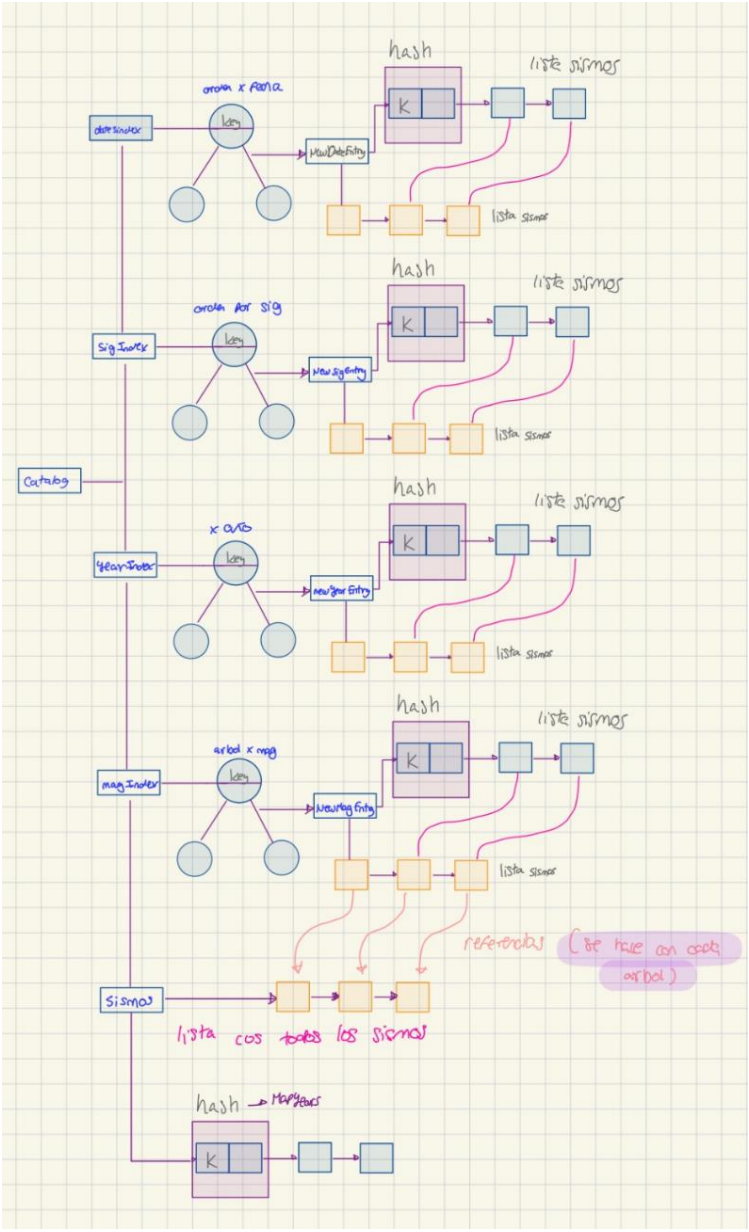
ANÁLISIS DEL RETO

Paula Avila, 202212078, pd.avila@uniandes.edu.co

Estefany Duarte, 202115210, e.duartej@uniandes.edu.co

Lucas Padilla , 202212727, *l.padillag*@uniandes.edu.co

Diagrama de las estructuras de datos



Requerimiento <<1>>

Descripción

```python
def req_1(catalog,inicialDate, finalDate):
    """
    Función que soluciona el requerimiento 1
    """
    lst = om.values(catalog["dateIndex"], inicialDate, finalDate)
    filtrada = om.newMap(omaptype="RBT",
                         cmpfunction=compareDates)
    totaldates = 0
    totalevents = 0
    for lstsismos in lt.iterator(lst):
        fecha = lstsismos["time"]
        om.put(filtrada,fecha,lstsismos)
        totaldates +=1
        totalevents += lstsismos["events"]

    return totaldates, totalevents, filtrada
```

Entrada	Catalog corresponde al diccionario con la base de datos, inicialDate corresponde a la fecha inicial en date.datetime y finalDate corresponde a la fecha final en date.datetime
Salidas	Totaldates corresponde a la cantidad de fechas encontradas en la base de datos, totalevents corresponde a la cantidad de sismos sin discriminar entre sus fechas, dentro de la base de datos y; filtrada corresponde al mapa que contiene los sismos entre las fechas con la misma estructura que el mapa dateIndex.
Implementado (Sí/No)	Sí por Estefany

