1ANÁLISIS DEL RETO

Estudiante 1 - Jessica Sofía Garay Acosta, 202310514, js.garay@uniandes.edu.co

Estudiante 2 - María Lucía Benavides, 202313423, m.benavidesd@uniandes.edu.co

Estudiante 3 - Daniel Mancilla Triviño, 202221038, d.mancilla@uniandes.edu.co

Requerimiento <<1>>

Descripción

Función para obtener los eventos sísmicos mundiales entre 2 fechas

```
def req_1(catalog, fecha_i, fecha_f):
    """
    Función que soluciona el requerimiento 1
    """
    by_date = catalog['dates_simple']
    list_val = om.valuesList(by_date, fecha_i, fecha_f )
    total = om.size(list_val)
    height = om.height(by_date)
    totales = (total, height)
    return list_val, totales
```

La función para la ejecución del primer requerimiento utiliza el mapa de la carga de datos el cual se llama dates_simple y cuenta con todas las fechas que se encuentran dentro del registro. Luego con este mapa utiliza la función de valuesList, la cual toma los valores que se encuentran dentro del intervalo otrorgado y devuelve una estructura tipo array_list con todos los sismos. Luego para hallar el tamaño y la altura de el mapa de las fechas se usan las funciones om.size y om.height y luego se almacenan en una tupla, la cual se devuelve con la lista.

Entrada	Fecha inicio del intervalo Fecha final del intervalo
Salidas	Total de eventos ocurridos entre las fechas ordenados cronológicamente
Implementado (Sí/No)	Si, grupal

Análisis de complejidad

Pasos	**Complejidad**
Acceder a mapa	O(1)

Om.valuesList()	O(logN)
Om.size()	O(1)
Om.height()	O(1)
TOTAL	***O(logN)***

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).

Entradas: 1999-03-21T05:00, 2004-10-23T17:30

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHZ**
Memoria RAM	8 GB
Sistema Operativo	Microsoft Windows 11 Home Single Language

Entrada	**Tiempo (ms)**
small	4.38
5 pct	20.66
10 pct	43.35
20 pct	88.88
30 pct	148.64
50 pct	200.36
80 pct	321.25
large	405.82

Tablas de datos

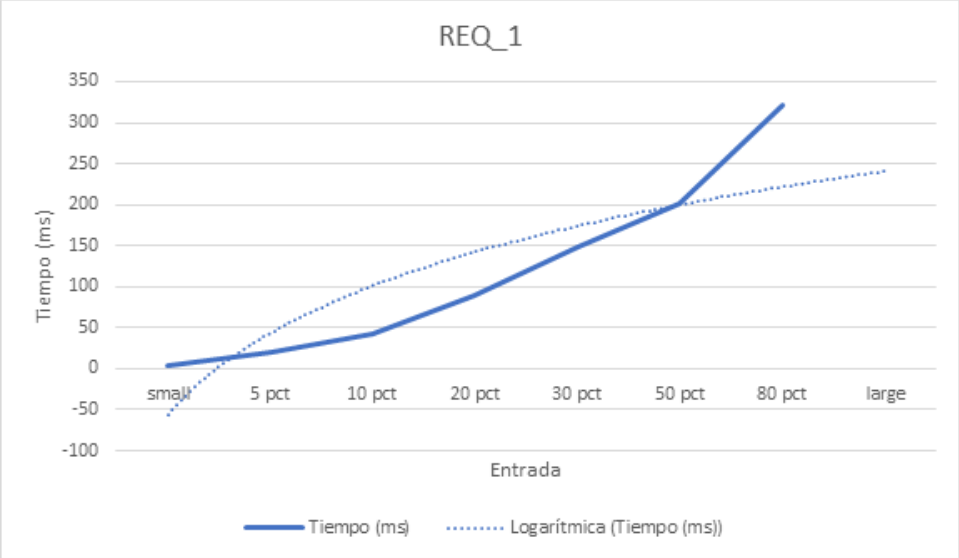
Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	912	4.38
5 pct	4560	20.66
10 pct	9130	43.35
20 pct	18452	88.88
30 pct	27935	148.64
50 pct	46501	200.36

80 pct	74351	321.25
large	92851	405.82

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Dentro de este requerimiento la complejidad teórica la expresamos como un logN gracias a el uso de la función de intervalos que se utilizará dentro del mapa de fechas. Esto se expresa en la función, a pesar de un cambio dentro de la ejecución, la cual puede deberse en mayor parte por la cantidad de datos y el procesamiento de la máquina.

