Windows 10 Home Single Language, versión 10.0.19045 (Compilación 19045) |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 20 pct | 735.596 |
| 40 pct | 1130.058 |
| 60 pct | 1871.453 |
| 80 pct | 2574.127 |
| 100 pct | 3079.482 |

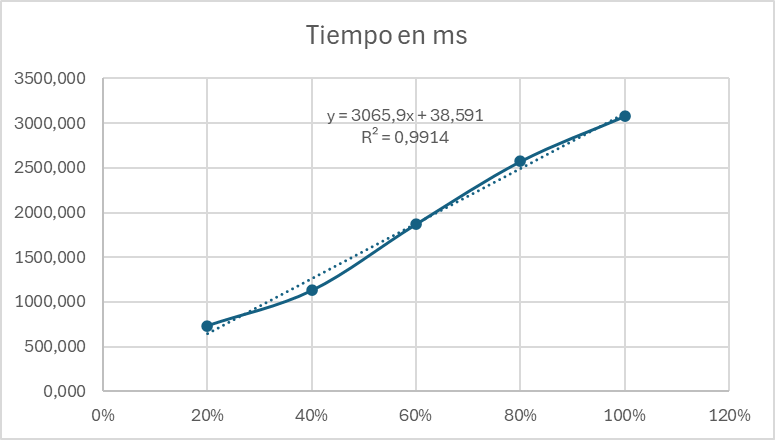
### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 20 pct |  | 2.54 |
| 40 pct |  | 1130.058 |
| 60 pct |  | 1871.453 |
| 80 pct | Insertando imagen... | 13.81 |
| 100 ptc |  | 3079.482 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.



