**ANÁLISIS DEL RETO**

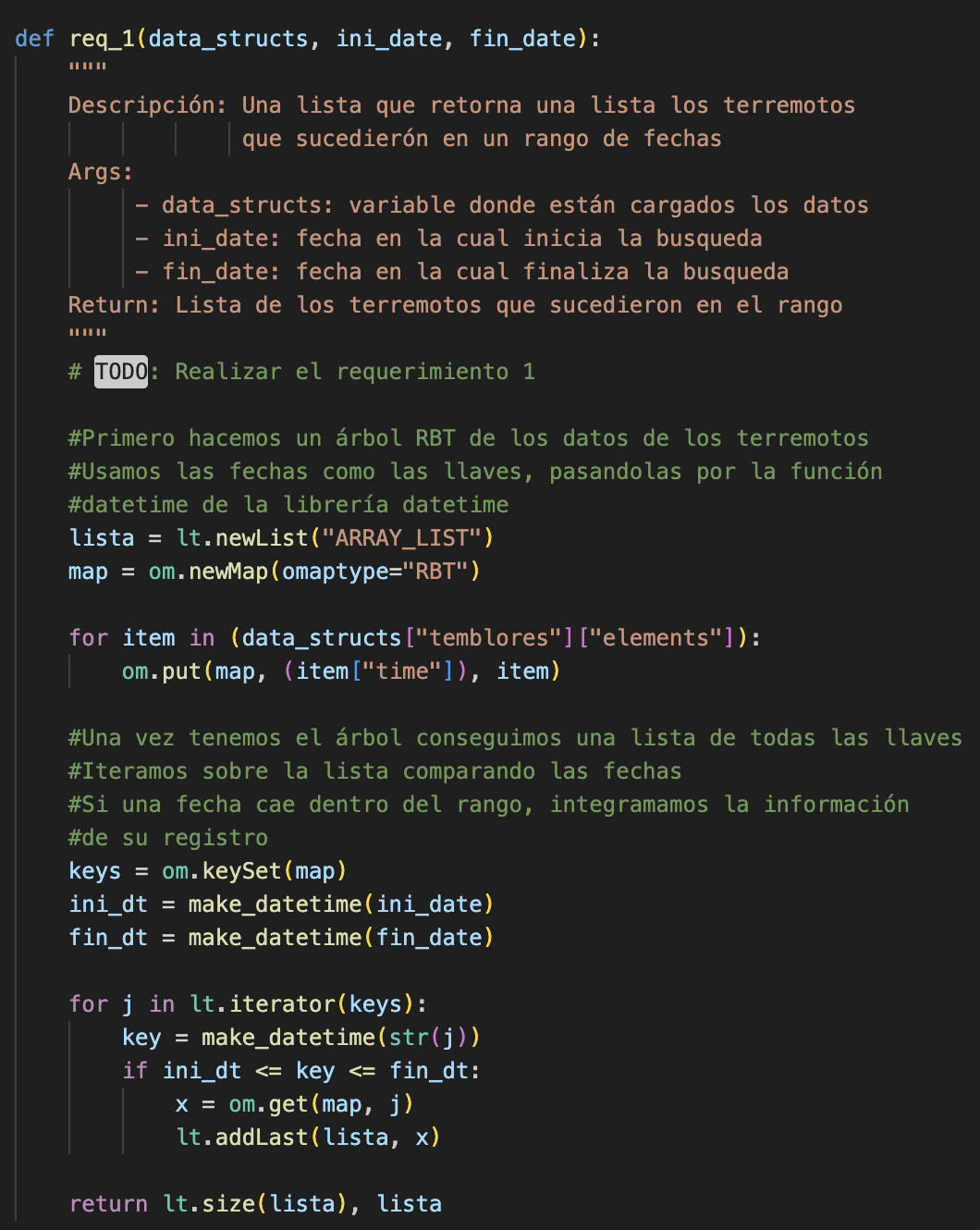
Andrés Ignacio Tello umerez, 202223807, [a.tellou@uniandes.edu.co](mailto:a.tellou@uniandes.edu.co)

