**ANÁLISIS DEL RETO**

Iker Barbosa, 202424135, i.barbosac@uniandes.edu.co

Camilo Castro, 202412178, email 2

Melisa Molina, 202312232, email 3

# **Requerimiento 1**

## **Descripción**

El punto del requerimiento es identificar el último registro recopilado según un año de interés. Por este motivo, usamos la librería datetime para poder comparar las fechas de la columna load\_time. Primero, revisamos que la lista no esté vacía, si lo está, retorna None. En caso de que no esté vacía, recorremos la lista para hallar el último registro añadido por medio de comparaciones y guardarlo en una variable. Por último, recorremos la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. Al encontrar el primero forzamos la salida del bucle con return().

|  |  |
| --- | --- |
| **Entrada** | agro, year:str |
| **Salidas** | retorno\_final = {"numero\_registros": 0, "registro": {}} |
| **Implementado (Sí/No)** | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad**  O(N), la función buscar\_anios(), que se guarda en la variable lista, se usa para filtrar solamente los registros por el año que entra por parámetro. Para filtrar se debe recorrer toda la lista.  O(1), se declara el diccionario que se va a retornar el diccionario al final. |
|  |  |
|  | O(1), consultar el tamaño de la lista filtrada. |
|  | O(n), recorrer la lista otra vez para hallar la fecha mas reciente. |
|  | O(n), recorre la lista para encontrar el registro con la fecha máxima y lo añade al diccionario que retorna la función. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el año 2024.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 34.431 |
| 200000 datos |  | 58.343 |
| 300000 datos |  | 91.552 |
| 400000 datos |  | 120.980 |
| 500000 datos |  | 136.652 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

## **Análisis**

Al tratarse de recorridos de lista completos para realizar comparaciones, por temas de velocidad decidimos usar Single Linked List, la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar para recorrer la lista a su totalidad, ya que debemos comparar todos los elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

# **Requerimiento 2**

## **Descripción**

El punto del requerimiento es identificar el último registro cargado dado un departamento de interés. Similar al requerimiento anterior, usamos la librería datetime para poder comparar las fechas de la columna load\_time y así identificar el último registro cargado. En este caso, simplemente recorremos la lista y con un condicional verificamos los datos correspondan al departamento que recibimos por parámetro y guardamos las fechas en variables para compararlas. Por último, recorremos la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. Al encontrar el primero forzamos la salida del bucle con return().

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | retorno\_final = {"numero\_registros": 0, "registro": {}} |
| **Implementado (Sí/No)** | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), recorrer la lista para hallar el último registro añadido dependiendo del departamento pasado por parámetro. |
|  | O(n), recorrer la lista para hallar el último registro añadido en el departamento pasado por parámetro. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el estado de CALIFORNIA.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 29.489 |
| 200000 datos |  | 86.088 |
| 300000 datos |  | 120.451 |
| 400000 datos |  | 145.154 |
| 500000 datos |  | 166.005 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

## **Análisis**

Al tratarse de recorridos de lista completos para realizar comparaciones, por temas de velocidad decidimos usar Single Linked List, la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar para recorrer la lista a su totalidad, ya que debemos comparar todos los elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Al inicio tiene una pendiente más inclinada, pero a medida que tenemos más datos, adopta su forma lineal y una pendiente más fija. Por lo que cumple con el comportamiento lineal esperado.

# **Requerimiento 3**

## **Descripción**

El punto del requerimiento es consultar los registros recopilados para un departamento de interés dado un rango de años de recopilación. Por este motivo, se usa una función auxiliar para obtener un **ArrayList** filtrado por años de carga. Desde ahí, se recorre el **ArrayList** más pequeño y se filtra por el nombre del departamento. En caso de que el número de registros sea menor o igual a 20, se devuelve la lista completa; en caso contrario, se devuelven únicamente los primeros 5 y los últimos 5 registros. Además, se lleva un conteo de registros según su fuente de recopilación (SURVEY o CENSUS).