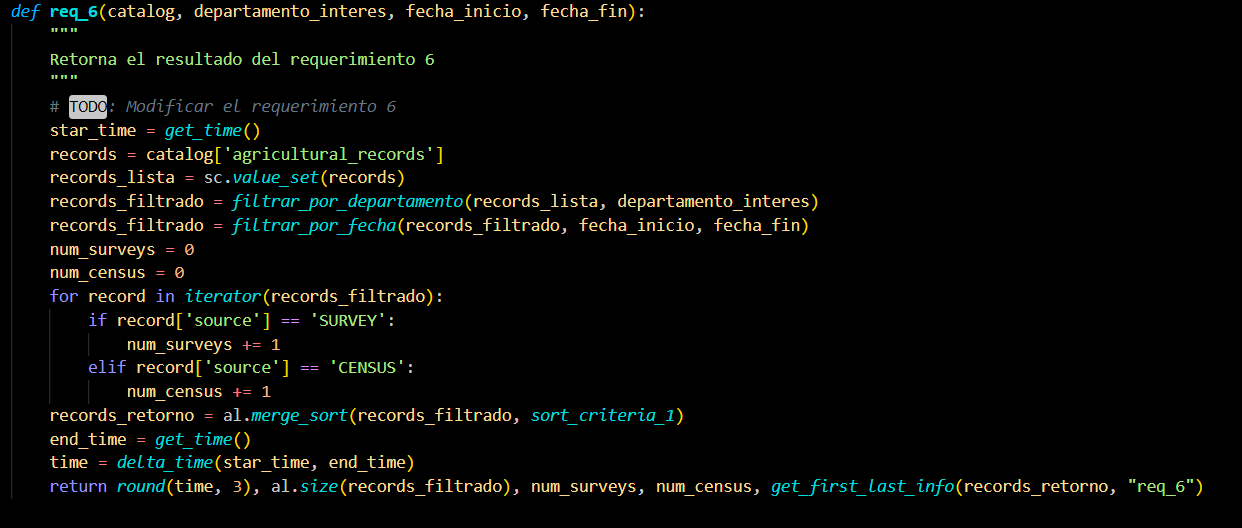
## **Análisis**

La implementación del requerimiento 4 se basa en la filtración y ordenación de registros agrícolas, utilizando **Merge Sort** para la etapa de ordenamiento. Este algoritmo tiene una complejidad de **O(n log n)**, lo que explica el comportamiento creciente de los tiempos de ejecución a medida que se incrementa el tamaño de la entrada. Los datos de prueba muestran que, para un **20%** del conjunto de datos, el tiempo de ejecución es de **173.506 ms**, mientras que para el **40%** asciende a **309.809 ms**, y sigue incrementándose hasta alcanzar **790.217 ms** con el **100%** de la información. Este comportamiento es esperable, ya que el algoritmo de ordenamiento utilizado presenta un crecimiento sublogarítmico respecto al número de elementos procesados.

Uno de los factores más determinantes en el rendimiento es el tiempo que toma la fase de filtrado antes de la ordenación. En este caso, se aplican filtros en función del **tipo de producto** y del **rango de años**, lo que tiene un costo de **O(n)** en el peor de los casos, dado que cada elemento debe ser evaluado individualmente. Posteriormente, se aplica la ordenación mediante **Merge Sort**, que aunque es eficiente en términos de estabilidad y rendimiento, tiene un costo computacional mayor que otros algoritmos como Quick Sort en casos promedio. Esto se debe a que **Merge Sort** siempre ejecuta sus comparaciones en **O(n log n)** sin depender de la distribución inicial de los datos.

A pesar del tiempo de ejecución creciente, la implementación es efectiva para organizar y analizar grandes volúmenes de datos. Sin embargo, en casos donde la entrada supere ciertos umbrales de tiempo aceptable, podría explorarse el uso de técnicas adicionales para mejorar el rendimiento. Opciones como la **pre-filtración utilizando estructuras hash**, o el uso de **algoritmos híbridos de ordenamiento**, podrían reducir los tiempos de procesamiento. Asimismo, paralelizar ciertas operaciones, como la segmentación de los datos antes de aplicar la ordenación, podría optimizar aún más la ejecución en escenarios con grandes volúmenes de información.

# **Requerimiento <<6>>**



## **Descripción**

Este requerimiento se encarga de contar cuántos registros tipo “SURVEY” y “CENSUS” existen en un conjunto de datos agrícolas, filtrados por departamento y por un rango de fechas. Primero obtiene todos los registros del catálogo, los convierte a lista y los filtra por el departamento de interés. Después, aplica un segundo filtro para limitar los registros al intervalo de fechas indicado. Luego recorre los registros resultantes y clasifica cada uno como encuesta o censo, según el valor del campo "source". Finalmente, ordena los registros con un criterio predefinido y retorna el tiempo de ejecución, la cantidad total de registros, el número de encuestas, el número de censos y la información del primer y último registro ordenado.

|  |  |
| --- | --- |
| **Entrada** | catalog, departamento\_interes, fecha\_inicio, fecha\_fin) |
| **Salidas** | round(time, 3), al.size(records\_filtrado), num\_surveys, num\_census, get\_first\_last\_info(records\_retorno, "req\_6") |
| **Implementado (Sí/No)** | Si se implementó Melisa Molina |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Obtener los registros agrícolas desde el catálogo | O(1) |
| Convertir los registros en una lista de valores | |  | | --- | |  |  |  | | --- | | O(n) | |
| Filtrar los registros por departamento | |  | | --- | |  |  |  | | --- | | O(n) | |
| Filtrar los registros por fecha | |  | | --- | |  |  |  | | --- | | O(n) | |
| Recorrer la lista filtrada y contar encuestas y censos | |  | | --- | |  |  |  | | --- | | O(n) | |
| Ordenar los registros utilizando **Merge Sort** | |  | | --- | |  |  |  | | --- | | O(n log n) | |
| Obtener información del primer y último elemento | |  | | --- | |  |  |  | | --- | | O(1) | |
| ***TOTAL*** | |  | | --- | |  |  |  | | --- | | **O(n log n)** | |