Requerimiento <<2>>

Descripción

```
def req_2(catalog, limi_mag, limf_mag):
    """
    Función que soluciona el requerimiento 2
    """
    mag = catalog['mag_simple']
    rta = om.valuesList(mag, limi_mag-0.1,limf_mag)
    total_sismos = lt.size(rta)
    return rta, total_sismos
```

Dentro de esta función, se toma el mapa de magnitudes para poder obtener los sismos que se encuentran entre un rango determinado, para finalmente retornar los sismos que cumple con la característica mencionada anteriormente.

Entrada	- Catálogo con todas las estructuras de datos del reto - Límite inferior de magnitud - Límite superior de magnitud
Salidas	- La lista de todos los sismos que cumplen con el rango dado por el usuario. - El total de sismos.
Implementado (Sí/No)	Sí. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación variable “mag”	O (1)
Asignación variable “rta”, con el uso de la librería del curso (om.values)	O (log(N+K)), siendo K el número de valores dentro del rango.
Asignación de variable “Total_sismos” con lt.size	O(1)
TOTAL	**O(log(N+K))**

Pruebas Realizadas

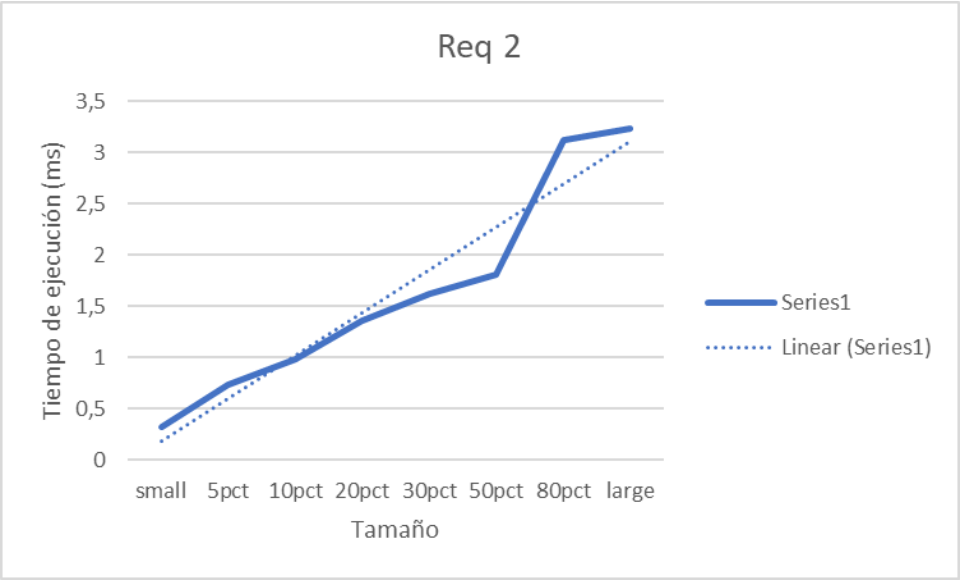
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Para elaborar las pruebas, se mantuvo constante las magnitudes que se deseaban buscar (3.5 y 8.7).

Procesadores	11th Gen Intel(R) Core(TM) i7-11375H @ 3.30GHz 3.30 GHz
Memoria RAM	16 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador x64 Windows 11

Tablas de datos

Entrada	Tiempo (s)
small	0,325
5pct	0,934
10pct	0,781
20pct	1,364
30pct	1,621
50pct	1,805
80pct	3,116
Large	3.237

Graficas



Análisis

Se evidencia que este requerimiento tiene un comportamiento logarítmico, a pesar de tener ciertas decaídas en los tiempos en el trascurso de las pruebas, las cuales pueden deberse al procesamiento del computador. Este comportamiento logarítmico se puede evidenciar en los tamaños más grandes, en los cuales los tiempos de ejecución tienden a mantenerse en un rango más corto, en valores cercanos a 3 (ms).