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ingresar a los valores del RBT	O(log(m))
Generar un nuevo RBT	O(1)
Generar contadores	O(1)
For para iteración de los valores	O(m)
Acceso al time del diccionario de la lstsismos	O(1)
Uso de la función put para agregar al nuevo RBT	O(log(m))
Suma de datos obtenido por accesos en un diccionario	O(1)
TOTAL	***m*log(m)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

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

Entrada	**Tiempo (ms)**
Small, 1999-03-21T05:00, 2004-10-23T17:30	15,838
5 pct, 1999-03-21T05:00, 2004-10-23T17:30	99,371
10 pct, 1999-03-21T05:00, 2004-10-23T17:30	697,864
20 pct, 1999-03-21T05:00, 2004-10-23T17:30	409,242
30 pct, 1999-03-21T05:00, 2004-10-23T17:30	649,414
50 pct, 1999-03-21T05:00, 2004-10-23T17:30	3277,604
80 pct, 1999-03-21T05:00, 2004-10-23T17:30	1817,245
Large, 1999-03-21T05:00, 2004-10-23T17:30	2367,042

Graficas

Análisis

Como observamos, en este caso tenemos un delta con anomalías como es el caso del 50%, esto puede deberse tanto a la distribución de datos como (lo que creemos) la forma de la carga de datos y el rendimiento del equipo. Por otro lado, si observamos el comportamiento vemos que sí hay una tendencia a un comportamiento logarítmico por lo que, la ejecución del código correspondería a lo visto

en la teoría y el análisis del código, donde para un caso promedio en el uso de arboles binarios ordenados tenemos un O(nlogn).

Requerimiento <<2>>

```
def req_2(catalog,inicialMag, finalMag):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    lst = om.values(catalog["magIndex"], inicialMag, finalMag)
    filtrada = om.newMap(omaptype="RBT",
                         cmpfunction=compareDates)
    totaldates = 0
    totalevents = 0
    for lstsismos in lt.iterator(lst):
        magni = lstsismos["mag"]
        om.put(filtrada,magni,lstsismos)
        totaldates +=1
        totalevents += lstsismos["events"]

    return totaldates, totalevents, filtrada
```

Descripción

Este requerimiento se encarga de retornar los sismos que se encontraban en un rango de magnitudes determinadas por el usuario. Estas se van a agregar a una lista, y se va a almacenar la cantidad de eventos entre el rango.

Entrada	El diccionario 'catalog' con la carga de datos, y el rango de la magnitud que se desea consultar.
Salidas	Todos los sismos entre esas magnitudes y la cantidad de eventos, junto al nuevo árbol creado filtrado.
Implementado (Sí/No)	Si, por Paula Avila.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ingresar a los valores entre magnitudes	O(1)
Iterar por cada elemento de la lista, siendo esta muy pequeña y constante	O(m)
Agregar al nuevo mapa cada sismo	O(1)

Ir sumando la cantidad de sismos agregados	O(1)
Retornar todo	O(1)
TOTAL	***O***(contante)

Pruebas Realizadas

Procesadores	**Intel(R) Core (TM) i3-10110U CPU @ 2.10GHz 2.60 GHz**
Memoria RAM	4 GB
Sistema Operativo	Windows 10 Pro

Entrada	**Tiempo (s)**
small	3.53
5 pct	4.41
10 pct	4.40
20 pct	4.62
30 pct	4.35
50 pct	4.28
80 pct	4.83
large	6.46

Graficas

Análisis

En este caso vemos que se mantiene en un aproximado de O(m), dado que, si bien no presenta una línea completamente recta, tiene un comportamiento logarítmico, esto se debe a que el cogido solo consiste en acceder e ingresar datos al árbol, lo cual no genera que se aumente la complejidad en gran medida.

Este comportamiento se rectifica con la gráfica, la cual presenta una apariencia logarítmica.