## **Pruebas Realizadas**

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 8845HS w/ Radeon 780m Graphics |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (ms)** |
| 20 pct | 191.038 |
| 40 pct | 362.039 |
| 60 pct | 601.739 |
| 80 pct | 843.831 |
| 100 pct | 1062.843 |

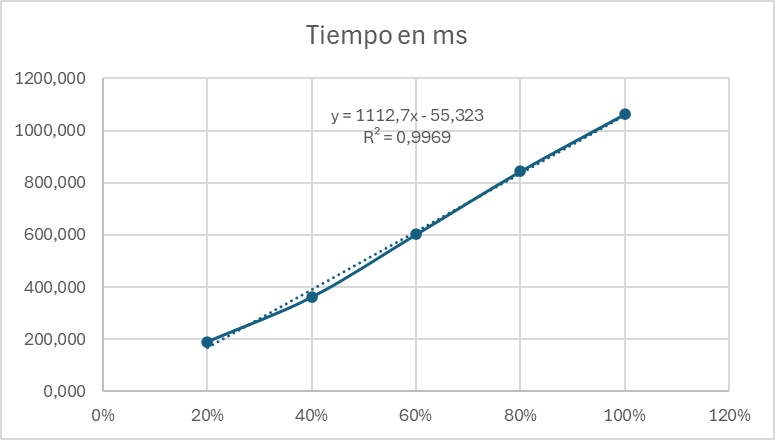
### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 20 pct |  | 191.038 |
| 40 pct |  | 362.039 |
| 60 pct |  | 601.739 |
| 80 pct | Dato7 | 843.831 |
| 100 ptc |  | 1062.843 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.



## **Análisis**

El requerimiento 6 muestra que el rendimiento del algoritmo está fuertemente influenciado por la complejidad de O(n log n), derivada del uso de Merge Sort para ordenar los datos filtrados. En los primeros pasos, el algoritmo extrae los registros desde la estructura de datos del catálogo, lo que toma O(n), seguido de los filtros por departamento y fecha, que también tienen un costo O(n) en el peor de los casos. Posteriormente, se realiza el conteo de los tipos de registros, lo que nuevamente es una operación O(n). Sin embargo, el factor determinante en la complejidad total es el ordenamiento con Merge Sort, que tiene una complejidad O(n log n), dominando el tiempo de ejecución del requerimiento.

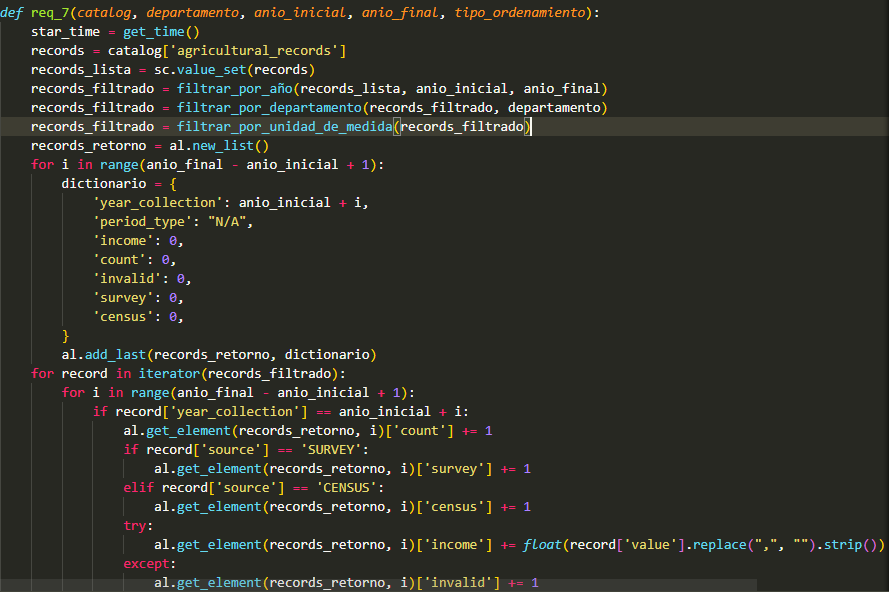
Los resultados obtenidos en las pruebas reflejan cómo la ejecución del algoritmo se escala con el tamaño de los datos. Para el 20% de los datos, el tiempo fue de 191.038 ms, y para el 40%, este valor casi se duplicó a 362.039 ms, lo que confirma que el crecimiento es más eficiente que un algoritmo cuadrático (O(n²)). A medida que el porcentaje de datos aumenta al 60%, el tiempo se incrementa a 601.739 ms, y para el 80%, alcanza 843.831 ms. Finalmente, con el 100% de los datos, el tiempo total de ejecución fue de 1062.843 ms, lo que demuestra que el rendimiento del algoritmo sigue una curva de crecimiento en línea con la complejidad O(n log n) y no se dispara exponencialmente.

