Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ANÁLISIS DEL RETO

Harold Esteban Piñeros Monroy, 202316402, h.pineros@uniandes.edu.co

Carlos Alberto Poveda Riaño , 202315546, ca.povedar1@uniandes.edu.co

Luis Sebastián Contreras Diaz, 202311819, ls.contreras@uniandes.edu.co

Requerimiento 1

Descripción

```python
def getDatesByRange(analyzer, initialDate, finalDate):
    """
    Retorna el numero de crimenes en un rago de fechas.
    """

    final = lt.newList('SINGLE_LINKED')
    dic = {}
    initialDate = datetime.datetime.strptime(initialDate, '%Y-%m-%dT%H:%M')
    finalDate = datetime.datetime.strptime(finalDate, '%Y-%m-%dT%H:%M')
    lst = om.values(analyzer, initialDate, finalDate)

    totearthquakes = lt.size(lst)

    events = 0
    for lstdate in lt.iterator(lst):
        for j in lt.iterator(lstdate):

            time = j['time']


            events += 1
            dic[time] = {
                'time':time,
                'events':1,
                'details':j

            }
            lt.addFirst(final,dic[time])



    return totearthquakes, final, events
```

Este requerimiento se encarga de ver los eventos sísmicos mundiales ocurridos durante un intervalo de fechas específico.

Entrada	Fecha inicial del intervalo (en formato "%Y-%m-%dT%H:%M"). • Fecha final del intervalo (en formato "%Y-%m-%dT%H:%M").• La significancia mínima del evento (sig).
Salidas	El número total de eventos sísmicos ocurridos durante las fechas indicadas. • Todos los eventos ocurridos en el intervalo ordenados cronológicamente desde el más reciente al más antiguo.
Implementado (Sí/No)	Si. Implementado por Luis Sebastián Contreras

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Inicialización de estructuras de datos y conversiones de fechas:	O(1)
Obtención de elementos según rango de fechas:	O(log(M))
Ciclo anidado for para recorrer la lista de valores	O(M*M)
Creación y adición de elementos a una lista	O(1)
TOTAL	***O(M*M)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron fecha inicial 1999-03-21T05:00 y fecha final 2004-10-23T17:30.

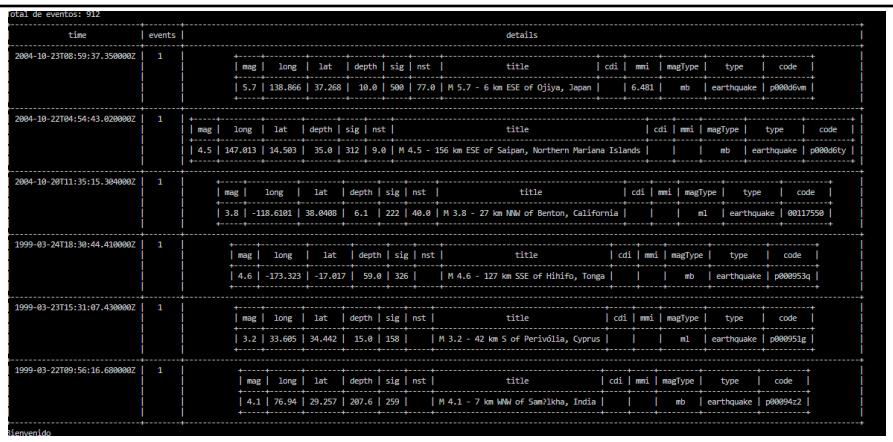
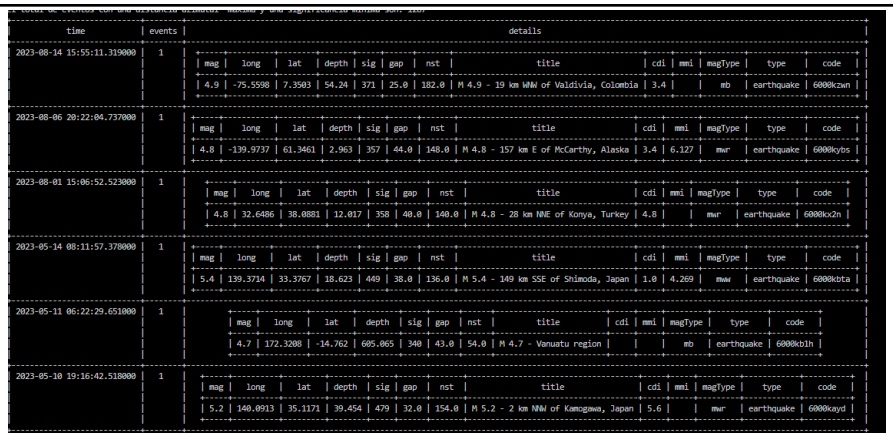
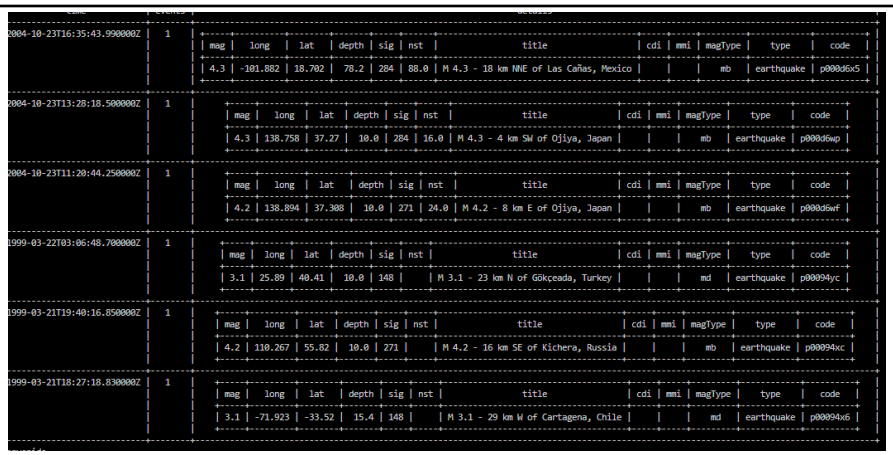
Procesadores	**12th Gen Intel(R) Core(TM) i5-1235U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