|  |  |
| --- | --- |
| **Entrada** | *agro\_al*, *department*: *str*, *año\_inicio*: *str*, *año\_fin*: *str* |
| **Salidas** | 1. return {           "total\_registros": size,          "survey\_count": survey,          "census\_count": census,          "registros": seleccionados      }   1. return "Error: ingreso un tipo de dato no válido" |
|  | Implementado, Camilo Castro |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | Esta función tiene una complejidad de O(N), donde N es la cantidad total de registros en agro\_al["agricultural\_records"]. La razón es que la función recorre toda la lista de registros una sola vez, verificando si cada uno se encuentra dentro del rango de años especificado y agregándolo a la lista de resultados en caso afirmativo. Como cada una de estas operaciones es de tiempo constante O(1) por cada elemento, la complejidad general queda determinada por el tamaño de la lista original, es decir, **O(N)**. |
|  | La función req\_3 tiene una complejidad de O(N + M). Primero, llama a buscar\_entre\_anios\_al, que tiene una complejidad O(N) al recorrer todos los registros. Luego, itera sobre la lista filtrada (lista), de tamaño M, para aplicar un segundo filtro por departamento y contar los tipos de fuente, lo que introduce un término O(M). Finalmente, selecciona los primeros y últimos 5 registros si hay más de 20, lo que es una operación O(1). Como el comportamiento dominante es la iteración sobre los registros originales y la lista filtrada, la complejidad total es O(N + M)..  Dado que es formato O(), entonces es **O(N)** |
|  | La función measure\_req\_3 tiene una complejidad de O(N + M), ya que simplemente mide el tiempo de ejecución de la función req\_3 sin realizar cálculos adicionales significativos. Como req\_3 es la que ejecuta la lógica de filtrado y selección de registros, su complejidad domina la función measure\_req\_3. Por lo tanto, la complejidad de measure\_req\_3 es equivalente a la de req\_3, es decir, O(N + M). Dado que es formato O(), entonces es **O(N)** |
|  | La función print\_req\_3 tiene una complejidad de O(N + M), ya que su ejecución está dominada por la llamada a measure\_req\_3, que a su vez ejecuta req\_3. Además, imprime los registros seleccionados en un bucle de O(1), ya que el número máximo de registros mostrados es constante (hasta 10 registros). Como la función req\_3 tiene una complejidad O(N + M) y print\_req\_3 no introduce iteraciones adicionales significativas, su complejidad total también es O(N + M).y retornamos el Array List. Dado que es formato O(), entonces es **O(N)** |
| ***TOTAL*** | ***O(N)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el tipo de producto US TOTAL, entre 2000 y 2001.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 8,00 GB |
| Sistema Operativo | Windows 10 Home Single Language |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

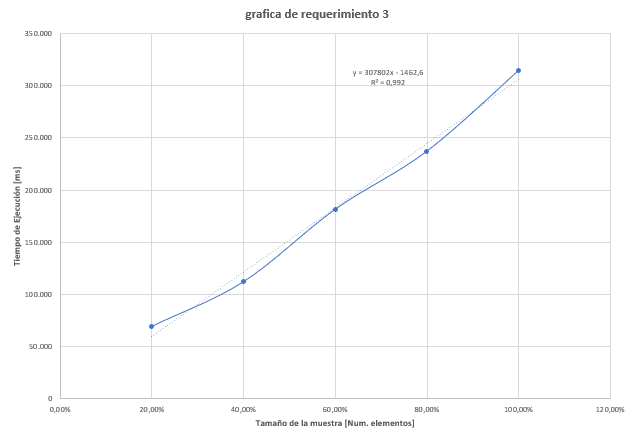
### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos | Y asi con 8 registros más | 69.608 ms |
| 200000 datos | Y asi con 8 resgistros más | 112.371 ms |
| 300000 datos | Y asi con 8 resgistros más | 181.784 ms |
| 400000 datos |  | 237.469 ms |
| 500000 datos | Y asi con 8 registros más | 314.861 ms |

### **Graficas**

Las gráfica con la representación de las pruebas realizadas.



## **Análisis**

Para el procesamiento de los datos, decidimos utilizar un Array List, ya que permite acceder rápidamente a los elementos mediante get\_element(), lo que tiene una complejidad de O(1). La implementación del requerimiento tiene un orden de complejidad O(n) en su mayoría, debido a que debemos recorrer el Array List una vez para filtrar los registros entre los años indicados, y luego recorrer nuevamente la lista filtrada para seleccionar aquellos que cumplen con el criterio del departamento.

Las operaciones de acceso a elementos (get\_element()), inserción al final (add\_last()) y conteo (size()) son todas O(1), lo que hace que las operaciones dentro del ciclo mantengan su eficiencia. Sin embargo, como el recorrido del Array List es lineal y se repite varias veces, el orden de complejidad total del requerimiento sigue siendo O(n).

Este comportamiento se puede evidenciar experimentalmente en una gráfica de tiempo de ejecución, donde se observa que el crecimiento es lineal con respecto a la cantidad de registros analizados.