Requerimiento <<3>>

```
def req_3(catalog, nmag, ndepth):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    maxmag = om.maxKey(catalog["magIndex"])
    lst = om.values(catalog["magIndex"],nmag, maxmag)

    filtrada = om.newMap(omaptype="RBT",
                         cmpfunction=compareDates)
    totaldates = 0
    totalevents = 0
    for datos in lt.iterator(lst):
        sismos = datos["lstSismos"]
        eventos = datos["events"]
        if eventos != 0:
            for sismo in lt.iterator(sismos):
                depth = sismo["depth"]
                if depth <= ndepth:
                    updateDateIndex(filtrada, sismo)
                    totaldates +=1
                    totalevents +=1

    if om.size(filtrada)<=10:
        return totaldates, totalevents, filtrada
    else:
        filtrada2 = om.newMap(omaptype="RBT",
                              cmpfunction=compareDates)
        for _ in range(10):
            key = om.maxKey(filtrada)
            value = om.get(filtrada,key)
            value = me.getValue(value)
            om.deleteMax(filtrada)
            om.put(filtrada2,key=key,value=value)
        return totaldates, totalevents, filtrada2
```

Este requerimiento se encarga de retornar los 10 eventos más recientes ocurridos según una magnitud y profundidad indicadas. Esto lo hace a través de entrar al árbol ordenado por magnitud y comparar con la

condición de profundidad, para entrar al árbol ordenado por fechas. Seguidamente, se calcula el 'top' 10 de los eventos más actuales maxKey y deleteMax.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El diccionario 'control' con la carga de datos,
Salidas	Devuelve la cantidad de eventos que ocurren con esas condiciones y el árbol filtrado. Y el top 10.
Implementado (Sí/No)	Si, Implementado por Paula Avila

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener la llave mas grande del mapa.	O(1)
Obtener los valores entre las llaves	O(1)
Se itera por cada elemento seleccionado anterior, y se compara para determinar si cumple la condición de profundidad.	O(n)
Se itera 10 veces para determinar el top 10 de los mas actuales	O(1)
Se va agregando en un nuevo árbol cada elemento	***O***(log(n))
TOTAL	***O***(n*log(n))

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

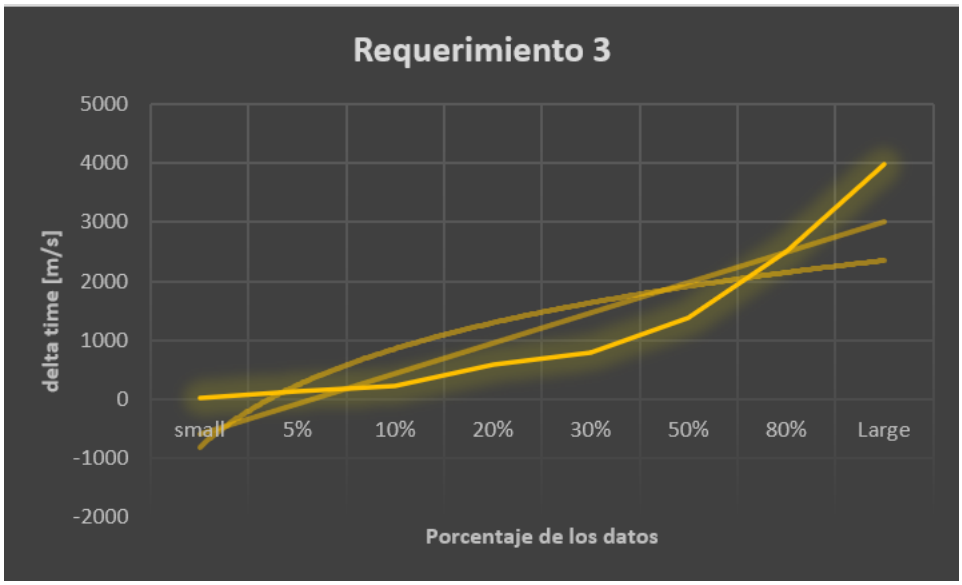
Procesadores	**Intel(R) Core (TM) i3-10110U CPU @ 2.10GHz 2.60 GHz**
Memoria RAM	4 GB
Sistema Operativo	Windows 10 Pro

Entrada	**Tiempo (ms)**
small	26.9
5 pct	126.11
10 pct	229.5
20 pct	587.3
30 pct	795.8
50 pct	1375.24

80 pct	2504.35
large	3973.1

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Lo primero que hace es acceder a la llave más grande del árbol ordenado por magnitudes. Después, selecciona con la función values las llaves que están entra la más grande y la llave que fue entrada, lo que devuelve una lista.
Seguidamente, se itera sobre esta lista para filtrarla y seleccionar solo las llaves que cumplan con la segunda condición. Posteriormente, se actualiza el mapa con estas llaves. Por último, para realizar el top 10, se itera 10 veces, para seleccionar el máximo con maxKey y deleteKey sucesivamente. En la gráfica se puede comprobar que el codigo se comporta como n * log (n)