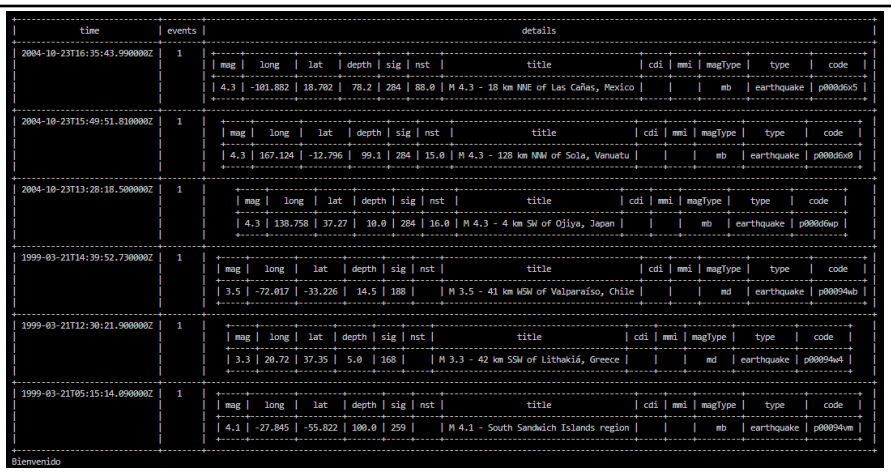
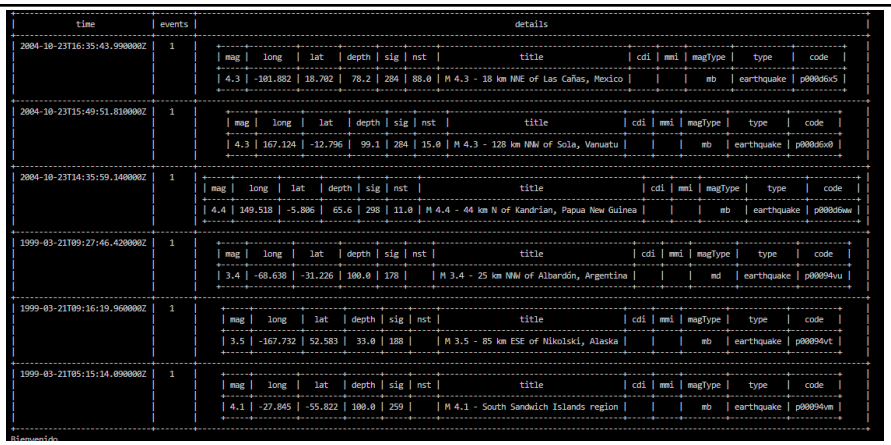
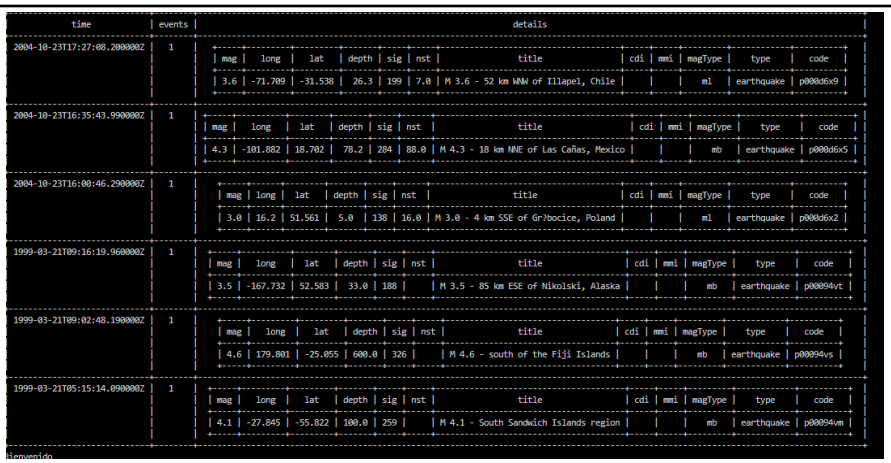
Entrada	Tiempo (ms)
small	12.81
5 pct	145.29
10 pct	207.78
20 pct	254.37
30 pct	805.64
50 pct	1454.28
80 pct	4837.21
large	12750.25