*Emile Bottagisio, 202223504*, e.bottagisio@uniandes.edu.co

Juan Esteban Guzmán, *202313334*, je.guzman2@uniandes.edu.co

# **Requerimiento <<1>>**

## **Descripción**



Este requerimiento se encarga de retornar una lista de terremotos en un rango de fechas. Lo primero que hacemos es hacer un arbol RBT y una lista, la lista sera lo que retornaremos y el arbol es nuestra estructura de datos. En la función sacamos una lista de las llaves del arbol que son las fechas de los terremotos. Despues recorremos la lista consiguiendo los registros que caen entre las dos fechas. Al final retorna la lista con los registros y el temaño de la lista que corresponde a la cantidad de registros.

|  |  |
| --- | --- |
| **Entrada** | * Control * Fecha inicial * Fecha final |
| **Salidas** | * Tamaño de la lista * Lista con datos |
| **Implementado (Sí/No)** | Si se implemento, lo implemento Andrés Tello |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 (Hacer la estructura de datos) | O(n) (n=cantidad de registros) |
| Paso 2 (Iterar por la lista) | O(n) |
| ***TOTAL*** | ***O(n2)*** |

## **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).

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
|  |  |
|  |  |
|  |  |

### **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 <<n>>**

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

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | Parámetros necesarios para resolver el requerimiento. |
| **Salidas** | Respuesta esperada del algoritmo. |
| **Implementado (Sí/No)** | Si se implementó y quien lo hizo. |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 | O(...) |
| Paso 2 | O(...) |
| Paso …. | O(...) |
| ***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).

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
|  |  |
|  |  |
|  |  |

### **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**

A screen shot of a computer program

Description automatically generated

Esta función se encarga de retornar los terremotos que cumplen con un minimo de significancia y un maximo de distancia azimutal. La funcion primero hace un arbol tipo RBT que usa sig como la llave. Con la estructura de datos, la función consigue la llave mas grande en el arbol y usa eso y los parametros dados por el usuario para conseguir los registros que tienen sig más alto que el parametro y menor que la mayor llave. Al final, hace un MergeSort y reemplaza todos los valores vacios por “Unknown”. Finalmente, se retorna el tamaño de la lista y la lista como tal.

|  |  |
| --- | --- |
| **Entrada** | * Control * La significancia minima * La distancia azimutal maxima |
| **Salidas** | * Tamaño de la lista * Lista con los datos |
| **Implementado (Sí/No)** | Si se implento, lo implemento Andrés Tello |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 (Hacer la estructura) | O(n) |
| Paso 2 (Organizar la lista) | O(Log(n)\*n) |
| Paso 3 (Iterar por la lista de llaves) | O(n) |
| ***TOTAL*** | ***O(Log(n)\*n3)*** |

## **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).

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
|  |  |
|  |  |
|  |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