Requerimiento <<4>>

Descripción

```python
def req_4(catalog, nsig, ngap):
    """
    Función que soluciona el requerimiento 4
    """
    maxsig = om.maxKey(catalog["sigIndex"])
    lst = om.values(catalog["sigIndex"],nsig, maxsig)

    filtrada = om.newMap(omaptype="RBT",
                         cmpfunction=compareDates)
    totaldates = 0
    totalevents = 0
    for datos in lt.iterator(lst):
        sismos = datos["lstSismos"]
        eventos = datos["events"]
        if eventos != 0:
            for sismo in lt.iterator(sismos):
                gap = sismo["gap"]
                if gap <= ngap:
                    updateDateIndex(filtrada, sismo)
                    totaldates +=1
                    totalevents +=1
    if om.size(filtrada)<=15:
        return totaldates, totalevents, filtrada
    else:
        filtrada2 = om.newMap(omaptype="RBT",
                              cmpfunction=compareDates)
        for _ in range(15):
            key = om.maxKey(filtrada)
            value = om.get(filtrada,key)
            value = me.getValue(value)
            om.deleteMax(filtrada)
            om.put(filtrada2,key=key,value=value)
        return totaldates, totalevents, filtrada2
```

Entrada	Catalog corresponde al diccionario con la base de datos, nsig corresponde al valor mínimo de la significancia y ngap sería el valor máximo de la distancia azimutal.
Salidas	Totaldates corresponde a la cantidad de fechas encontradas en la base de datos, totalevents corresponde a la cantidad de sismos sin discriminar entre sus fechas, dentro de la base de datos y; filtrada corresponde al mapa que contiene los sismos entre las fechas con la misma estructura que el mapa dateIndex.
Implementado (Sí/No)	Sí por Estefany

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener la llave mas grande del mapa.	O(1)
Obtener los valores entre las llaves	O(1)
Se itera por cada elemento seleccionado anterior, y se compara para determinar si cumple la condición de profundidad.	O(n)
Se itera 10 veces para determinar el top 10 de los mas actuales	O(1)
Se va agregando en un nuevo árbol cada elemento	***O***(log(n))
TOTAL	***O***(n)

Pruebas Realizadas

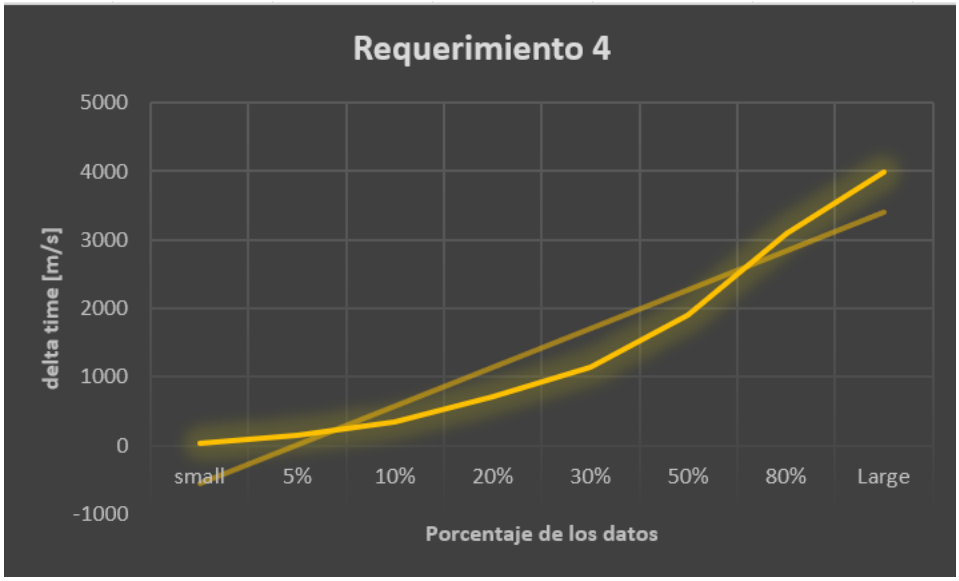
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

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

Entrada	**Tiempo (ms)**
Small, 300, 45.00	29,375
5 pct, 300, 45.00	159,73
10 pct, 300, 45.00	344,159
20 pct, 300, 45.00	711,89
30 pct, 300, 45.00	1141,80
50 pct, 300, 45.00	1895,73
80 pct, 300, 45.00	3101,71
Large, 300, 45.00	3989,157

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Vemos como la complejidad se ve determinada por el for -> O(n). Debido a esto encontramos el comportamiento de la función. Esto debido a que la parte que reporta una complejidad no constante en nuestra función es la de generar el mapa filtrado por las condiciones descritas en el documento. De modo

que se ve una semejanza clara entre la teoría vista en clase, en lo observado al analizar la complejidad temporal del código y de la complejidad observada en la ejecución del mismo.