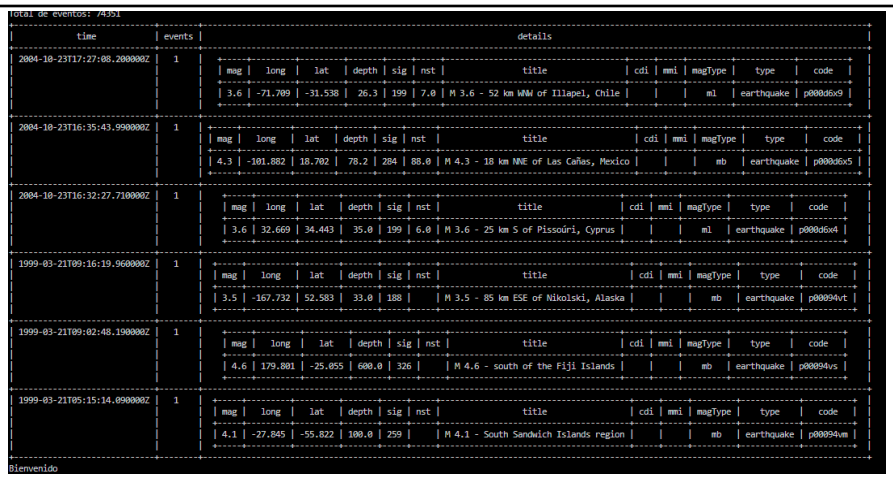
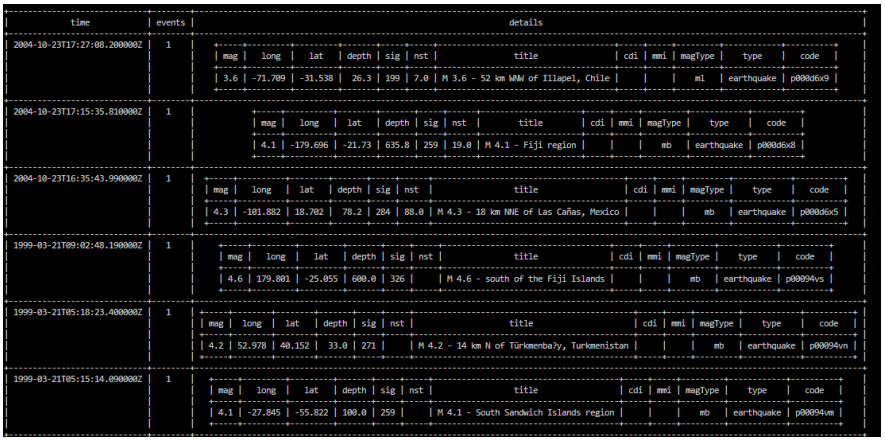
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo**