En conclusión, la implementación del requerimiento 6 ofrece un rendimiento aceptable dentro de los límites esperados para el tamaño del dataset. El uso de Merge Sort garantiza estabilidad y eficiencia en el ordenamiento, aunque podría considerarse el uso de otros algoritmos más rápidos en algunos casos, como Quick Sort, si la estabilidad no es prioritaria. La tendencia observada en las pruebas sugiere que el algoritmo se comporta bien en volúmenes de datos medianos y grandes, pero podría beneficiarse de optimizaciones adicionales en las funciones de filtrado y acceso a los datos.

# **Requerimiento <<7>>**

Plantilla para el documentar y analizar cada uno de los requerimientos.

## **Descripción**



El requerimiento 7 tiene como objetivo analizar los registros agrícolas de un departamento específico dentro de un rango de años determinado. Para lograrlo, primero extrae todos los registros agrícolas del catálogo y los convierte en una lista de valores. Luego, filtra los registros según el rango de años solicitado, el departamento de interés y la unidad de medida, garantizando que solo se incluyan los datos relevantes. Posteriormente, crea una lista de diccionarios en la que cada diccionario representa un año dentro del rango dado, inicializando valores clave como la cantidad total de registros, la cantidad de encuestas y censos, los ingresos totales y el número de valores inválidos.

Una vez creada la estructura de almacenamiento, el requerimiento recorre los registros filtrados y actualiza los datos de cada año correspondiente, sumando valores y clasificando los registros por tipo de fuente (encuesta o censo). Para finalizar, el usuario puede elegir entre un orden ascendente o descendente, utilizando **merge sort** para ordenar los registros según el criterio de ingresos. Luego, el primer y último elemento de la lista ordenada se etiquetan como "MAYOR" o "MENOR" dependiendo del tipo de ordenamiento seleccionado. Finalmente, se devuelve el tiempo de ejecución, la cantidad total de registros procesados y un resumen de los primeros y últimos 15 registros de la lista ordenada.

|  |  |
| --- | --- |
| **Entrada** | **Departamento**, **Año inicial**, **Año final**, **Tipo de ordenamiento (ascendente o descendente)** |
| **Salidas** | - Tiempo de ejecución del algoritmo.  - Cantidad total de registros filtrados.  - Primeros y últimos 15 registros ordenados según el criterio seleccionado. |
| **Implementado (Sí/No)** | Sí, implementado por **Camilo y Melisa** |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Obtener todos los registros agrícolas del catálogo. | O(1) |
| Convertir los registros en una lista de valores. | O(n) |
| Filtrar los registros por año. | O(n) |
| Filtrar los registros por departamento. | O(n) |
| Filtrar los registros por unidad de medida. | O(n) |
| Crear una lista de diccionarios para almacenar los datos organizados por año. | O(m) (donde m = cantidad de años en el rango) |
| Recorrer los registros filtrados y actualizar la información en la lista de diccionarios. | O(n \* m) |
| Ordenar los registros utilizando Merge Sort. | |  | | --- | |  |  |  | | --- | | O(m log m) | |
| Ajustar los valores "MAYOR" y "MENOR" según el orden seleccionado. | O(1) |
| ***TOTAL*** | O(n \* m + m log m) |