Requerimiento <<5>>

Descripción

```python
def req_5(catalog, mindepth, minnst):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5

    prof = om.maxKey(catalog["depthIndex"])
    estaciones = om.maxKey(catalog["nstIndex"])

    filtro = om.values(catalog["depthIndex"], mindepth, prof)

    listaFin = lt.newList(datastructure="ARRAY_LIST")
    for valor in lt.iterator(filtro):
        condicion = om.values(catalog["nstIndex"], minnst, estaciones)
        for temblor in lt.iterator(condicion):
            for est in lt.iterator(temblor["nst"]):
                lt.addLast(listaFin, est)

    Final = lt.size(listaFin)

    sorteados = sorted(lt.toArray(listaFin), key=lambda x: x["time"], reverse=True)

    elegidos = lt.subList(sorteados, 0, 20) if Final > 20 else sorteados

    return elegidos, Final
```

Entrada	A esta función le entran por parámetros: Catalog que es el diccionario con la base de datos, mindepth que es la profunidad minima del evento, y minnst que es el número mínimo de estaciones que detectan el evento.
Salidas	Retorna elegidos, el cual contiene el número total de eventos sísmicos registrados dentro de los límites de profundidad y estaciones de medición indicadas.
Implementado (Sí/No)	Sí por Lucas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Obtener profundidad máxima y número máximo de estaciones	O(1)
Filtrar temblores por profundidad	O(1)
Crear lista para almacenar estaciones	O(1)
Iterar por temblores y sus estaciones	O(n*m) n son temblores, m son estaciones
Hallar tamaño de lista	***O***(1)
Ordenar por tiempo de mayor a menor	***O***(n log n)
Seleccionar top 20	***O***(1)
TOTAL	***O***(n log m) n son temblores, m son estaciones

Pruebas Realizadas

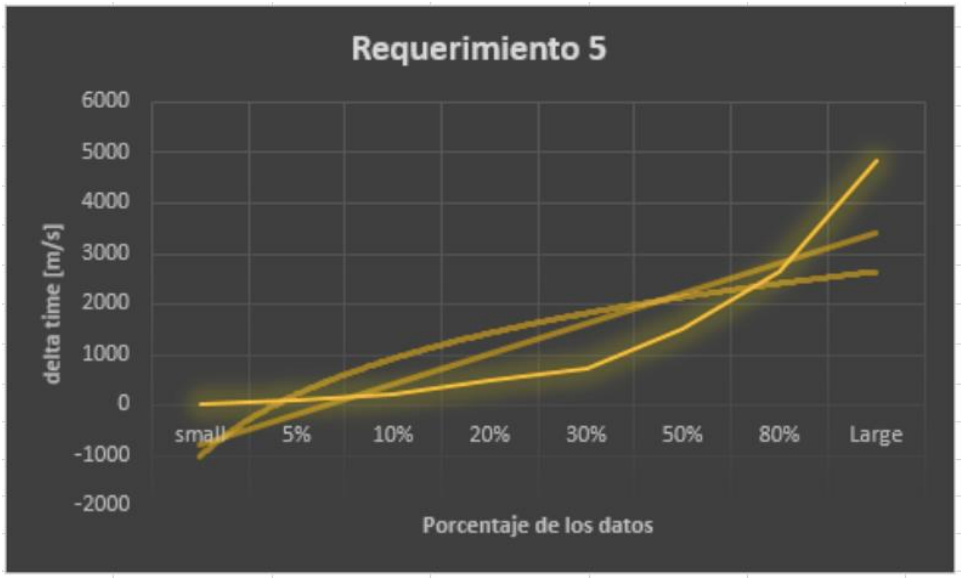
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

Procesadores	1,8 GHz Intel Core i5 de dos núcleos
Memoria RAM	8 GB
Sistema Operativo	MacOS Monterey

