ANÁLISIS DEL RETO

Sofía Acosta Muriel, s.acosta112@uniandes.edu.co, 202211999.

Andrés Julián Bolívar Castañeda, a.bolivarc@uniandes.edu.co, 202214834.

Ariadna del Mar Soto Parada, ad.soto@uniandes.edu.co, 201818217.

Carga de datos

Descripción

En la función principal de carga de datos, se crean las siguientes estructuras:

- ***Temblores:*** Es una ARRAY_LIST con los registros de todos los temblores de la base de datos.
- ***DateIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una fecha (hay dos versiones del árbol, una tiene en cuenta los minutos y segundos y la otra no), y su valor contiene lo siguiente:
  - ***Lstemblores***: Es una lista ARRAY_LIST con los temblores ocurridos en la fecha.
- ***DateHash:*** Es una tabla de Hash, en la cual la llave de cada entrada es una fecha, y su valor contiene lo siguiente:
  - ***Lstemblores***: Es una lista ARRAY_LIST con los temblores ocurridos en la fecha.
- ***MagIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una magnitud aproximada a tres cifras decimales, y su valor contiene lo siguiente:
  - ***Lstemblores***: Es una lista ARRAY_LIST con los temblores ocurridos en la fecha.
- ***SigIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una significancia, y su valor contiene lo siguiente
  - ***GapIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una distancia azimutal y su valor es una lista SINGLE_LINKED con los temblores con esa distancia azimutal.
  - ***Lstemblores***: Es una lista ARRAY_LIST con los temblores con la significancia.
- ***DepthIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una profundidad, y su valor contiene lo siguiente:
  - ***NstIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es un número de estaciones y su valor es una lista SINGLE_LINKED con los temblores que fueron reportados por ese número de estaciones.
  - ***Lstemblores***: Es una lista ARRAY_LIST con los temblores con la profundidad.
- ***TimeIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una fecha, y su valor contiene lo siguiente:
  - ***DepthIndex:*** Es un árbol binario RBT, en el cual la llave de cada entrada es una profundidad y su valor es una lista SINGLE_LINKED con los temblores con esa profundidad.
  - ***Lstemblores***: Es una lista ARRAY_LIST con los temblores reportados en la fecha.

Diagrama

A continuación, se muestra la representación gráfica de la estructura general diseñada para resolver los requerimientos del reto.[1]

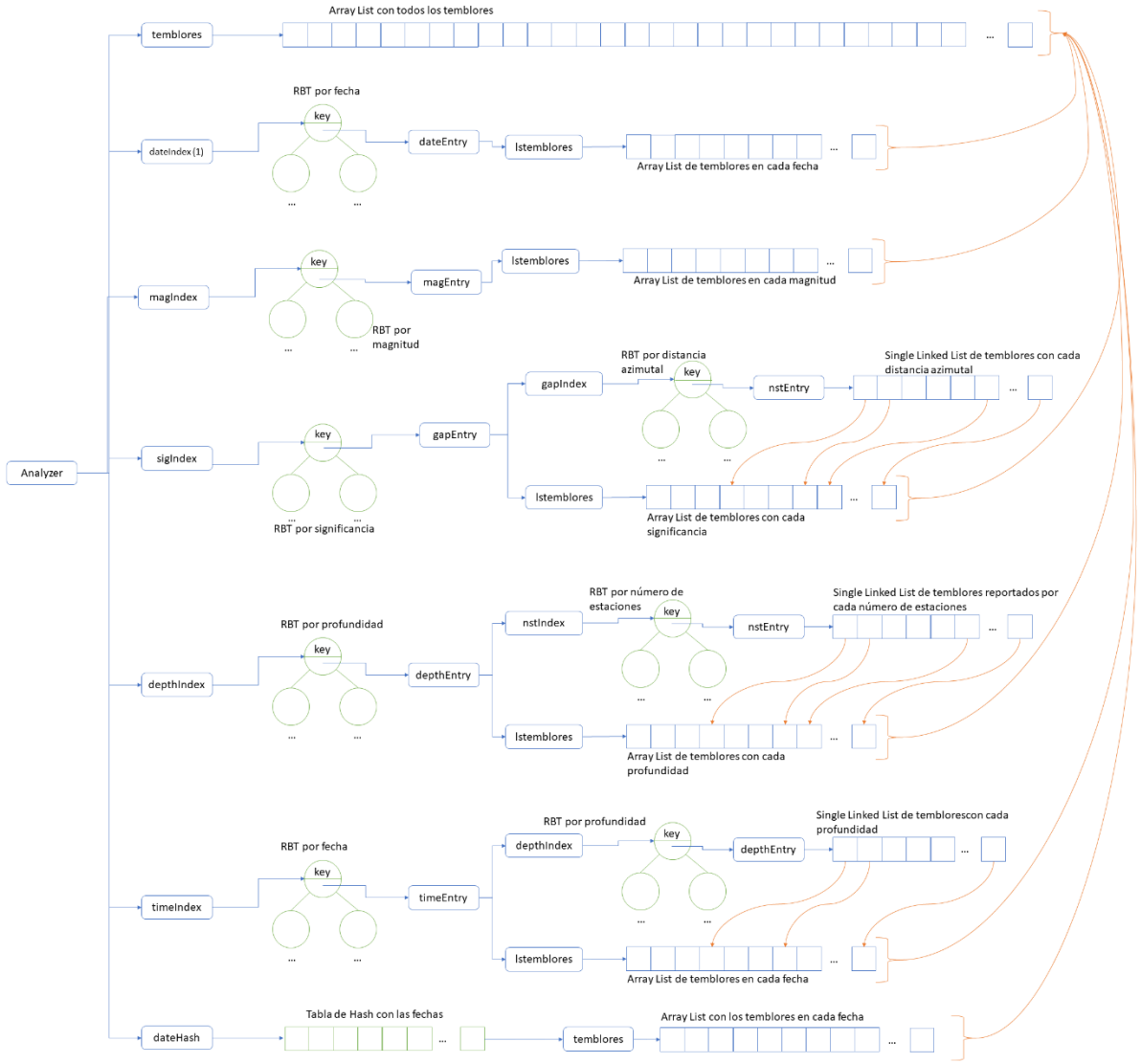


Figura I.1: Estructura de datos general para el reto 3

[1] El diagrama se puede apreciar mejor en el PDF anexo en el repositorio.