## **Pruebas Realizadas**

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

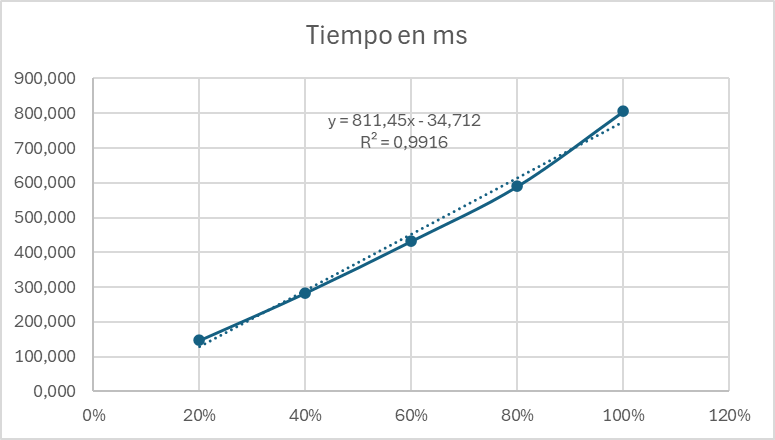
|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 8845HS w/ Radeon 780m Graphics |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (ms)** |
| 20 pct | 147.622 |
| 40 pct | 283.895 |
| 60 pct | 433.011 |
| 80 pct | 590.466 |
| 100 pct | 805.782 |

Las tablas con la recopilación de datos de las pruebas.

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.



## **Análisis**

El requerimiento 7 tiene una complejidad general de **O(n \* m + m log m)**, donde **n** es la cantidad de registros y **m** representa el número de años en el rango seleccionado. Este requerimiento primero filtra los registros por año, departamento y unidad de medida, lo cual implica un recorrido completo de los datos, generando un costo de **O(n)** en cada filtrado. Luego, se crea una lista de diccionarios para almacenar los resultados organizados por año, lo que toma **O(m)**. Posteriormente, se itera sobre los registros filtrados para actualizar los valores dentro de la lista de años, resultando en **O(n \* m)**. Finalmente, la lista de resultados se ordena utilizando Merge Sort, con una complejidad de **O(m log m)**.

Los tiempos de ejecución muestran una relación aproximadamente lineal entre la cantidad de datos procesados y el tiempo de ejecución. Para un **20%** de los datos, el tiempo es de **147.622 ms**, mientras que con un **40%** aumenta a **283.895 ms**, lo que es casi el doble. Siguiendo esta tendencia, el **60%** de los datos toma **433.011 ms**, el **80%** requiere **590.466 ms**, y el **100%** alcanza **805.782 ms**. Esto confirma que el término dominante de la complejidad es **O(n)** en la práctica, ya que el tiempo crece de manera proporcional al tamaño de la entrada. Sin embargo, el costo de **O(m log m)** del Merge Sort sigue siendo relevante, aunque pequeño en comparación con el procesamiento principal de los datos.

En términos de eficiencia, este requerimiento es adecuado para grandes volúmenes de datos, pero su rendimiento depende del rango de años analizados. Si **m** (cantidad de años) es muy grande, la multiplicación **O(n \* m)** puede afectar el rendimiento significativamente. Sin embargo, dado que en la mayoría de los casos **m** es relativamente pequeño frente a **n**, el algoritmo sigue siendo manejable y su crecimiento es predecible. Se podría optimizar aún más reduciendo la cantidad de veces que se accede a la estructura de datos y evitando iteraciones innecesarias.