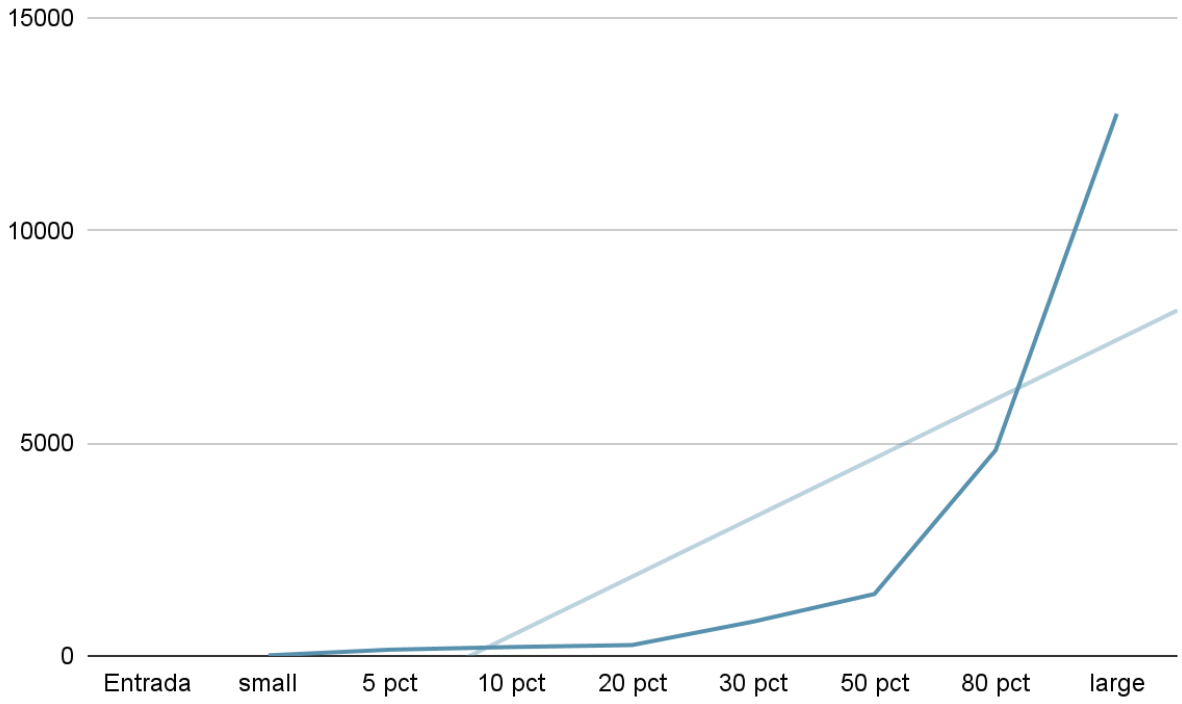
small		12.81
5 pct		145.29
10 pct		207.78

20 pct		254.37
30 pct		805.64
50 pct		1454.28

80 pct		4837.21
large		12750.25

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *arbo*l tiene una complejidad lineal O(N), la implementación incurre en ciclos anidados, lo cual lo puede hacer complejo, O(M^2).

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 exponencial esperado.

Requerimiento 2

```
def req_2(analyzer, initialmag, finalmag):
    """
    Retorna el numero de crimenes en un rago de fechas.
    """
    final = lt.newList('ARRAY_LIST')
    dic = {}
    lst = om.values(analyzer, initialmag, finalmag)
    totearthquakes = lt.size(lst)

    events = 0
    for lstdate in lt.iterator(lst):
        for j in lt.iterator(lstdate):

            mag = j['mag']


            events += 1
            dic[mag] = {
                'mag':mag,
                'events':1,
                'details':j

            }
            lt.addFirst(final,dic[mag])



    return totearthquakes, final, events
```

Descripción

Este requerimiento se encarga de ver los eventos sísmicos mundiales ocurridos durante un intervalo de fechas específico.

Entrada	Magnitud inicial del intervalo(mag) • Magnitud final del intervalo(mag)
Salidas	El número total de eventos sísmicos ocurridos dentro del intervalo indicado. • Todos los eventos ocurridos en el intervalo ordenados cronológicamente desde el más reciente al más antiguo.
Implementado (Sí/No)	Si. Implementado por Harold Piñeros Monroy

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Inicialización de estructuras de datos y conversiones de fechas:	O(1)
Obtención de elementos según rango demagnitud:	O(log(M))
Ciclo anidado for para recorrer la lista de valores	O(M*M)
Creación y adición de elementos a una lista	O(1)
TOTAL	***O(M*M)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron magnitud inicial 3.5 y magnitud final 6.5.