Pruebas Realizadas

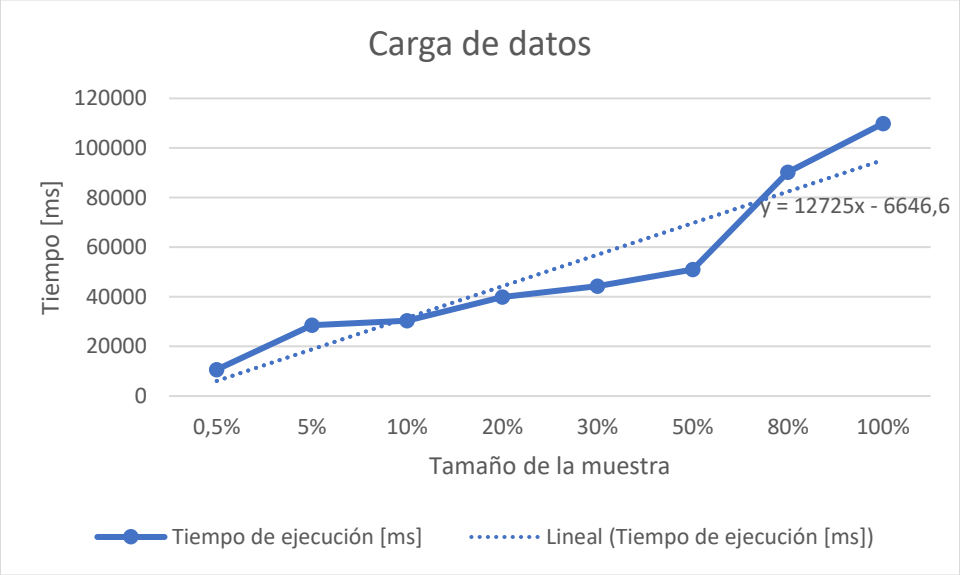
Se realizaron las pruebas con cada tamaño en una máquina con procesador AMD Ryzen 5 4800HS with Radeon Graphics, memoria RAM de 8.0 GB y sistema operativo Windows 11.

Entrada	Tiempo de ejecución [ms]
small	10675.7975
5pct	28507.159
10pct	30438.0052
20pct	39925.4215
30pct	44354.3668
50pct	50966.0751
80pct	90176.2999
large	109901.8588

Tabla I.2: Pruebas de tiempos de la carga de datos

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica I.3: Tiempos de las pruebas de la carga de datos

Requerimiento 1

Descripción

Para la implementación de este requerimiento se hizo uso de la estructura de datos implementada en la carga de datos llamada "dateIndex", que un árbol binario ordenado RBT donde las llaves son las fechas de ocurrencia de los eventos. De este modo, se hizo uso de la función "om.values" para extraer todos los sismos que sucedieron entre las fechas pasadas por parámetro. Finalmente, se recorre la lista de los datos ordenados para poder determinar el tamaño de la respuesta obtenida por el algoritmo.

Entrada	La estructura de datos, la fecha inicial de interés, la fecha final de interés.
Salidas	Todos los eventos ocurridos entre las fechas pasadas por parámetro.
Implementado (Sí/No)	Fue implementado por Andrés Julián Bolívar Castañeda.

Tabla 1.1: Descripción del requerimiento 1

Análisis de complejidad

La implementación de este requerimiento se basa en 2 operaciones principales. La primera operación es la obtención de las llaves con los eventos sísmicos que ocurrieron en el intervalo de tiempo de interés por medio de la operación "om.values", con una complejidad $O(\log n)$, donde n representa la cantidad de fechas que entran en el intervalo de interés. La última operación es el cálculo del tamaño de la respuesta, con una complejidad $O(\log n)$. A continuación, en la Tabla 1.2 se presenta el análisis de la complejidad.

Pasos	**Complejidad**
Obtención de los datos en el intervalo de interés ***"om.values(…)"***	$O(\log n)$
Ciclo ***for*** para el cálculo del tamaño de la respuesta	$O(\log n)$
TOTAL	$\boldsymbol{O(\log n)}$

Tabla 1.2: Análisis de complejidad para el requerimiento 1

Pruebas Realizadas

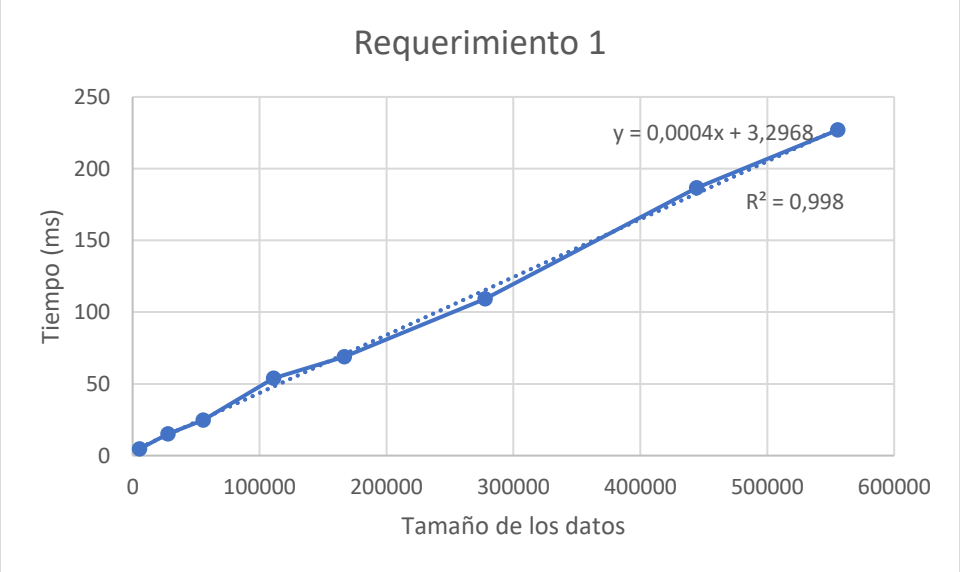
Se realizaron todas las pruebas en una máquina con procesador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, memoria RAM de 8.0 GB y sistema operativo Windows 10. Los parámetros de entrada fueron *"1999-03-21T05:00"* como fecha inicial y *"2004-10-23T17:30"* como fecha final, obteniendo, para el tamaño small, el resultado del ejemplo en el enunciado.

Entrada	Tamaño	Tiempo (ms)
small	5555	4,71
5pct	27779	15,21
10pct	55559	24,62
20pct	111119	54,02
30pct	166679	68,84
50pct	277794	109,18
80pct	444468	186,68
large	555586	226,94

Tabla 1.3: Pruebas de tiempos del requerimiento 1

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 1.4: Tiempos de las pruebas del requerimiento 1

Análisis

Como se evidencia en la gráfica 1.4 junto con la ecuación del gráfico, el comportamiento del algoritmo que soluciona el requerimiento 1 se ajusta a una función con ecuación lineal en donde la pendiente de la recta es casi 0, es decir, el comportamiento del algoritmo se acomoda entre un comportamiento lineal y un comportamiento constante. Dicho comportamiento que se ajusta entre la función lineal y constante puede ser la función logarítmica, lo cual sustenta que la complejidad temporal de este algoritmo es $O(\log n)$. Además, también se evidencia que, en el mejor de los casos, el algoritmo tiene a comportarse como un algoritmo constante.

Requerimiento 2

Descripción

```
def req_2(data_structs, magMin, magMax):
    """
    Función que soluciona el requerimiento 2
    """
    magMin = round(float(magMin),3)
    magMax = round(float(magMax),3)
    magIndex = data_structs['magIndex']
    lst = om.values(magIndex,magMin,magMax)
    size = 0
    for mag in lt.iterator(lst):
        size += lt.size(mag['lstemblores'])
    return lst, size
```

Figura 2.1: Función principal – Requerimiento 2

Para solucionar este requerimiento se hizo uso de la estructura de datos implementada en la carga de datos "magIndex" que es un árbol RBT en el cual las llaves son las magnitudes de los eventos aproximadas a una cifra decimal y los valores son listas con los registros.