Requerimiento <<3>>

Descripción

```python
def req_3(catalog, mag_min, prof_max):
    """
    Consulta los eventos sísmicos más recientes que superen una magnitud y no superen una profundidad
    dada

    :param catalog: Catálogo de sismos.
    :param mag_min: Magnitud mínima del evento.
    :param prof_Max: Profundidad máxima del evento.
    :return: final = Lista de los 10 eventos más recientes que cumplen con los criterios.
    """
    mag_map = catalog ['mag_simple']
    mag_max = om.maxKey(mag_map)
    lista_mag = om.values(mag_map,mag_min,mag_max)


    final = lt.newList("ARRAY_LIST")

    for lista in lt.iterator(lista_mag):
        for info in lt.iterator(lista):
            if not info ["depth"] == "" or "Unknown":
                if float(info ["depth"]) <= float(prof_max):
                    lt.addLast(final,info)

    merg.sort(final,cmp_req3)
    total = lt.size(final)

    return final, total
```

Esta función toma el mapa de equipos y se seleccionará el equipo del mapa. Luego de esto se filtrará los partidos del equipo dentro de las fechas otorgadas por el usuario. Además, por toda la función se sacarán los diferentes totales que son necesarios para el view. De esta manera, junto con el catálogo, el nombre del equipo y un intervalo de fechas, llegamos a una lista de partidos la cual cuenta con los partidos del equipo en la fecha y los diferentes totales para presentar en el view.

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Sí. Daniel Mancilla

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Asignación de la variable “mag_map “(arbol que tiene por llaves las magnitudes y por valor lista de diccionarios)	O (1)
Asignación de la variable “mag_max” con la funcion maxKey()	O (logN)
Utilizacion de la funcion om.values para crear una lista con los valores entre mag_min	O (1)
Variable “final= Array List vacío donde se va a almacenar la información para usarla en el tabulate	O (1)
Ciclo dentro de un primer ciclo para recorrer la infamación dentro de la lista para almacenar la información dentro de la lista vacía “final”, a través de dos if en los cuales se dan las condiciones para poder almacenar la información	O (N*M)
Se realiza sort a la lista para que los resultados se guarden de más reciente a más antiguo	O (NlogN)
Se le saca el size a la última lista creada para tener la variable, “total”, que tiene la cantidad de los eventos sucedidos	O(1)
TOTAL	***O(N*M)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Para elaborar las pruebas, se mantuvo constante los parámetros que se deseaban buscar (23.0 y 20).

Especificaciones del computador donde se realizaron las pruebas

- 8th Gen Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz 2.21 GHz

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada: 4.7, 10.00	**Tiempo (s)**
Small	91.864
20pct	679.644
50pct	1745.699
80pct	1679.379
Large	2262.535

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

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

Con las preubas realizadas, como se ve en la gráfica, podemos identificar un aumento (la mayoría de las veces) en el tiempo a medida que se aumenta el tamaño de la carga, aumentando en gran mayoría posterior a los 20pct.

Requerimiento <<4>>

```python
def req_4(catalog, min_sig, max_gap):
    """
    Función que soluciona el requerimiento 4
    """
    sig_map = catalog['sig_map']
    limite_sup = om.maxKey(sig_map)
    mapa_new = om.newMap(omaptype= 'RBT', cmpfunction= cmp_tree)
    mapa_new = om.valuesListMap(sig_map, min_sig, limite_sup, mapa_new)
    lista = om.valuesBelow(mapa_new, max_gap, cmp_tree)
    lista = lt.mini_sort(lista, 'time')
    event15 = lt.subList(lista, lt.size(lista)-14, 15)
    total = lt.size(lista)
    return event15, total
```

Descripción

Entrada	catalog: Un catálogo que parece contener información, ya que se accede a su clave 'sig_map'. min_sig: Un valor utilizado en alguna operación relacionada con sig_map.

	max_gap: Otro valor utilizado en alguna operación, posiblemente relacionada con la búsqueda de valores por debajo de cierto límite.
Salidas	Even15: Los 15 eventos de respues Total: los totales necesarios para consola
Implementado (Sí/No)	Sí. María Lucía Benavides