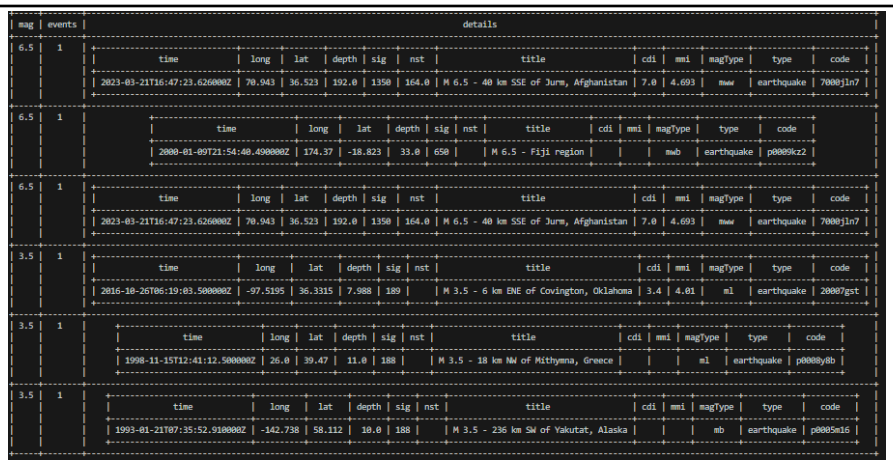
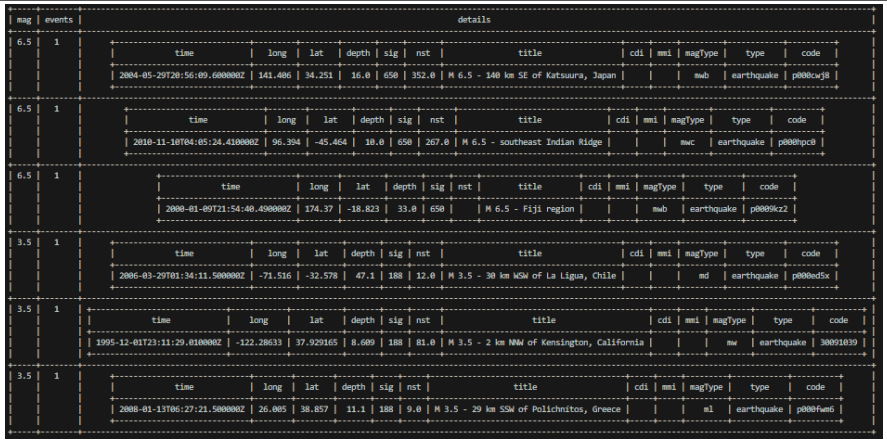
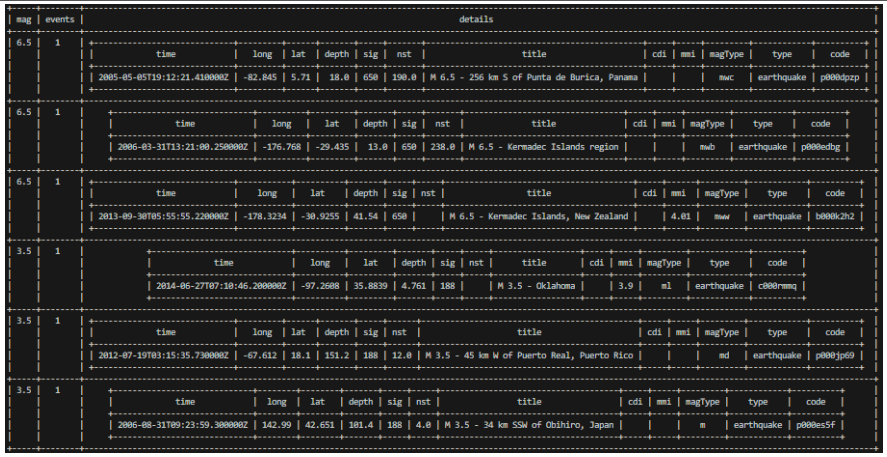
Procesadores	8th Gen Intel(R) Core(TM) i7-8565U
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