Entrada	Catálogo con las estructuras de datos, límite superior de magnitud, límite inferior de magnitud.
Salidas	Lista de los registros en el rango de magnitud, cantidad de elementos obtenidos para la consulta.
Implementado (Sí/No)	Sí. Implementación grupal.

Tabla 2.2: Descripción del requerimiento 2

Análisis de complejidad

Pasos	**Complejidad**
Ajustar los formatos de las entradas y obtener el índice por magnitud	$O(1)$
Obtener la lista de los registros en las magnitudes, para m valores únicos de magnitud en el árbol	$O(\log(m))$
Recorrer cada entrada en el rango de magnitud para obtener el número de datos	$O(m)$
TOTAL	$\boldsymbol{O(m)}$

Tabla 2.3: Análisis de complejidad para el requerimiento 2

Pruebas Realizadas

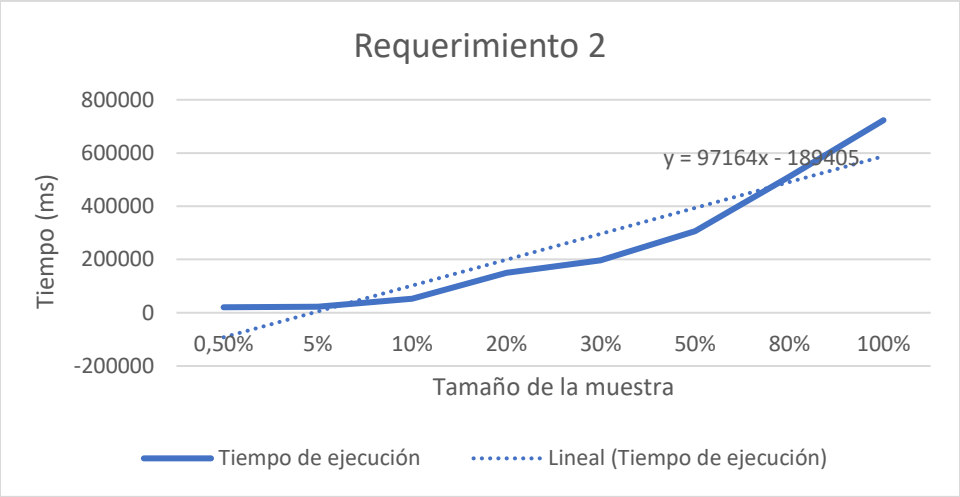
Se realizaron todas las pruebas en una máquina con procesador AMD Ryzen 5 4800HS with Radeon Graphics, memoria RAM de 8.0 GB y sistema operativo Windows 11. Los parámetros de entrada fueron "3.5" como magnitud mínima y "6.5" como magnitud máxima, obteniendo, para el tamaño small, el resultado del ejemplo en el enunciado.

Entrada	Tiempo (s)
small	20534.4745
5pct	22832.4259
10pct	52770.3717
20pct	150495.0622
30pct	196355.9862
50pct	306355.9862
80pct	510089.4505
large	723230.6544

Tabla 2.4: Pruebas de tiempos del requerimiento 2

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 2.5: Pruebas de tiempos del requerimiento 2

Análisis

Como se puede ver en la gráfica, la complejidad temporal de la función se puede modelar como una función potencia, con el exponente entre 1 y 2, dado que no está muy dispersa con respecto a la aproximación lineal, pero sí se nota su tendencia a aumentar más. Esto concuerda con el análisis de complejidad realizado, en el cual se obtuvo una complejidad de $\boldsymbol{O}(m_e \cdot s_m)$.

Requerimiento 3

Descripción

```python
def req_3(data_structs, magMin, depthMax):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    magIndex = data_structs['magIndex']
    fechas = lt.newList('ARRAY_LIST')

    resultado = {"resul": lt.newList("SINGLE_LINKED"),
                 "total": 0}
    magMax = om.maxKey(magIndex)
    magni = om.keys(magIndex, magMin, magMax) #lista con magnitudes de interes


    for magnitud in lt.iterator(magni):
        entry = om.get(magIndex, magnitud)
        entryTemb = me.getValue(entry)
        temblores = entryTemb["lstemblores"]
        for temb in  lt.iterator(temblores):
            if float(temb["depth"])<= float(depthMax):
                lt.addLast(fechas, temb)
    sortReq2(fechas)

    tamaño = lt.size(fechas)

    if tamaño <= 10:
        prim_temblores = lt.subList(fechas,1,tamaño)
    else:
        prim_temblores = lt.subList(fechas,1,10)

    tama = lt.size(prim_temblores)
    if tama <= 6:
        prim_tres_temblores = lt.subList(prim_temblores,1,tama)
        ulti_tres_temblores = lt.newList()
    else:
        prim_tres_temblores = lt.subList(prim_temblores,1,3)
        ulti_tres_temblores = lt.subList(prim_temblores,tama-2,3)

    lt.addLast(resultado['resul'], prim_tres_temblores)
    lt.addLast(resultado['resul'], ulti_tres_temblores)
    resultado['total'] = tamaño
    return resultado
```

Figura 2.1: Función principal – Requerimiento 2

Para solucionar este requerimiento se hizo uso de la estructura de datos implementada en la carga de datos “magIndex” que es un árbol RBT en el cual las llaves son las magnitudes de los eventos aproximadas a una cifra decimal y los valores son listas con los registros.

Entrada	Catálogo con las estructuras de datos, límite inferior de magnitud, límite superior de profundidad.
Salidas	Lista de los registros en el rango deseado, cantidad de elementos obtenidos para la consulta.
Implementado (Sí/No)	Fue implementado por Ariadna Soto

Tabla 2.2: Descripción del requerimiento 3

Análisis de complejidad

Pasos	**Complejidad**
Obtener el rango de magnitud	O (log(n))
Obtener la lista de los registros en las magnitudes, para valores únicos de magnitud en el árbol	O (n + m)
Filtrar l	O (k)
TOTAL	O (log(n))