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación y acceso de datos con parametros	O (1)
Función maxKey que saca la mayor llave	O(logN)
Función valuesListMap que genera un mapa a partir de una lista de valores	O(logM) con m como los valores de la lista
Uso de la función valuesBelow que saca todos los hijso izquierdos de una llave	O (logM) con m como los valores del nuevo mapa
Uso de la función mini_sort	O(k) con k como el numeor de hijos izquierdos
Uso de la sublista	O(1)
Uso del size	O(1)
TOTAL	***O(logN.)***

Pruebas Realizadas

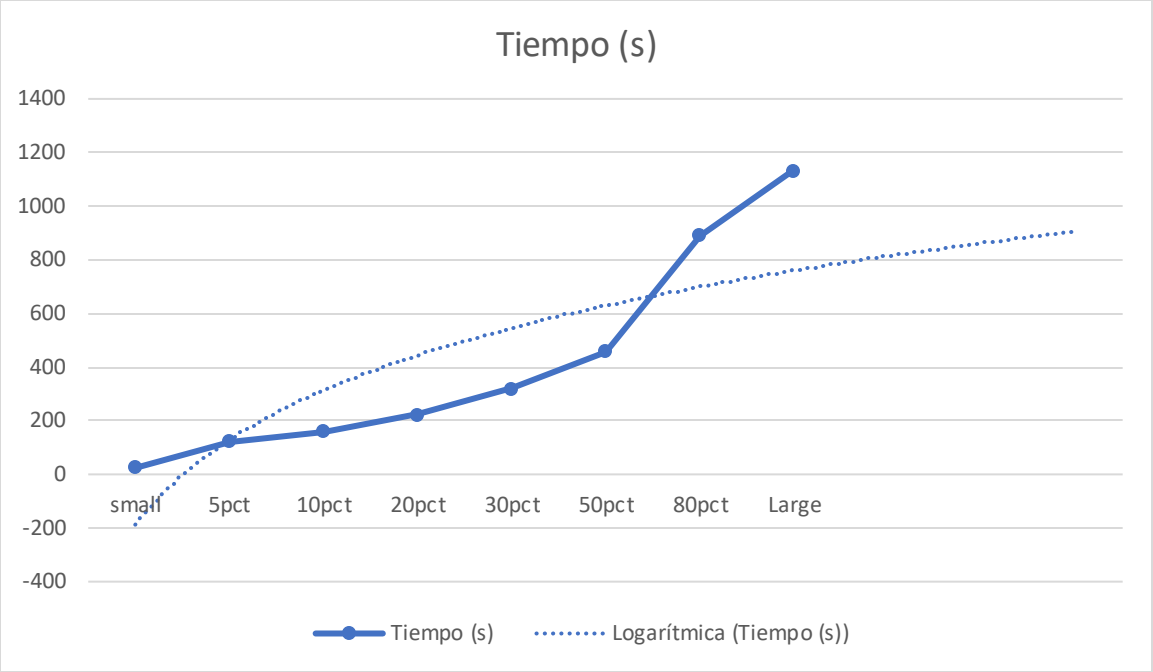
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada : 300, 45.000

Entrada	**Tiempo (s)**
small	24,041
5pct	123,018
10pct	198,823
20pct	254,2372
30pct	320,2832
50pct	457,374
80pct	889,232
Large	1134,239

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Según las gráficas podemos entender el aumento de complejidad que se explica gracias a las diferentes funciones utilizadas y aplicadas dentro del código. Tomando en cuenta lo resultados, podemos asumir un carácter lineal dentro del requerimiento demostrado, que expresa el comportamiento que se entendió dentro del análisis de complejidad gracias a su aproximación logarítmica.

Requerimiento <<5>>

Descripción

```python
def req_5(catalog, depth_min, nst_min):
    """
    Consulta los eventos sísmicos más recientes que superen una profundidad mínima y sean identificados
    por un número mínimo de estaciones de monitoreo.

    :param catalog: Catálogo de sismos.
    :param depth_min: Profundidad mínima del evento.
    :param nst_min: Número mínimo de estaciones de monitoreo.
    :return: Lista de los 20 eventos más recientes que cumplen con los criterios.
    """
    mapa = catalog['dephts']
    limite_sup = om.maxKey(mapa)
    mapa_new = om.newMap(omaptype= 'RBT', cmpfunction= cmp_nst)
    mapa_new = om.valuesListMap1(mapa, depth_min, limite_sup, mapa_new)
    lista = om.valuesAbove(mapa_new, nst_min-1, cmp_tree)
    lista = lt.mini_sort1(lista, 'time')
    if lt.size(lista) >= 20:
        eventos = lista['elements'][:20]
    else:
        eventos = lista
    total = lt.size(lista)

    return eventos, total
```