# **Requerimiento 4**

## **Descripción**

El punto del requerimiento es consultar los registros recopilados para un tipo de producto de interés dado un rango de años de recopilación dado. Por este motivo, use una función auxiliar para que me dé un Array List filtrado entre años de carga. Desde acá es más fácil recorrer de nuevo un Array List más pequeño y filtrarlo por tipo de producto. En caso de que el número de registros sea menor o igual a 20 devuelvo una Pila, en caso contrario, devuelvo un Array List con acceso a elementos.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | 1. Tupla con pila, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. 2. Tupla con Array List, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. |
|  | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), la función auxiliar buscar\_entre\_anios\_al() es una función que recorre todo el Array List para filtrar entre los años recibidos por parametro. |
|  | O(n), recorrer la lista filtrada entre años para hallar los registros que coincidan con el tipo de producto que se recibió por parámetro. |
|  | O(n), si el tamaño de lista es menor o igual a 20, recorremos la lista para añadir sus elementos a una pila y retornarla. |
|  | O(n), en caso de que el tamaño de la lista sea mayor a 20. Creamos una nueva Array List con new\_list(), e iteramos la lista para añadir los primeros 5 elementos y los últimos 5 con get.element() y retornamos el Array List. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el tipo de producto EGGS, entre 1980 y 2024.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 72.495 |
| 200000 datos |  | 142.293 |
| 300000 datos |  | 249.642 |
| 400000 datos |  | 293.690 |
| 500000 datos |  | 359.728 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

## **Análisis**

Por temas de retorno decidimos usar Array List, ya que nos facilita el acceso a elementos con get\_element() y es de complejidad O(1), la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar los Array List varias veces poder realizar filtros, añadir elementos y obtener elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por poco es una recta completa, indicando su crecimiento lineal.

# **Requerimiento 6**

## **Descripción**

El punto del requerimiento es conocer la estadística del departamento de mi interés para un rango de fechas de carga de los registros dado. Por este motivo, use una función auxiliar para que me dé un Array List filtrado entre fechas de carga que funciona con la librería datetime. Desde acá es más fácil recorrer de nuevo un Array List más pequeño y filtrarlo por departamento. En caso de que el número de registros sea menor o igual a 20 devuelvo una Pila, en caso contrario, devuelvo un Array List con acceso a elementos.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | 1. Tupla con pila, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. 2. Tupla con Array List, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. |
|  | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), la función auxiliar buscar\_entre\_fechas\_al() es una función que recorre todo el Array List para filtrar entre fechas de carga recibidas por parámetro. |
|  | O(n), recorrer la lista filtrada entre fechas de carga para hallar los registros que coincidan con el tipo de departamento que se recibió por parámetro. |
|  | O(n), si el tamaño de lista es menor o igual a 20, recorremos la lista para añadir sus elementos a una pila y retornarla. |
|  | O(n), en caso de que el tamaño de la lista sea mayor a 20. Creamos una nueva Array List con new\_list(), e iteramos la lista para añadir los primeros 5 elementos y los últimos 5 con get.element() y retornamos el Array List. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos, el estado de CALIFORNIA, la fecha de carga 2012-01-01 00:00:00 y la fecha de carga 2024-11-07 12:00:00 .

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 571.472 |
| 200000 datos |  | 1167.862 |
| 300000 datos |  | 1743.198 |
| 400000 datos |  | 2334.033 |
| 500000 datos |  | 2891.148 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

## **Análisis**

Similar al requerimiento 4, decidimos usar Array List, ya que nos facilita el acceso a elementos con get\_element() y es de complejidad O(1), la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar los Array List varias veces poder realizar filtros, añadir elementos y obtener elementos.

Este comportamiento se puede evidenciar perfectamente en la gráfica, especialmente en la del Ryzen 7 5700G. Una recta completa, lo que indica su crecimiento lineal.

# **Requerimiento 7**

## **Descripción**

El punto del requerimiento es conocer el año con mayor y menor ingresos para un departamento de interés en un rango de fechas de recolección de los registros dado. Para esta consulta, solo se deben tener en cutna los registros que contengan un “$”.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** |  |
| **Implementado (Sí/No)** | Implementado, Iker Barbosa. Camilo Castro |