Entrada	**Tiempo (ms)**
Small	30,5
5 pct	113,73
10 pct	224,03
20 pct	496,5
30 pct	749,12
50 pct	1502,11
80 pct	2639,4
Large	4853,9

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad de este requerimiento está dada principalmente por dos operaciones, la iteración sobre temblores filtrados y las estaciones asociadas, junto al ordenamiento de las estaciones por tiempo descendente. La iteración que se realiza implica acceder y procesar información asociada a cada temblor y sus estaciones, por lo que tiene una complejidad de O(n*m) donde n son temblores y m son estaciones. En el ordenamiento hecho con sorted, este tiene una complejidad de ***O***(n log m). Como vemos en ambos casos, la complejidad depende de la cantidad de temblores y estaciones en el catálogo. Al ver esta complejidad, vemos que la función se comporta acorde a esta.

Requerimiento <<6>>

```python
def req_6(data_structs, year, latRef, longRef, radio, n):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    mapYears = data_structs['mapYears']
    year = int(year)
    entry = mp.get(mapYears, year)
    earthquakesYear = me.getValue(entry)
    earthquakesYear = earthquakesYear['datos']
    tembloresArea = lt.newList('ARRAY_LIST')

    for temblor in lt.iterator(earthquakesYear):
        if AreaPresent(latRef, longRef, float(temblor['lat']),   float(temblor['long']), radio):
            temblor['distance'] = haversine(latRef, longRef, float(
                temblor['lat']),   float(temblor['long']))
            lt.addLast(tembloresArea, temblor)

    sortedAreas = quk.sort(tembloresArea, sort_criteria_REQ6)
    maxArea, posicion = encontrar_mayor_magnitud(sortedAreas)

    Sublista = obtener_sublista(sortedAreas, posicion, n)

    if lt.size(Sublista) <= 6:
        TablaMaxArea, TablaTopN = CreateTables(maxArea, Sublista)
    else:
        List6 = FirstandALst(Sublista)
        TablaMaxArea, TablaTopN = CreateTables(maxArea, List6)

    return TablaMaxArea, TablaTopN, lt.size(tembloresArea), lt.size(Sublista), maxArea
```

Descripción

Este requerimiento se encarga de retornar el evento sísmico más significativo ocurrido en el área durante el año indicado en formato tabla y los N eventos sísmicos registrados dentro del área circundante.

Entrada	El diccionario 'catalog' con la carga de datos, año relevante, latitud de referencia para el área de eventos (lat), longitud de referencia para el área de eventos (long), radio [km] del área circundante (float), número de los N eventos de magnitud más cercana a mostrar.
Salidas	El evento sísmico más significativo ocurrido en el área durante el año indicado en formato tabla, número total de eventos sísmicos registrados dentro del área circundante y la tabla con los N eventos sísmicos más cercanos en tiempo
Implementado (Sí/No)	Si, por Paula Avila.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ingresar al hash ordenada por años.	O(1)
Obtener la lista que contiene todos los sismos de ese año	O(1)

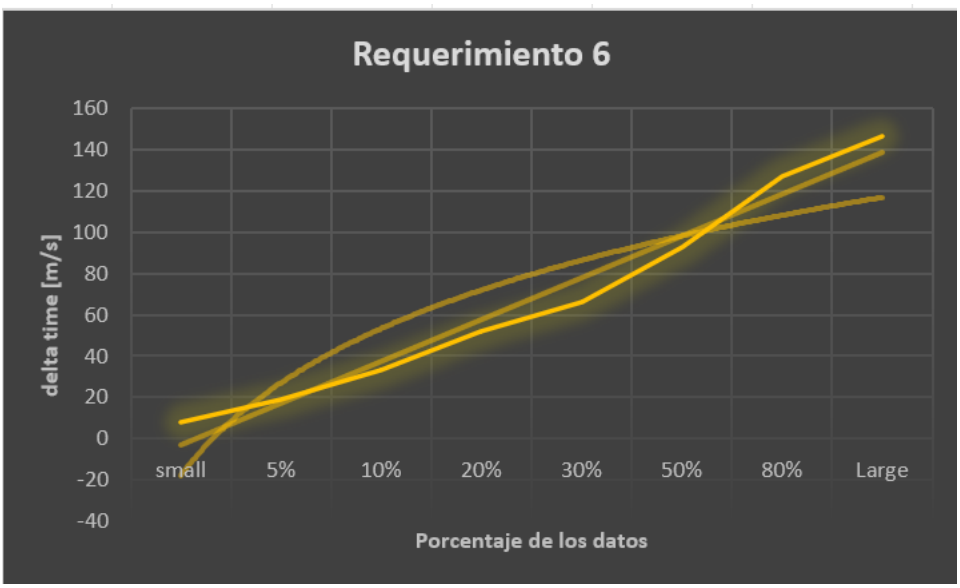
Crear un array	O(1)
Iterar por cada sismo presente en ese año	O(m)
Comparar el área de ese sismo, para determinar si se encuentra en la zona.	O(1)
Agregar a una lista de sismos si si se encuentra en la zona	O(1)
Ordena la lista	O(n)
Retornar el máximo	O(1)
Crear sublista dependiendo de los N	O(1)
Llamar función de printeo	O(1)
Retornar tablas, tamaños y el área	O(1)
TOTAL	***O(m)***