Dentro de esta función, se toma el mapa de profundidades (árbol rbt), para luego a partir de este y los parámetros del usuario, elaborar otro árbol RBT que cumpla con la segunda petición del usuario, esta respecto a las estaciones. Tras esto, se procede a organizar los valores del segundo árbol dependiendo de su fecha. Finalmente se retornan aquellos sismos que cumplen con los dos requerimientos del usuario, junto con el total de sismos que cumplen.

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Para elaborar las pruebas, se mantuvo constante la fecha inicial y la final que entraban por parámetro (fecha incial: 2006-05-14, fecha final: 2023-04-21) y para evidenciar mejores resultados se hicieron varias pruebas con tres goleadores famosos (Cristiano Ronaldo, Messi, Neymar)

Procesadores	11th Gen Intel(R) Core(TM) i7-11375H @ 3.30GHz 3.30 GHz
Memoria RAM	16 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador x64 Windows 11

Tabla de datos

Entrada	- Catálogo con todas las estructuras de datos del reto - Profundidad mínima - Estaciones mínimas
Salidas	- La lista de eventos

	- Total de sismos
Implementado (Sí/No)	Sí. Jessica Garay

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación variable "mapa" equivalente al árbol RBT de profundidades.	O (1)
Asignación variable "límite_sup"	O (log(N))
Elaboración de un nuevo árbol RBT (mapa_new)	O (1)
Asignación del mapa a partir de los valores dentro del rango de profundidad mínima y profundidad máxima (limite_sup). Se toman los valores y se crea un nuevo mapa. Esto se realizó tras realizar modificaciones a DISClib.	O (log(M)), siendo M los valores dentro el rango de profundidad mínima y limite_sup.
Asignación variable "lista", se crea una nueva con todos aquellos valores del árbol (mapa_new) que cumplen con ser mayores o iguales al número de estaciones dado por el usuario.	O (M+K), siendo K el número de valores dentro del rango.
Organización cronológica de la lista con base en sus fechas.	O(QlogQ), siendo Q el número de elementos de la lista.
Verificación del tamaño de la lista, para brindar al usuario únicamente los 20 sismos más recientes. Si es necesario creación de una sublista.	O (1)
Tamaño de la lista para saber el número de sismos.	O (1)
TOTAL	***O(NlogN)***

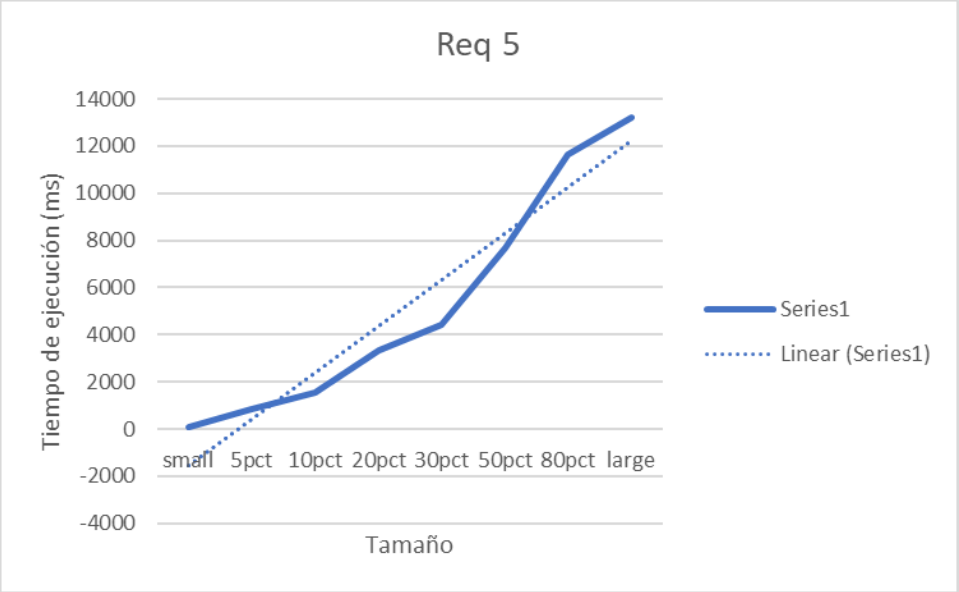
Pruebas Realizadas

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 (ms)**
small	92.272
5pct	815.005
10pct	1526.381
20pct	3350.361
30pct	4416.0198
50pct	7691.673
80pct	11632.417