Tabla 2.3: Análisis de complejidad para el requerimiento 2

Pruebas Realizadas

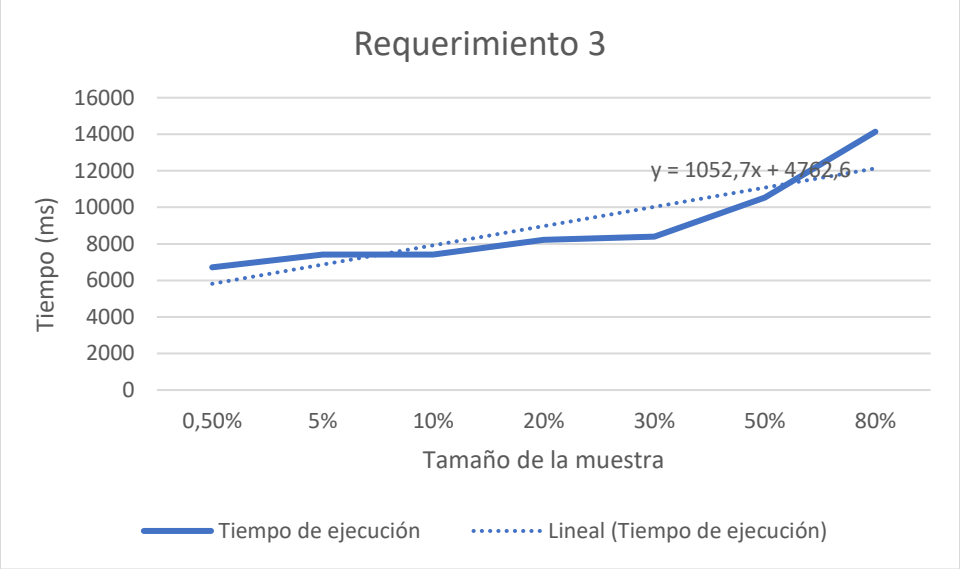
Se realizaron todas las pruebas en una máquina con procesador AMD Ryzen 5 4800HS with Radeon Graphics, memoria RAM de 8.0 GB y sistema operativo Windows 11. Los parámetros de entrada fueron "3.5" como magnitud mínima y "6.5" como magnitud máxima, obteniendo, para el tamaño small, el resultado del ejemplo en el enunciado.

Entrada	Tiempo de ejecución (ms)
small	6414.1717
5pct	6715.6506
10pct	7411.515
20pct	7411.5105
30pct	8221.9764
50pct	8388.3857
80pct	10524.7148
large	14139.6311

Tabla 2.4: Pruebas de tiempos del requerimiento 3

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 2.5: Pruebas de tiempos del requerimiento 3

Análisis

Como se puede ver en la gráfica, la complejidad temporal de la función se puede modelar como una función potencia, con el exponente entre 1 y 2, dado que no está muy dispersa con respecto a la aproximación lineal, pero sí se nota su tendencia a aumentar más. Esto concuerda con el análisis de complejidad realizado, en el cual se obtuvo una complejidad de .

Requerimiento 4

Descripción

```
def req_4(data_structs, sig, gap):
    """
    Función que soluciona el requerimiento 4: Consultar los 15 eventos sísmicos
    más recientes, mayores o iguales a una significancia sig y menores o iguales
    a una distancia azimutal gap.
    """
    sigIndex = data_structs['sigIndex']
    size = 0
    lt_15 = lt.newList()
    sig_values = om.values(sigIndex, sig, om.maxKey(sigIndex)) # Entradas en el rango de significancia
    for sig_info in lt.iterator(sig_values): # Registros en el rango de significancia
        gap_values = om.values(sig_info['gapIndex'],0,gap) # Entradas en el rango de distancia azimutal
        for gap_info in lt.iterator(gap_values): # Registros en el rango de distancia azimutal
            for eq in lt.iterator(gap_info['lstemblores']):
                # Agregar valor la lista y sumar la unidad a la consulta
                lt_15 = update_lt_15(lt_15, eq)
                size += 1
    return lt_15, size
```

Figura 4.1: Función principal – Requerimiento 4

Este requerimiento consiste en consultar los 15 eventos sísmicos más recientes, mayores o iguales a una significancia y menores o iguales a una distancia azimutal dadas. Para ello, se realizó la consulta por significancia y la subconsulta por distancia azimutal, las cuales obtienen los registros en los rangos de significancia y distancia azimutal de los árboles ordenados 'sigIndex' y 'gapIndex', respectivamente, de la estructura de datos general, y los agrega en una lista, por medio de la función *update_lt_15*, que evalúa si el registro debe ir en la lista de respuesta o no, comparando su fecha con la de los registros que ya están en la lista, y tiene una complejidad constante, porque la lista nunca tendrá más de 15 elementos.

Entrada	Catálogo con las estructuras de datos, significancia mínima, distancia azimutal máxima.
Salidas	Lista de los registros en los rangos de significancia y distancia azimutal ordenados por fecha.
Implementado (Sí/No)	Sí. Implementado por Sofía Acosta.

Tabla 4.2: Descripción del requerimiento 4

Análisis de complejidad

Pasos	**Complejidad**
Obtener el índice por significancia y crear una nueva lista encadenada para la respuesta de la consulta (NewList)	$O(1)$
Obtener la lista de las entradas con significancia mayor o igual a la indicada (Values, MaxKey) del árbol por significancia 'sigIndex' con s elementos	$O(\log(s))$
Recorrer la lista de las entradas con las significancias solicitadas (de tamaño s_e) para obtener sus respectivos árboles por distancias azimutales	$O(s_e)$
Obtener las entradas de los temblores con las distancias azimutales solicitadas del árbol por distancia azimutal 'gapIndex' (con g elementos)	$O(\log(g))$
Recorrer la lista de entradas con las distancias azimutales (de tamaño g_e) para obtener la lista de temblores	$O(g_e)$
Recorrer la lista de temblores obtenida (de tamaño g_t)	$O(g_t)$
Si el partido cumple con las condiciones, agregarlo a la lista de respuesta	$O(1)$
TOTAL	$\boldsymbol{O(s_e \cdot g_e \cdot g_t)}$

Tabla 4.3: Complejidad temporal del requerimiento 4

Pruebas Realizadas

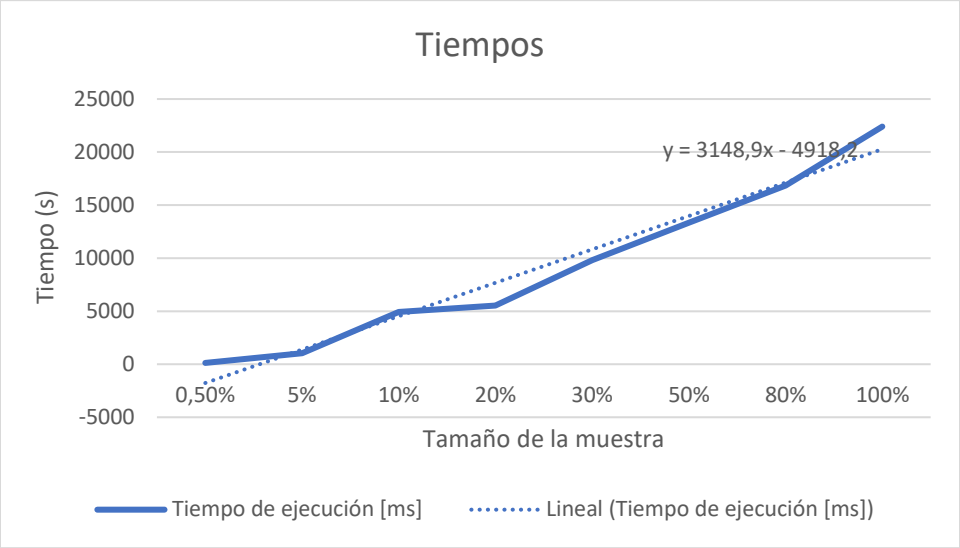
Se realizaron todas las pruebas en una máquina con procesador AMD Ryzen 5 4800HS with Radeon Graphics, memoria RAM de 8.0 GB y sistema operativo Windows 11. Los parámetros de entrada fueron "300" como significancia mínima y "45" como distancia azimutal máxima, obteniendo, para el tamaño small, el resultado del ejemplo en el enunciado.