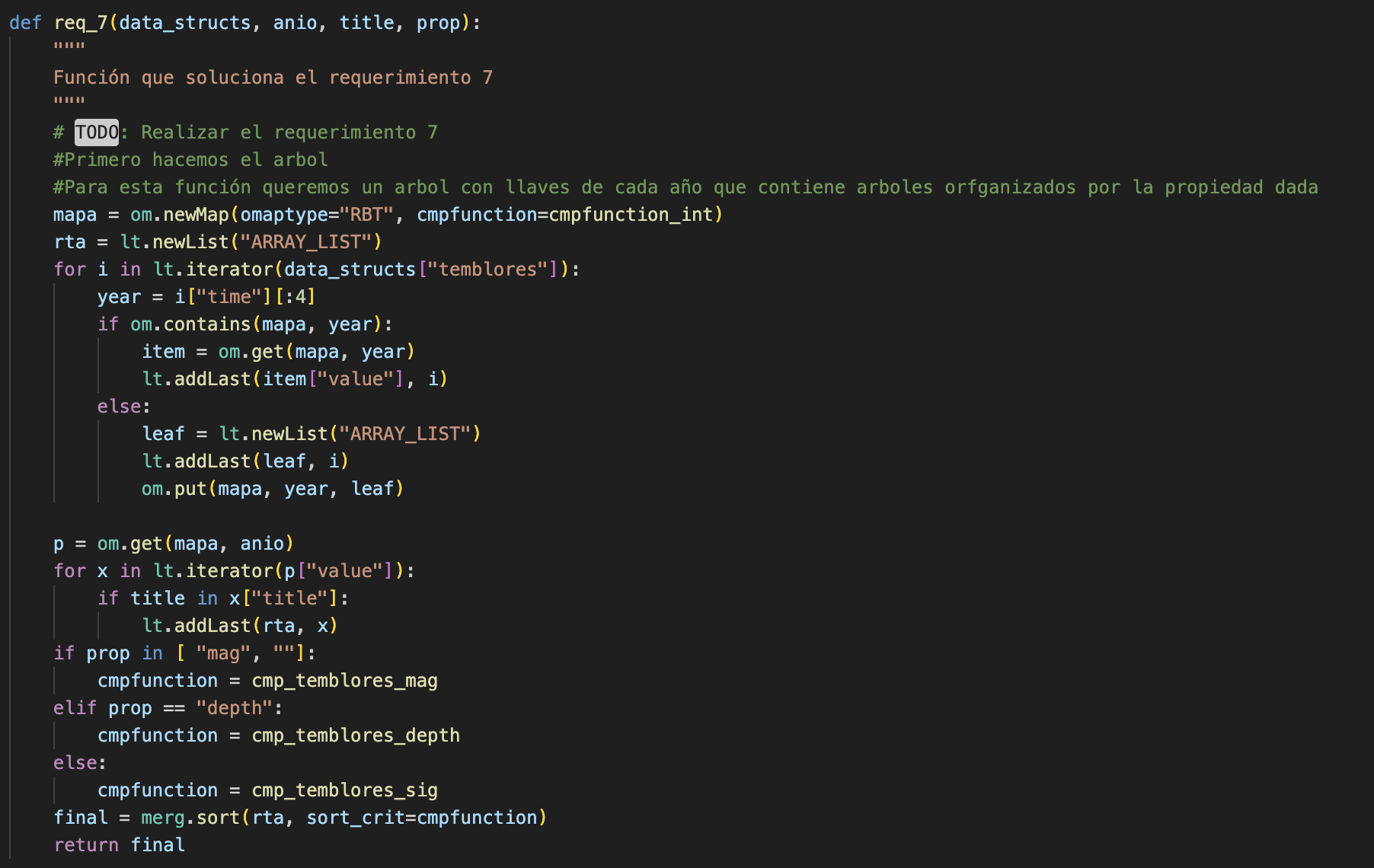
## **Análisis**

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

# **Requerimiento <<7>>**

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

## **Descripción**



Esta función se encarga de conseguir los terremotos de una región en un año y hacer un histograma de la magnitud, la significancia o la profundidad. Primero hacemos un arbol de los años en el cual el valor de cada año es una lista con todos los terremotos de ese año. Luego, iteramos en la lista del año deseado consiguiendo los datos que corresponden a la región. Finalmente, hacemos MergeSort de acuerdo a la propiedad que quiera visualizar el usuario.

|  |  |
| --- | --- |
| **Entrada** | * Control * Año * Región * Propiedad |
| **Salidas** | * Una lista con todos los terremotos de la regíon y del año |
| **Implementado (Sí/No)** | Si se implemento, fue implementado por Andrés Tello |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1 (Hacer la estructura) | O(n) |
| Paso 2 (Iterar en la lista del año) | O(m) (Donde m < n) |
| Paso 3 (MergeSort) | O(Log(n)\*n |
| ***TOTAL*** | ***O(Log(n)\*n2\*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).

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
|  |  |
|  |  |
|  |  |

### **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 Ejemplo**

## **Descripción**



Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

|  |  |
| --- | --- |
| **Entrada** | Estructuras de datos del modelo, ID. |
| **Salidas** | El elemento con el ID dado, si no existe se retorna None |
| **Implementado (Sí/No)** | Si. Implementado por Juan Andrés Ariza |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Buscar si el elemento existe (isPresent) | O(n) |
| Obtener el elemento (getElement) | O(1) |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 4800HS with Radeon Graphics |
| Memoria RAM | 8 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (ms)** |
| small | 0.05 |
| 5 pct | 0.33 |
| 10 pct | 1.28 |
| 20 pct | 2.54 |
| 30 pct | 4.98 |
| 50 pct | 7.51 |
| 80 pct | 13.81 |
| large | 25.97 |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| small | Dato1 | 0.05 |
| 5 pct | Dato2 | 0.33 |
| 10 pct | Dato3 | 1.28 |
| 20 pct | Dato4 | 2.54 |
| 30 pct | Dato5 | 4.98 |
| 50 pct | Dato6 | 7.51 |
| 80 pct | Dato7 | 13.81 |
| large | Dato8 | 25.97 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.