Pruebas Realizadas

Procesadores	**Intel(R) Core (TM) i3-10110U CPU @ 2.10GHz 2.60 GHz**
Memoria RAM	4 GB
Sistema Operativo	Windows 10 Pro

Entrada	**Tiempo (s)**
small	7.83
5 pct	18.71
10 pct	33.5
20 pct	52.19
30 pct	66.53
50 pct	92.65
80 pct	126.9
large	146.26

Graficas



Análisis

En este caso vemos que se mantiene en un aproximado de O(m), dado que, lo que brinda principalmente la complejidad al código es la función de iterar por cada sismo que ocurre en ese año, y ordenar la lista con los sismos ya filtrados.

Las funciones auxiliares consisten en comparar los valores para determinar si los sismos entran en consideración o no, a través de la fórmula de Hevier, proporcionada. Por otro lado, la función auxiliar de las sublistas se encarga de recortar de acuerdo con lo solicitado.

Este comportamiento se rectifica con la gráfica, la cual presenta una apariencia casi que línea, por lo que se puede determinar que la complejidad si es de O(m):

Requerimiento <<7>>

```python
def req_7(catalog, year, title, propiedad, bins):
    """
    Función que soluciona el requerimiento 7
    """
    mapYears = catalog["yearIndex"]
    datorYear = om.get(mapYears,year)
    sismosYear = me.getValue(datorYear)

    sismos = sismosYear["lstSismos"]

    mapProp = om.newMap(omaptype="RBT",
                        cmpfunction=compareDates)

    filtrada = om.newMap(omaptype="RBT",
                         cmpfunction=compareDates)

    for sismo in lt.iterator(sismos):
        if title in sismo["title"]:
            updateDateIndex_7(filtrada,sismo, propiedad)
            updatePropIndex_7(mapProp,sismo, propiedad)

    prop_min = float(om.minKey(mapProp))
    prop_max = float(om.maxKey(mapProp))
    inter_total = prop_max-prop_min
    lin_int = round(prop_min,2)
    paso = round((inter_total/bins), 2)
    n_pasos = bins

    intervalos = om.newMap(omaptype="RBT",
                           cmpfunction=compareDates)

    for i in range(1,n_pasos+1):
        if i < n_pasos:

            lsu_int = round((lin_int+ paso),2)
            llave = f"({lin_int}, {lsu_int})"
            valor = 0
            eventos  = om.values(mapProp, lin_int, lsu_int)
            for evento in lt.iterator(eventos):
                valor += evento["events"]
            om.put(intervalos,llave, valor)

        else:
            llave = f"({lin_int}, {prop_max})"
            valor = 0
            eventos  = om.values(mapProp, lin_int, prop_max)
            for evento in lt.iterator(eventos):
                valor += evento["events"]
            om.put(intervalos,llave, valor)

        lin_int = lsu_int

    #Sacar valores:

    llaves_int = om.keySet(intervalos)

    numpy_intervalos = []
    valores = []

    for llave in lt.iterator(llaves_int):
        numpy_intervalos.append(llave)
        dato = om.get(intervalos,llave)
        valores.append(me.getValue(dato))

    numpy_intervalos = np.array(numpy_intervalos)
    valores = np.array(valores)
    tamano = om.size(filtrada)

    return tamano, filtrada,numpy_intervalos,valores
```