Entrada	**Tiempo de ejecución [ms]**
small	123.6767
5pct	1046.302237
10pct	4921.239835
20pct	5521.990183
30pct	9813.322202
50pct	13338.32456
80pct	16846.2848
large	22404.7165

Tabla 4.4: Pruebas de tiempos del requerimiento 4

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 4.5: Tiempos de las pruebas del requerimiento 4

Análisis

Como se puede observar en la gráfica, la complejidad temporal de la implementación del requerimiento 4 se puede modelar como una función lineal, ya que los datos obtenidos no están dispersos con respecto a la curva de tendencia. Así, efectivamente, la complejidad es $O(s_e \cdot g_e \cdot g_t)$, donde s_e es la cantidad de significancias en el rango solicitado, g_e es el número de entradas con la distancia azimutal solicitada dentro de cada significancia y g_t es el número de registros que cumplen con ambas condiciones dentro de cada significancia. Ahora, hay que tener en cuenta que $s_e \cdot g_e \cdot g_t$ es ligeramente superior al número de elementos que cumplen con las condiciones de la consulta, por lo que la complejidad sí coincide con una función lineal (un poco mayor), tomando el tamaño de la consulta g_r como referencia. Esta complejidad se justifica debido a que es necesario obtener el número TOTAL de temblores en los rangos de significancia y distancia azimutal, para luego mostrar los primeros 15, por lo que se debe extraer el rango del árbol de significancia y luego del sub-árbol de distancia azimutal.

Requerimiento 5

Descripción

Para este requerimiento, se hizo uso de los índices implementados en el analizador llamados “heap” y “depthIndex”. El primer índice mencionado se utilizó para dar la respuesta principal del requerimiento, donde se piden los 20 primeros eventos ocurridos recientemente que cumplen con una magnitud mínima y un número de estaciones mínimo. En este orden de ideas, el índice “heap” es un heap orientado a menor que permite ir revisando las fechas de forma descendente, es decir, de la más reciente a la más antigua. En cada una de las entradas al heap, se revisa un heap interno basado en la magnitud, para garantizar que se cumpla con la restricción de la magnitud y minimizar las revisiones. Finalmente, dentro de cada entrada de este heap interno se revisa otro heap basado en el número de estaciones que registran el sismo para garantizar el cumplimiento de la segunda restricción. Cabe agregar que este recorrido se detiene en cuanto se encuentren los 20 eventos para la respuesta. El segundo de los índices mencionados se implementa para hacer fácil el cálculo de la cantidad de datos que cumplen la restricción.

Entrada	La estructura de datos, la profundidad mínima del sismo y el número de estaciones mínimo que registraron el evento.
Salidas	Todos los eventos ocurridos que cumplen la condición de la profundidad mínimo y el número de eventos que registran el evento mínimo.
Implementado (Sí/No)	Fue implementado por Andrés Julián Bolívar Castañeda.

Tabla 5.1: Descripción del requerimiento 5

Análisis de complejidad

La solución de este requerimiento se basa en 2 operaciones principales. La primera de ellas consiste en hacer un recorrido sobre el heap basado en las fechas, sobre los heap’s internos de cada fecha basados en la magnitud y sobre los heap’s internos de cada magnitud basados en el número de estaciones que registra el sismo. Dicho recorrido se realiza para revisar cuáles eventos más recientes cumplen las condiciones propuestas por el requerimiento, por lo que se detiene en cuanto encuentra los 20 eventos de interés. De este modo, la complejidad de esta operación es $O(m * k * l)$, en donde m representa la cantidad de fechas distintas que hay en todos los datos, k representa la cantidad de magnitudes distintas de los sismos en cada fecha y l representa la cantidad de eventos que ocurren en una fecha específica con una magnitud específica. Sin embargo, en realidad este recorrido, en el peor de los casos, lo que haría realmente sería recorrer todos los datos cargados, lo que determinaría una complejidad $O(n)$. De la misma manera, en el mejor de los casos, los 20 eventos de interés se encontrarían en las primeras posiciones revisadas, lo que le permite al algoritmo llegar a tener un comportamiento constante, es decir, $O(1)$. En este orden de ideas, la última operación principal es un doble ciclo for anidado que registra el tamaño de la respuesta con una complejidad $O(m * k)$.

Pasos	**Complejidad**
Triple ciclo ***for*** para obtener los datos que también cumplen la condición de número de estaciones mínimo y agregarlos a la lista de la respuesta	$O(m * k * l)/O(n)$ $O(1)$ *en el mejor de los casos*
Doble ciclo ***for*** para determinar el tamaño de la respuesta	$O(m * k)$
TOTAL	$\boldsymbol{O(m * k * l)/O(n)}$

Tabla 5.2: Análisis de complejidad para el requerimiento 5

Pruebas Realizadas

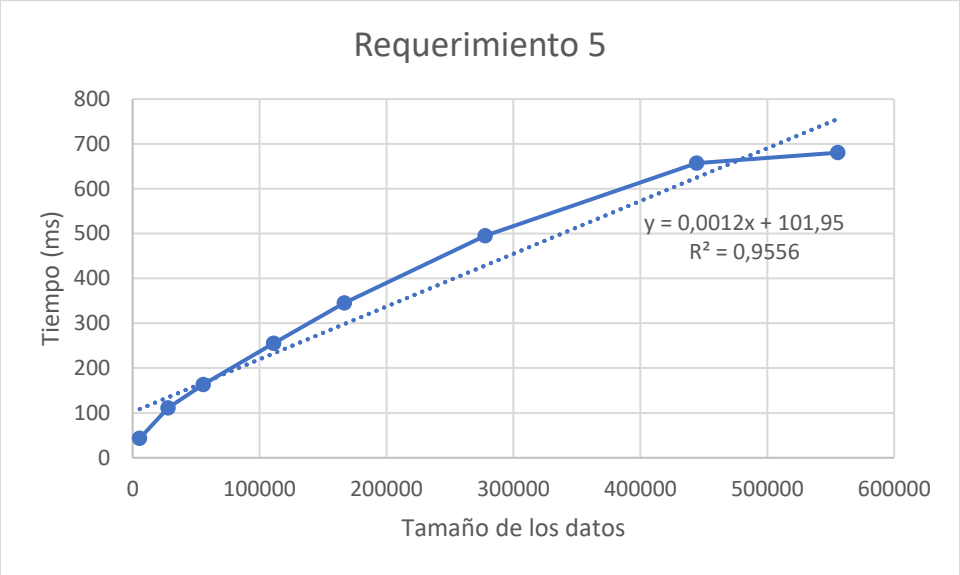
Se realizaron todas las pruebas en una máquina con procesador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, memoria RAM de 8.0 GB y sistema operativo Windows 10. Los parámetros de entrada fueron *"23"* como profundidad mínima y *"38"* como cantidad de estaciones mínima, obteniendo, para el tamaño small, el resultado del ejemplo en el enunciado.