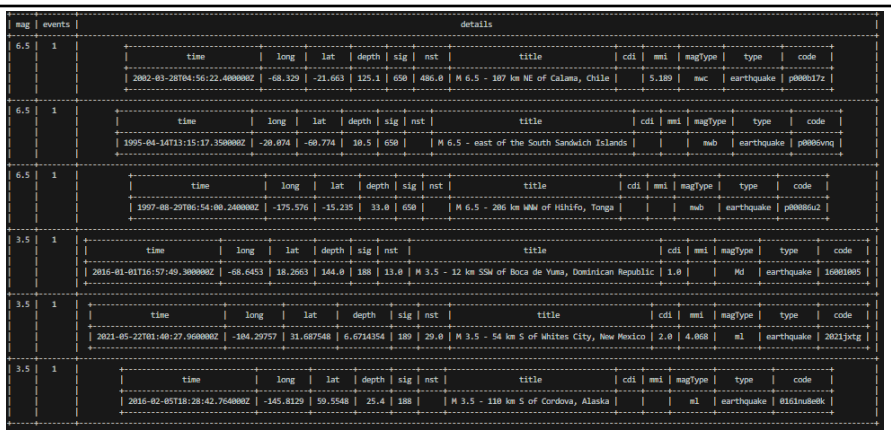
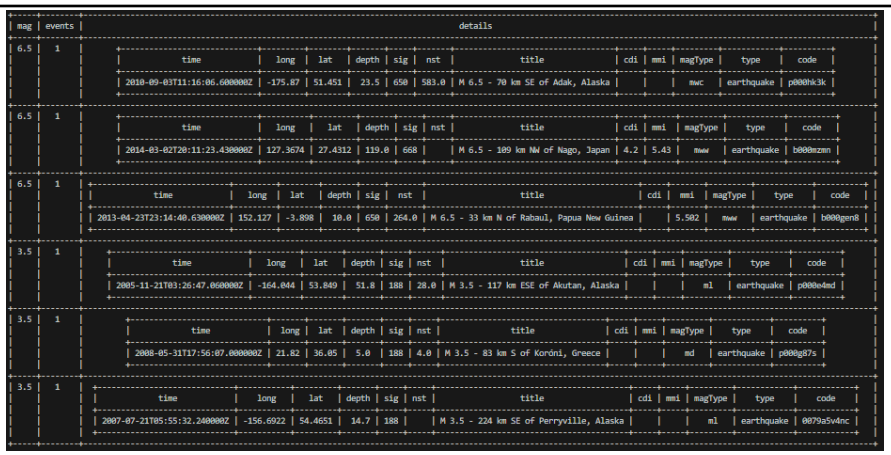
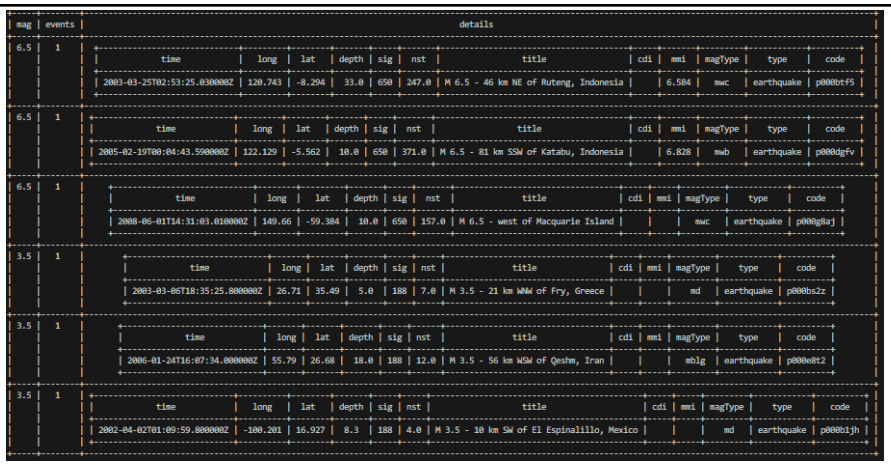
Entrada	Tiempo (ms)
small	79.15
5 pct	298.87
10 pct	1571.45
20 pct	4504.60
30 pct	6616.65
50 pct	14659.79
80 pct	49492.91
large	61633.50

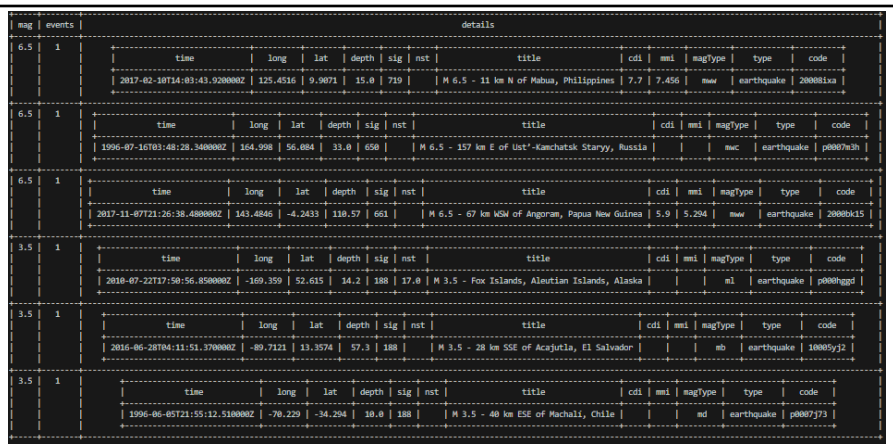
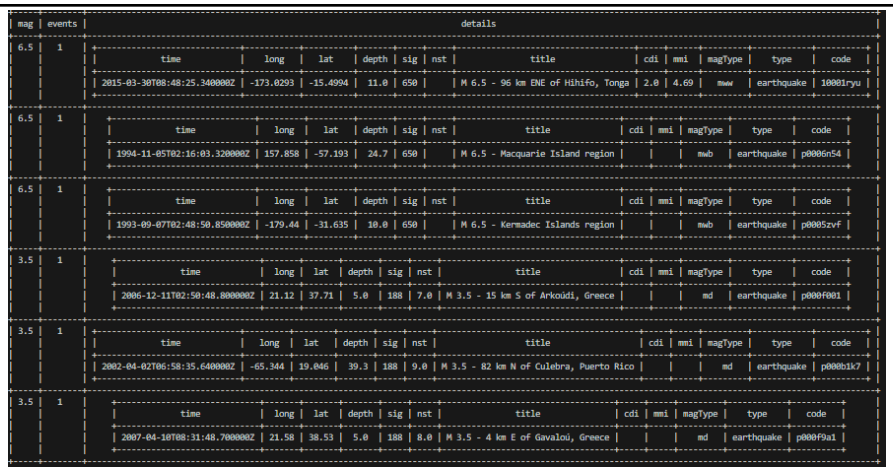
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo

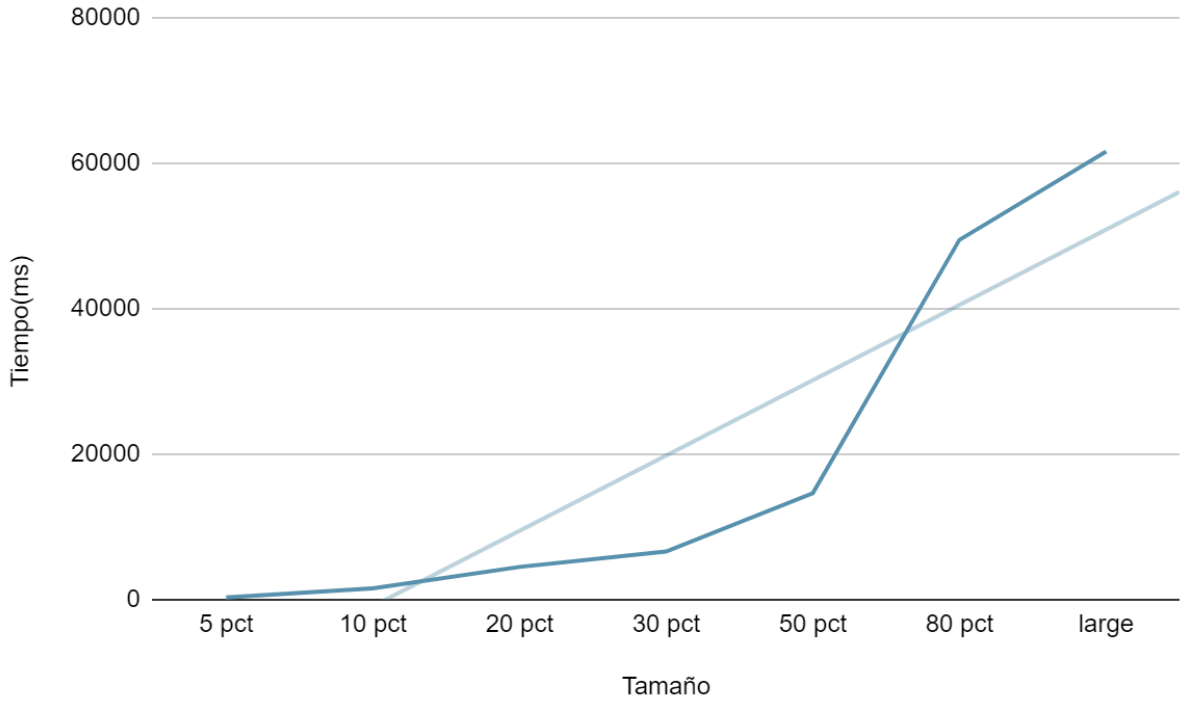
small		79.15
5 pct		298.87
10 pct		1571.45

20 pct		4504.60
30 pct		6616.65
50 pct		14659.79

80 pct		49492.91
large		61633.50

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *arbo*l tiene una complejidad lineal O(N), la implementación incurre en ciclos anidados, lo cual lo puede hacer complejo, O(Y^2).

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 exponencial esperado.

Requerimiento 3

```
def req_3(min_mag,max_depth,analyzer):
    """
    Función que soluciona el requerimiento 3
    """

    final = lt.newList('ARRAY_LIST')
    newLista = lt.newList('ARRAY_LIST')
    hp = heap.newHeap(compare_dicts)
    dic = {}
    data_structs = analyzer['mag']

    x = om.values(data_structs,float(min_mag), float(om.maxKey(data_structs)))

    for i in lt.iterator(x):

        f = om.values(i,float(om.minKey(i)),float(max_depth))
        for j in lt.iterator(f):
            for z in lt.iterator(j):
                if len(z['depth'])>0:

                 if float(z['depth'])>0:
                    lt.addFirst(newLista,z)

    sa.sort(newLista,compareReq3)
    a = lt.subList(newLista,1,17)
    for z in lt.iterator(a):
        time = z['time']



        dic[time] = {
            'time':time,
            'events':1,
            'details':z

             }
        lt.addLast(final,dic[time])
    return final
```

Descripción

Este requerimiento se encarga de o consultar los 15 eventos sísmicos más recientes ocurridos que superen una significancia mínima y que sean menores a una distancia azimutal indicada.