Large	13226.27

Graficas



Análisis

Se evidencia que este requerimiento tiene un comportamiento logarítmico, y aumenta a una mayor velocidad en comparación con otros requerimientos debido a la creación de nuevos mapas y el uso del sorted para organizar la lista. A pesar de esto, en los valores más grandes el cambio de tiempo no es tan grande. A pesar de demorar más en desarrollarse este requerimiento, su uso nos permite tener una carga de datos estable.

Requerimiento <<6>>

Descripción

```python
def req_6(catalog, year, lat, long, radius, n):
    """
    Función que soluciona el requerimiento 6
    """
    latLongMap = me.getValue(mp.get(catalog['lat_long'], year))['latLong_records']
    hav_entry = (lat, long)
    diferencia = om.valuesBelow(latLongMap, radius, cmp_diferencia_radius, (True, hav_entry ) )
    signList = lt.mini_sort(diferencia, 'sig')
    sign = lt.getElement(signList, lt.size(signList))
    time = lt.mini_sort(diferencia, 'time')
    listBA = binary_storage(time, sign, n)
    total = lt.size(diferencia)
    return sign, total, listBA
```

Esta función toma como entrada un catálogo de todos los sismos, incluyendo el año, la latitud, la longitud, el radio y el número de eventos cercanos a ese radio. Además, utiliza la estructura del catálogo llamada 'lat_long', la cual es un mapa ordenado que contiene las latitudes y longitudes organizadas de una manera específica para facilitar la búsqueda, utilizando llaves de tipo tupla. Dentro de la función, se emplea la función 'values below', que extrae todos los hijos izquierdos de un nodo y los compara con el radio dado. Luego, se utiliza la función 'mini_sort' para ordenar los elementos de la lista resultante según un criterio específico. Finalmente, se hace uso de la función 'binary storage', la cual identifica un elemento y selecciona los 'n' valores más cercanos a este. El resultado se almacena en una variable, y la función retorna el elemento más grande, el total y las listas de los elementos anteriores y posteriores dentro de una tupla.

Entrada	Catálogo de sismos, año, latitud, longitud, radio, n eventos cercanos
Salidas	Evento sísmico más significativo
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Sacar el mapa lat_longRecords	O(1)
Recorrido de valuesBelow	O(logN)
Uso de la función mini_sort	O(m) m siendo el número de elementos dentro del radio
Uso de binary_storage	O(NlogN)
Uso de lt.size()	O (1)
TOTAL	***O(NlogN)***