Entrada	**Tamaño**	**Tiempo (ms)**
small	5555	43,05
5pct	27779	111,42
10pct	55559	163,16
20pct	111119	255,31
30pct	166679	345,34
50pct	277794	494,77
80pct	444468	657,21
large	555586	844,31

Tabla 5.3: Pruebas de tiempos del requerimiento 5

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 5.4: Tiempos de las pruebas del requerimiento 5

Análisis

Como se evidencia en la gráfica 5.4 y en su ecuación de gráfico, el comportamiento del algoritmo para el requerimiento 5 se ajusta a un comportamiento lineal, es decir, se determina un comportamiento $O(n)$ según las pruebas. Del mismo modo, la gráfica establece una pendiente de la recta con un valor cercano a 0, es decir, el tiempo de ejecución no dependería del tamaño de los datos, lo cual sustenta que, en un buen caso, en dependencia del tamaño de los datos, el algoritmo podría tener un comportamiento constante, en otras palabras, la complejidad temporal podría llegar a ser $O(1)$.

Requerimiento 6

Descripción

Este requerimiento tiene como objetivo reportar el evento más significativo y los n eventos más próximos cronológicamente ocurridos dentro del área alrededor de un punto. Para su implementación se hizo uso de la estructura de datos de la carga, que es una tabla de Hash o mapa no ordenado donde las llaves son los años de las ocurrencias de los eventos y los valores son listas con los eventos de dicho año. Con base en esto, se obtienen los eventos del año de interés y se realiza un recorrido para determinar los eventos que ocurrieron dentro del radio de interés y establecer el de mayor significancia. Finalmente, dependiendo del tamaño de la respuesta y la ubicación temporal del evento de mayor significancia, se extraen los n eventos antes y después del evento principal.

Entrada	La estructura de datos, el año de interés, la latitud del punto de interés, la longitud del punto de interés, el radio de interés y el número de eventos antes y después del evento principal.
Salidas	El evento de mayor significancia que cumple con los parámetros de entrada, y la lista con los n eventos que ocurrieron antes y después del evento de interés.
Implementado (Sí/No)	Fue implementado por Andrés Julián Bolívar Castañeda.

Tabla 6.1: Descripción del requerimiento 6

Análisis de complejidad

Para solucionar este requerimiento, se realizan 3 tareas principales. La primera de ellas es la obtención de los eventos que ocurrieron en el año de interés, la cual, gracias a la implementación de la tabla de Hash, tiene una complejidad temporal de $O(1)$. En segunda instancia, se realiza un recorrido completo sobre los eventos del año de interés, para determinar los eventos dentro del radio dado por parámetro y el evento de mayor significancia, por lo que se determina una complejidad $O(n)$, donde n representa la cantidad de eventos en el año de interés. Finalmente, gracias a la estructuración de los datos en la carga de datos, para obtener los n eventos antes y después del evento principal, se usa la función "**lt.sublist(…)**", por lo que la complejidad temporal es $O(1)$ en este caso.

Pasos	**Complejidad**
Obtención de los eventos en el año dado	$O(1)$
Ciclo ***for*** para el análisis de las distancias y significancia	$O(n)$
Obtención de los n eventos antes y después del evento principal	$O(1)$
TOTAL	$\boldsymbol{O(n)}$

Tabla 6.2: Análisis de complejidad para el requerimiento 6

Pruebas Realizadas

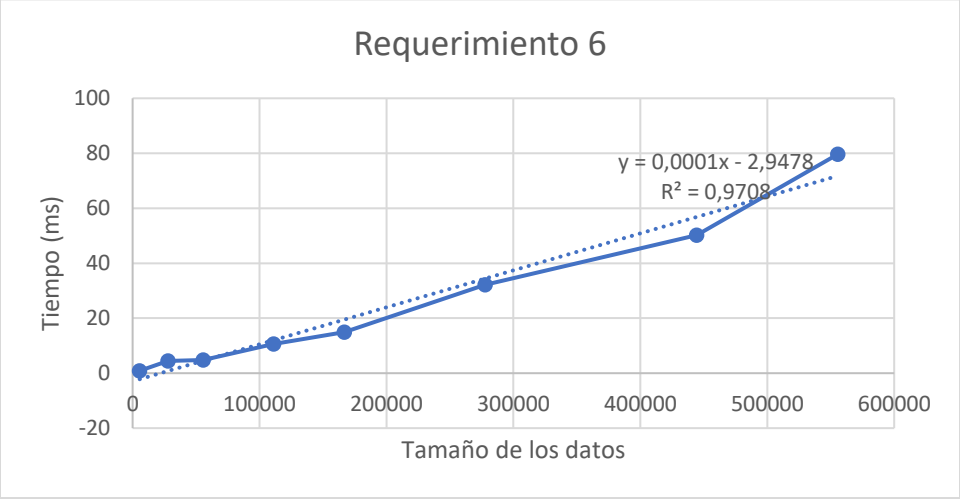
Se realizaron todas las pruebas en una máquina con procesador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, memoria RAM de 8.0 GB y sistema operativo Windows 10. Los parámetros de entrada fueron *"2022"* como año, *"4.674"* como latitud, *"-74.068"* como longitud, *"3000 km"* como radio y *"5"* como número de eventos a retornar, antes y después del evento de mayor significancia.

Entrada	Tamaño	Tiempo (ms)
small	5555	0,88
5pct	27779	4,46
10pct	55559	4,83
20pct	111119	10,61
30pct	166679	14,87
50pct	277794	32,1
80pct	444468	50,2
large	555586	79,65

Tabla 6.3: Pruebas de tiempos del requerimiento 6

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 6.4: Tiempos de las pruebas del requerimiento 5

Análisis

La gráfica 6.4, junto con la ecuación del gráfico, evidencian que el algoritmo que soluciona el requerimiento 6 tiene un comportamiento lineal, es decir, tiene una complejidad temporal $O(n)$ en el peor de los casos, el cual sería que todos los sismos ocurrieran en el año consultado. En este orden de ideas, debido a que la cantidad de sismos que ocurren en un solo año se puede considerar relativamente pequeña en comparación al total de datos, el algoritmo puede aproximarse a un comportamiento constantes, es decir, no depende de la cantidad de datos, tal como se evidencia en la ecuación de la gráfica con una pendiente de casi 0, es decir, una complejidad temporal de $O(1)$.

Requerimiento 7

Descripción

```python
def req_7(data_structs, year, area, prop, bins):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    # Obtener los registros en el año
    st_date = datetime.date(year=year, month=1, day=1)
    end_date = datetime.date(year=year, month=12, day=31)
    eq_year = om.values(data_structs['dateIndex'], st_date, end_date)
    #Obtener los registros en el área
    lt_hist_data = lt.newList(cmpfunction=compareCodes)
    num_eq_year = 0
    num_eq_year_title = 0
    if not lt.isEmpty(eq_year):
        for eq_info in lt.iterator(eq_year):
            eq_lst = eq_info['lstemblores']
            for eq in lt.iterator(eq_lst):
                num_eq_year += 1
                if area.lower() in eq['title'].lower():
                    num_eq_year_title += 1
                    #Agregar los registros que cumplen a la lista
                    lt.addLast(lt_hist_data, eq)
    h_bins, h_values, min_val, max_val = histogram_data_req7(lt_hist_data, prop, bins)
    lt_hist_data = merg.sort(lt_hist_data, compareReq4)
    return num_eq_year, num_eq_year_title, h_bins, h_values, min_val, max_val, lt_hist_data
```

```python
def histogram_data_req7(data, prop, bins):
    data = sortReq7(data, prop)
    # Valores mínimo y máximo
    if not lt.isEmpty(data):
        min_val = lt.firstElement(data)[prop]
        max_val = lt.lastElement(data)[prop]
        # Intervalos
        len_bins = (max_val-min_val)/bins
        h_bins = lt.newList('ARRAY_LIST')
        lim = min_val
        for i in range(bins):
            lt.addLast(h_bins, lim)
            lim += len_bins
        lt.addLast(h_bins, max_val)
        # Valores histograma
        h_values = lt.newList('ARRAY_LIST')
        for eq in lt.iterator(data):
            if not eq[prop] == 'unknown':
                lt.addLast(h_values, eq[prop])
    else:
        h_bins = lt.newList('ARRAY_LIST')
        h_values = lt.newList('ARRAY_LIST')
        min_val = 0
        max_val = 0
    return h_bins, h_values, min_val, max_val
```