Descripción

Entrada	Catalog corresponde al diccionario con la base de datos, year corresponde al año de interes en date.datetime, title corresponde a la zona de interés, prop corresponde a la propiedad de interés y bins al numero de barras que deseamos en el gráfico
Salidas	Tamano corresponde a la cantidad de datos relevantes, filtrada al mapa con los sismos relevantes, numpy_intervalos es el numpy con los nombres de los intervalos y values es el numpy con el valor de la cantidad de sismos en razón a la casilla correspondiente en numpy_intervalos
Implementado (Sí/No)	Sí por Estefany

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Acedemos al RBT yearIndex, y utilizamos la función get() y getValue() para obtener el dato de los sismos relevante para la función con respecto al año	O(1)
Accedemos a la lista de los sismos	O(1)
Generaramos dos nuevos RBT (filtrada y mapProp)	O(1)
For para iteración de los sismos en la lista de sismos	O(m)
Acceso al time del diccionario de la lstsismos	O(1)
Uso de la función auxiliar updateDateIndex_7()	O(log(m))
Uso de la función auxiliar updatePropIndex_7()	O(log(m))
Acceder a las llaves min y max del RBT mapProp	O(1)
Realizar operaciones matemáticas basicas	O(1)
Crear un tercer RBT intervalos	O(1)
For en el tamaño de bins	O(c)
Usar values() para los límites de paso	O(log(m))
For evento en la lista eventos retornada por values	O(m)
Usar la función put para agregar	O(m)
Generar el keySet para intervalos	O(c)
For de llave en la lista de llaves	O(c)
Usar las funciones get, getValue y append	O(m)
Generar los np.array	O(1)
TOTAL	***m*log(m)***

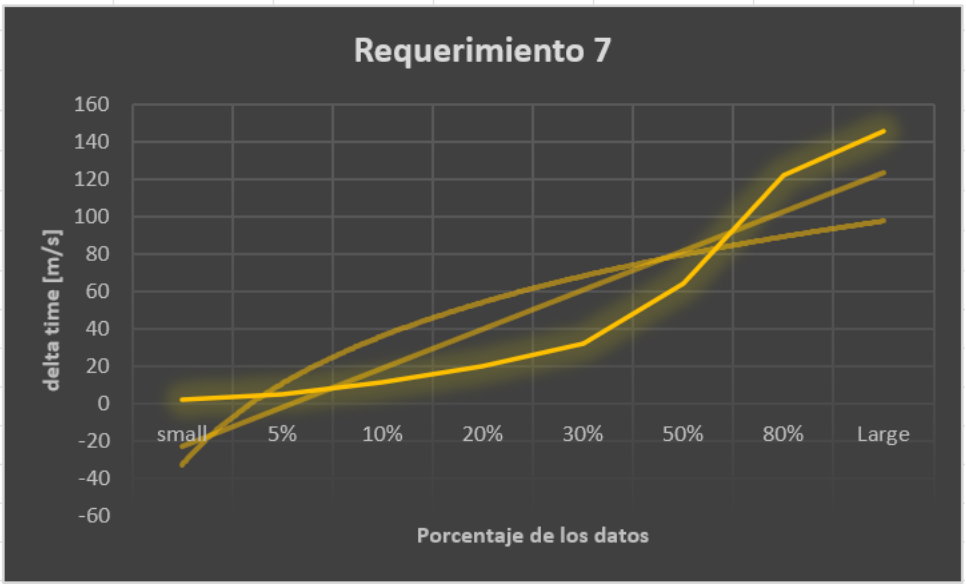
Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

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

Entrada	**Tiempo (ms)**
Small, 2022, Alaska, mag, 10	2,38
5 pct, 2022, Alaska, mag, 10	5,13
10 pct, 2022, Alaska, mag, 10	11,474
20 pct, 2022, Alaska, mag, 10	20,342
30 pct, 2022, Alaska, mag, 10	32,427
50 pct, 2022, Alaska, mag, 10	64,206
80 pct, 2022, Alaska, mag, 10	122,235
Large, 2022, Alaska, mag, 10	146,006

Graficas



Análisis

A diferencia de lo que vemos en el requerimiento 7 donde vemos la misma complejidad teórica, O(mlogm), donde en ese caso vemos una tendencia más a la parte logm, en este requerimiento vemos una tendencia en el comportamiento hacia m. También está el hecho de que omitimos por concepciones generales de la estimación de O la influencia de los for del n_intervalos, que en el caso presentado es la constante 10. En el caso de esta función puede ser que también para cantidades más altas de datos se pueda evidenciar de mejor manera la parte logarítmica de la complejidad.