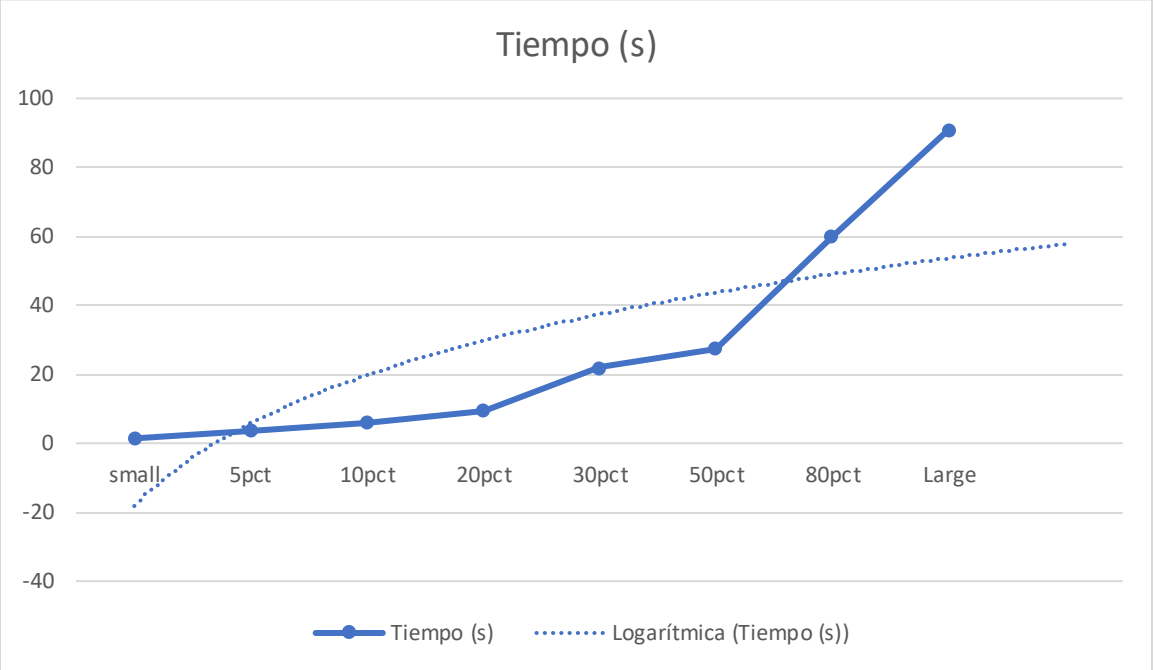
Pruebas Realizadas

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)
small	1.396
5pct	3.623
10pct	6.051
20pct	9.515
30pct	21.846
50pct	27.335
80pct	59.907
Large	90.822

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al analizar este requerimiento podemos apreciar un comportamiento similar a lo planteado en su análisis de complejidad. Si tomamos en cuenta factores como la utilización de la función de totales, podemos asumir y entender este tipo de comportamiento dentro de la gráfica temporal. Así mismo, gracias a las diferentes implementaciones, se asume constante la segunda gráfica gracias a diversas características.

Requerimiento <<7>>

Descripción

```python
def req_7(catalog, year, title, prop, bins):
    """
    Función que soluciona el requerimiento 7
    """
    #prop llega como un entero 1, 2 o 3
    mapa_year = catalog['prop_map']
    mapa_tit = me.getValue(mp.get(mapa_year, year))['titles']
    datos = me.getValue(mp.get(mapa_tit, title))
    propi = lt.getElement(datos['props'], prop)
    records = datos['records']
    range_bins ={}
    low = round(min(propi['elements']), 4)
    high = round(max(propi['elements']), 4)
    totales = (om.size(records) ,low, high)
    dif_prop = (high- low)/bins
    residuo = (high- low)%bins
    high = low + dif_prop
    for i in range(1, bins+1):
        llave= str(low) + '-' + str(high)
        t_low = [0, 0, 0, 0]
        t_high = [0, 0, 0, 0]
        t_low[prop] = low
        t_high[prop] = high
        valores = om.values(records, t_low, t_high)
        range_bins[llave] = valores
        low = high
        high= round(low + dif_prop, 4)
    return range_bins, totales
```

Dentro de esta función, se toma la tabla de hash de propiedades (árbol rbt) , para luego a partir de este y los parámetros del usuario, obtener los valores necesarios a partir de lo que requiere el usuario. Luego de obtener estos datos, se realiza un diagrama de barras.

Entrada	catalog: Catálogo con todas las estructuras que organizan la información Year: El año de búsqueda Title: Región de búsqueda Prop: Propiedad que se desea usar Bins: segmentos del histograma
Salidas	Información del histograma Totales de sismos
Implementado (Sí/No)	Sí.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

```python
def req_7(catalog, year, title, prop, bins):
    """
    Función que soluciona el requerimiento 7
    """
    #prop llega como un entero 1, 2 o 3
    mapa_year = catalog['prop_map']
    mapa_tit = me.getValue(mp.get(mapa_year, year))['titles']
    datos = me.getValue(mp.get(mapa_tit, title))
    propi = lt.getElement(datos['props'], prop)
    records = datos['records']
    range_bins ={}
    low = round(min(propi['elements']), 4)
    high = round(max(propi['elements']), 4)
    totales = (om.size(records) ,low, high)
    dif_prop = (high- low)/bins
    residuo = (high- low)%bins
    high = low + dif_prop
    for i in range(1, bins+1):
        llave= str(low) + '-' + str(high)
        t_low = [0, 0, 0, 0]
        t_high = [0, 0, 0, 0]
        t_low[prop] = low
        t_high[prop] = high
        valores = om.values(records, t_low, t_high)
        range_bins[llave] = valores
        low = high
        high= round(low + dif_prop, 4)
    return range_bins, totales
```