Figura 7.1: Funciones principal y auxiliar – Requerimiento 7

Este requerimiento, contabiliza los eventos sísmicos ocurridos en una región y un año especifico según alguna propiedad de interés como lo son su magnitud (mag), profundidad (depth) o la significancia del evento (sig). Para ello, se obtuvieron los registros en el año solicitado del árbol *'dateIndex'* y se evaluó si ocurrieron en el área de consulta. Posteriormente, se extrajeron los valores de la propiedad de interés y, a partir de ellos se creó un histograma con el número de intervalos ingresado (mediante la herramienta para hacer histogramas de la librería Matplotlib, en la vista), obteniendo la información para hacerlo por medio de la función *histogram_data_req7* (cuya complejidad está resaltada en color azul en el análisis a continuación).

Entrada	Catálogo con las estructuras de datos, año de consulta, área de consulta, propiedad de interés y número de intervalos.
Salidas	Información general de la consulta (temblores en el año, temblores en el área, listas con límites de los intervalos y valores a graficar, registro mínimo y máximo) y lista del total de datos de la consulta.
Implementado (Sí/No)	Sí. Implementado por Sofía Acosta.

Tabla 7.2: Descripción del requerimiento 7

Análisis de complejidad

Pasos	**Complejidad**
Actualizar el formato de las fechas ingresadas	$O(1)$
Obtener la lista de fechas en el año del árbol por fechas (de tamaño d, Values)	$O(\log(d))$
Crear una nueva lista single linked para los datos del histograma (NewList)	$O(1)$
Evaluar si hay registros en el año (IsEmpty)	$O(1)$
Recorrer las fechas en la lista de entradas (de tamaño d_f) para obtener la lista de los temblores en cada fecha	$O(d_f)$
Recorrer la lista temblores de cada fecha (de tamaño t_f) para evaluar si ocurrieron en el área de interés	$O(t_f)$
Si cumple con las condiciones, agregar el registro a la lista de información del histograma (AddLast)	$O(1)$
Generar la información necesaria para hacer el histograma	
Ordenar la lista de información del histograma (de tamaño t_h) por la propiedad de interés con MergeSort	$O(t_h \cdot \log(t_h))$
Evaluar si hay registros (IsEmpty)	$O(1)$
Obtener el elemento máximo y mínimo (FirstElement, LastElement) y calcular la longitud del rango	$O(1)$
Crear una lista single linked para guardar los rangos del histograma (NewList)	$O(1)$
Agregar a la lista de rangos los límites según el número de intervalos b (AddLast)	$O(b)$
Crear una lista single linked para guardar los valores de la propiedad relevante del histograma (NewList, AddLast), recorriendo la lista de información del histograma de tamaño t_h	$O(t_h)$
Ordenar la lista del requerimiento (de tamaño t_h) por fechas con MergeSort	$O(t_h \cdot \log(t_h))$
TOTAL	$\boldsymbol{O(t_h \cdot \log(t_h))}$

Tabla 7.3: Análisis de complejidad para el requerimiento 7

Pruebas Realizadas

Se realizaron todas las pruebas en una máquina con procesador AMD Ryzen 5 4800HS with Radeon Graphics, memoria RAM de 8.0 GB y sistema operativo Windows 11. Los parámetros de entrada fueron *"2020"* como año de consulta, *"Alaska"* como área de interés, *"mag"* como propiedad de interés y *"10"* como número de intervalos, obteniendo, para el tamaño small, el resultado del ejemplo en el enunciado. A continuación, se muestra el histograma obtenido:

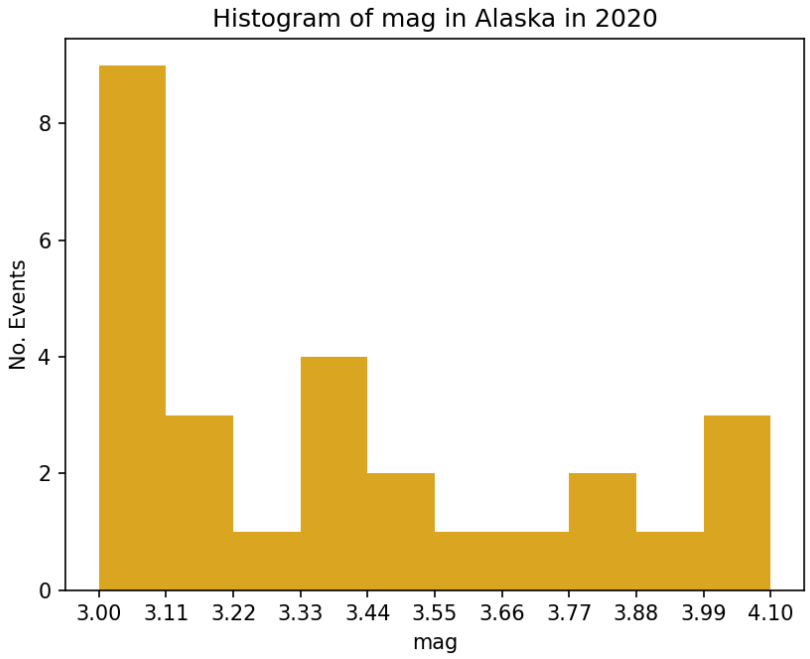


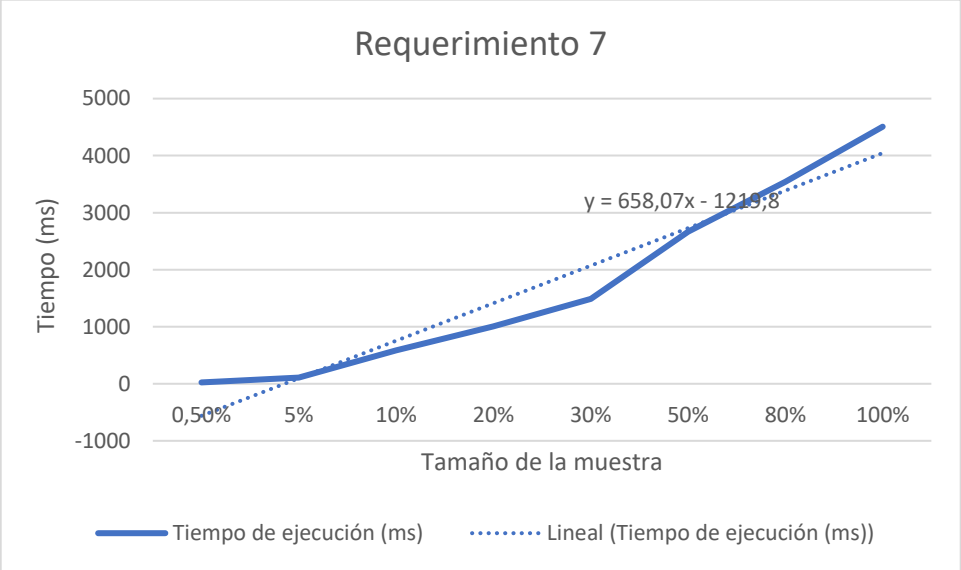
Figura 7.4: Resultado histograma – Prueba requerimiento 7