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), buscar\_entre\_anios() es una función que se usa para recorrer la lista completa y filtrar entre dos años que recibe por parámetro. El retorno de esta función se guarda en la variable lista. |
|  | O(n), la variable lista\_filtrada guarda el valor de retorno de la función remover\_value\_lista(). Esta función recibe un departamento por parametro y se encarga de filtrar por departamento, guarda solamente las funciones que contengan $ en su unidad de medida y también ignora los que contienen $ y value (D) |
|  | O(1), guarda en la variable total\_registros el tamaño de la lista\_filtrada calculada anteriormente. |
|  | O(n), se crea un diccionario para poder almacenar los ingresos por año. Se recorre la lista filtrada y se extraen los años para que sean llaves de diccionario. Diccionarios de listas enlazadas. También cuenta con las llaves lista, que almacena las listas enlazadas del año en cuestión, total\_ingresos, que almacena la sumatoria de ingresos, census, que almacena el conteo de registros tipo CENSUS, survey, que almacena el conteo de registros tipo SURVEY, validos, almacena el conteo de registros validos, no\_validos, que almacena el conteo de registros no\_validos.  La inserción en la última posición es O(1) |
|  | Una vez creados los diccionarios, se recorren y se accede a la llave lista. El recorrido de esta llave es O(n), ya que se recorren listas enlazadas por años dentro del diccionario y se hacen los respectivos incrementos al propio diccionario. Como el número de años es fijo, la complejidad completa es O(n), ya que se recorren solamente una vez.  La función es\_numero() O(1), solamente verifica si el valor del registro se puede convertir a flotante, si retorna True, cuenta como registro valido. De lo contrario retorna false y se cuenta como registro no valido. |
|  | *O(n), ya que en el peor de los casos estas comparaciones se harían n veces si tenemos muchos años.* |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos, CALIFORNIA, 1970, 2024.

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 136.391 |
| 200000 datos |  | 432.197 |
| 300000 datos |  | 710.842 |
| 400000 datos |  | 982.640 |
| 500000 datos |  | 1298.784 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

## **Análisis**

El requerimiento 7 presenta un crecimiento lineal. Se recorre al inicio dos veces para cumplir ciertos filtros, se recorre otra vez para crear diccionarios de listas enlazadas y se recorren al final para hacer las sumatorias y comparaciones necesarias para el retorno.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

# **Requerimiento 8**

## **Descripción**

El punto del requerimiento es conocer el cuál es el departamento con mayor diferencia promedio de tiempo entre la recopilación de los registros y su carga a la plataforma de entre todos los registros. Para este requerimiento decidimos crear un diccionario de listas enlazadas en donde la llave del diccionario representa un departamento y la lista enlazada contiene la información necesaria para saber la mayor diferencia. Para el cálculo de diferencias hicimos uso de la librería Datetime.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | ret = (estado, promedio, año\_max, año\_min,diferencia\_max,  diferencia\_min, census, survey) |
| **Implementado (Sí/No)** | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| **Texto  El contenido generado por IA puede ser incorrecto.** | O(n), filtrar los registros para evitar añadir los valores que contengan “$” y (“D”). Se recorre la lista completa y se retorna la lista filtrada. |
| Texto  El contenido generado por IA puede ser incorrecto.  Texto  El contenido generado por IA puede ser incorrecto. | O(n), recorrer la lista para así crear un diccionario donde las llaves son cada departamento. Si no existe, crea la llave y crea una nueva Single Linked List con sl.new\_list(). En caso de que existe, crea un nodo con información,  O(1) para añadirlo al final en la Single Linked List con sl.add\_last(). |
| Interfaz de usuario gráfica, Texto, Chat o mensaje de texto  El contenido generado por IA puede ser incorrecto. | O(N), Se accede a las llaves del diccionario y se recorre la lista enlazada solamente una vez por estado hasta el final para realizar las comparaciones y sumatorias necesarias. |
|  |  |
| ***TOTAL*** | ***O(n)*** |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), filtrar los registros para evitar añadir los valores que contengan “$” y (“D”) |
|  | O(n), recorrer la lista para así crear un diccionario donde las llaves son cada departamento. Si no existe, crea la llave y crea una nueva Single Linked List con sl.new\_list(). En caso de que existe, crea un nodo con información, para añadirlo en la Single Linked List con sl.add\_last(). |
|  | O(k), el diccionario tiene un numero de estados fijo. Por lo tanto, se recorre k veces, siendo k una constante. |
|  | O(N), recorre la lista enlazada hasta el final para realizar las comparaciones y sumatorias necesarias. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |  |
| --- | --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |  |
| 100000 datos |  | 1150.047 |  |
| 200000 datos |  | 2524.231 |  |
| 300000 datos |  | 3260.040 |  |
| 400000 datos |  | 4533.304 |  |
| 500000 datos |  | 5719.113 |  |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| Procesadores | 12th Gen Intel(R) Core(TM) i5-12450H |
| Memoria RAM | 16 GB |
| Sistema Operativo | Windows 11 |

## **Análisis**

El requerimiento 8 presenta un crecimiento lineal. Simplemente se recorre por completo una primera vez para filtrar los datos, una segunda para crear diccionarios de listas enlazadas y luego se recorre el diccionario completo para poder hacer los cálculos necesarios.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

Por la cantidad de variables, comparaciones y sumatorias que se deben realizar aumenta su tiempo de ejecución, comparado a otros requerimientos.