Pasos	**Complejidad**
Asignación de variable “mapa_year” con base en la tabla de hash de “prop_map”	O (1)
Asignación de variable “mapa_tit” con me.getValue	O (N)
Asignación de variable “datos” con me.getValue con base en la asignación anterior.	O(M), siendo M la longitud de “mapa_tit”
Get_Element a partir de la lista de datos[“records”].	O(1)
Asignación del valor con menor valor	O(1)
Asignación del valor con mayor valor	O(1)
Determinar el tamaño de la variable “records”	O(1)
Operación matemática sobre variables (asignación en “dif_prop”	O(1)
Operación matemática sobre variables (asignación en “high”	O(1)
Ciclo for para creación de histograma	O(B), siendo B el número de segmentos dados por el usuario
Asignación de la variable “Valores” con base en om.Values que obtiene los valores que cumplen con ser parte de un rango.	O(LogN+K), siendo K la cantidad de valores dentro del rango.
TOTAL	***O (N)***

Pruebas Realizadas

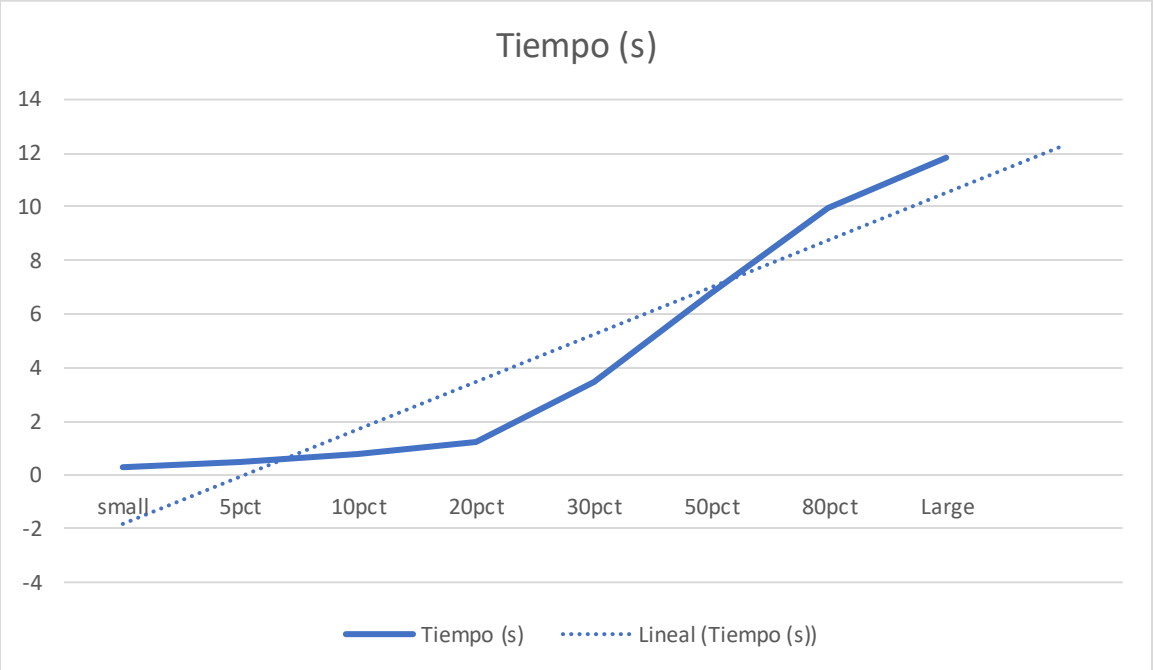
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**

small	0,292
5pct	0,485
10pct	0,789
20pct	1,238
30pct	3,478
50pct	6,768
80pct	9,928
Large	11,832

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Las gráficas demuestran de manera adecuada el análisis otorgado dentro del análisis de complejidad y reflejan la efectividad de usar las diferentes estructuras en comparación a otras. Gracias al uso transversal de mapas dentro de la creación de una estructura de datos, se puede inferir que la buena y clara complejidad se debe a este factor.