Entrada	Tiempo de ejecución (ms)
small	23.9679
5pct	108.6926
10pct	586.1242
20pct	1005.9372
30pct	1491.1293
50pct	2667.5324
80pct	3541.6185
large	4507.4073

Tabla 7.5: Pruebas de tiempos del requerimiento 7

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 7.6: Tiempos de las pruebas del requerimiento 7

Análisis

Como se puede observar en la gráfica, la complejidad temporal de la implementación del requerimiento 7 la complejidad se puede modelar como una función lineal-logarítmica, ya que los datos obtenidos no están dispersos con respecto a la curva de tendencia, aunque sí se nota que son superiores. Así, efectivamente, la complejidad es $O(t_h \cdot \log(t_h))$, donde t_h es el total de temblores en el año y en la región indicadas. Esto se debe a que el tamaño de t_h es casi igual a $d_f \cdot t_f$, donde d_f es el número de fechas en el año y t_f es el número de temblores en cada fecha, por lo que, a pesar del doble ciclo for, la complejidad no alcanza a ser mucho mayor que una lineal (tomando t_h como referencia). Ahora, también es necesario ordenar dos veces la lista de temblores (lo cual justifica la complejidad lineal-logarítmica) por la propiedad de interés, para obtener el valor mínimo y máximo, y otra por fecha, para el resultado que se debe mostrar.

Requerimiento 8

Descripción

```
def req_8(data_structs, req, parametros):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    if req == 1:
        return req_1(data_structs, lt.getElement(parametros,1),lt.getElement(parametros,2))
    elif req == 4:
        r = req_4(data_structs,lt.getElement(parametros,1),lt.getElement(parametros,2))['heap']['elements']
        if lt.size(r) < 15:
            return r
        else:
            return lt.subList(r,1,15)
    elif req == 5:
        return req_5(data_structs, lt.getElement(parametros,1),lt.getElement(parametros,2))
    elif req == 7:
        return req_7(data_structs, lt.getElement(parametros,1),'Alaska',lt.getElement(parametros,2), 1)
```

Figura 8.1: Función principal – Requerimiento 8

Este requerimiento muestra los requerimientos 1, 2, 4, 5, 6 y 7 gráficamente en un mapa, usando la librería folium. Para ello, se llama a los requerimientos ya diseñados y se muestran los resultados en un mapa, lo cual se realiza desde la vista.

Entrada	Parámetros del requerimiento deseado.
Salidas	Mapa con los resultados de la consulta
Implementado (Sí/No)	Sí. Implementado por Andrés Bolívar

Tabla 8.2: Descripción del requerimiento 7

Análisis de complejidad

Este requerimiento tiene una complejidad igual a la del requerimiento de mayor complejidad, pero sumado con un factor igual al tamaño de la lista de mayor tamaño obtenida en una consulta, ya que se debe iterar para obtener los resultados a imprimir en el mapa.

Pruebas Realizadas

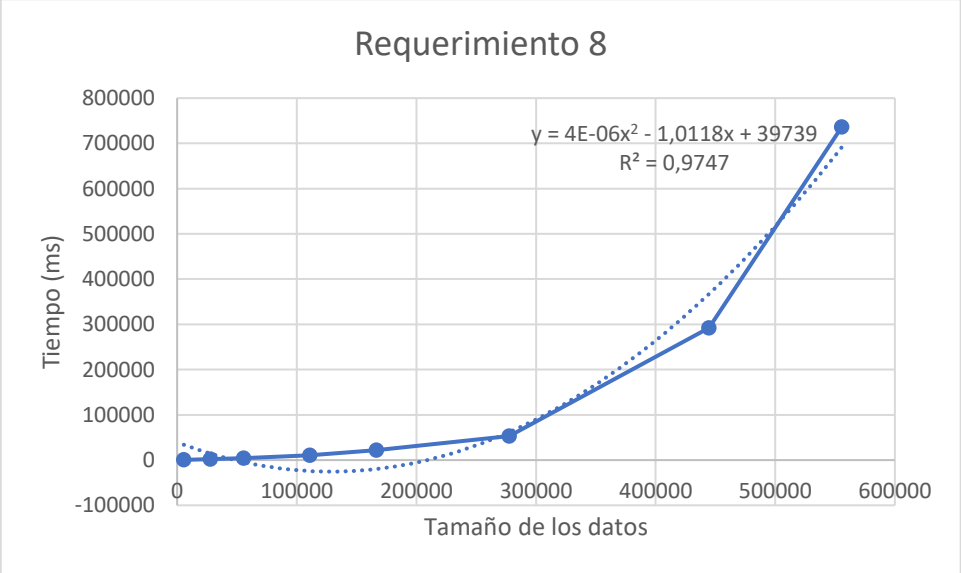
Se realizaron todas las pruebas en una máquina con procesador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, memoria RAM de 8.0 GB y sistema operativo Windows 10. Los parámetros de entrada fueron los mismos que se pasaron en las pruebas de cada uno de los requerimientos anteriores:

Entrada	Tamaño	Tiempo (ms)
small	5555	429,64
5pct	27779	1865,89
10pct	55559	4572,07
20pct	111119	10535,59
30pct	166679	22245,34
50pct	277794	52983,72
80pct	444468	292264,05
large	555586	736147,69

Tabla 8.3: Pruebas de tiempos del requerimiento 8

Gráficas

La gráfica compara los tiempos obtenidos con su estimación a una función lineal.



Gráfica 8.4: Tiempos de las pruebas del requerimiento 8

Análisis

Como se puede apreciar en la gráfica 8.4, junto con la ecuación del gráfico, el algoritmo del requerimiento 8 se ajusta a un comportamiento cuadrático, según las pruebas realizadas, es decir, tiene una complejidad temporal $O(n^2)$. Este componente cuadrático puede estar dado por una estimación de la suma de la complejidad de los anteriores requerimientos con el tamaño de las repuestas que se brindan. Sin embargo, el coeficiente que acompaña a la variable cuadrática es casi 0, por lo que en ciertos caso

(ingresando otros parámetros) puede que el algoritmo se comporte de manera lineal, lo cual indicaría que la complejidad temporal de la solución del requerimiento está dada por el tamaño de las respuestas que dan los requerimientos, en otras palabras, puede comportarse con una complejidad laboral $O(n)$.