Entrada	• La magnitud mínima del evento (mag). • La profundidad máxima del evento (depth).
Salidas	El número total de eventos sísmicos registrados mayores a la magnitud y menores a la profundidad indicada • Los diez (10)

	eventos cronológicamente más recientes que cumplan con los parámetros especificados. Cada uno de los eventos en la consulta
Implementado (Sí/No)	Si. Implementado por Harold Piñeros Monroy

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de listas y diccionarios (tiempo constante): Esto incluye inicializar las estructuras de datos como listas y diccionarios.	O(1)
Buscar valores en un árbol binario	O(log(N))
Ciclo anidado for para recorrer la lista de valores	O(M)
Buscar valores en un árbol binario	O(log(M))
Ciclo anidado for para recorrer los valores	O(Y)
Verificación de condiciones y operaciones condicionales (tiempo constante)	O(1)
Ciclo anidado for para recorrer los valores	O(Y*Y)
Añadir al final de la lista los elementos	O(1)
Ordenamiento de la lista (sa.sort)	O(Y(log(Y))
TOTAL	***O(Y*(Y))***

Pruebas Realizadas

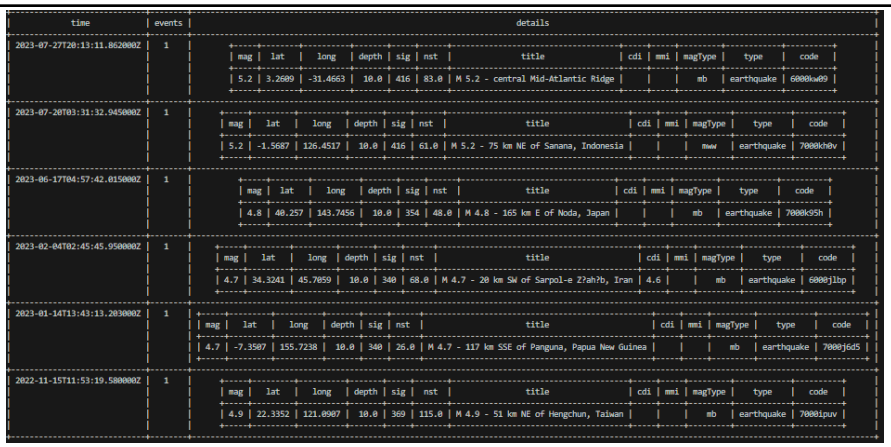
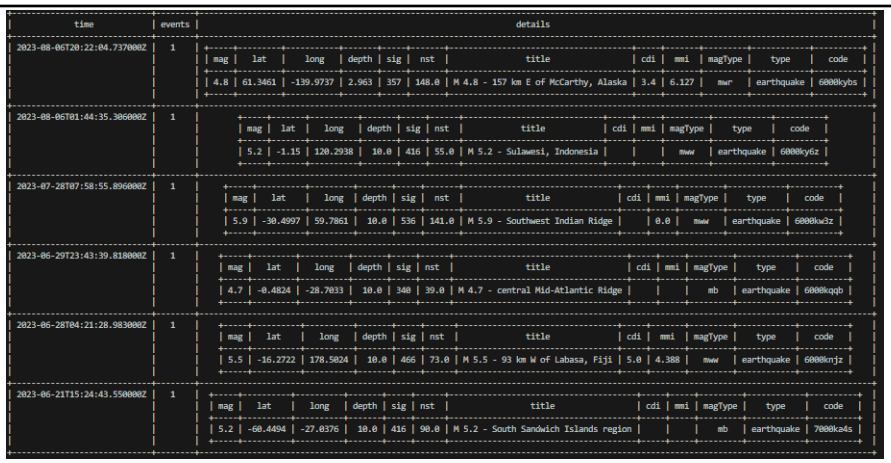
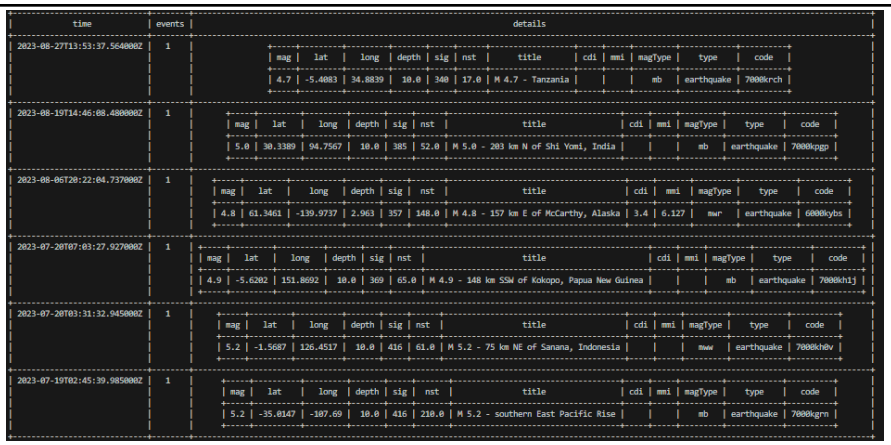
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron significancia mínima 300.0 y distancia azímutal máxima de 48.0.

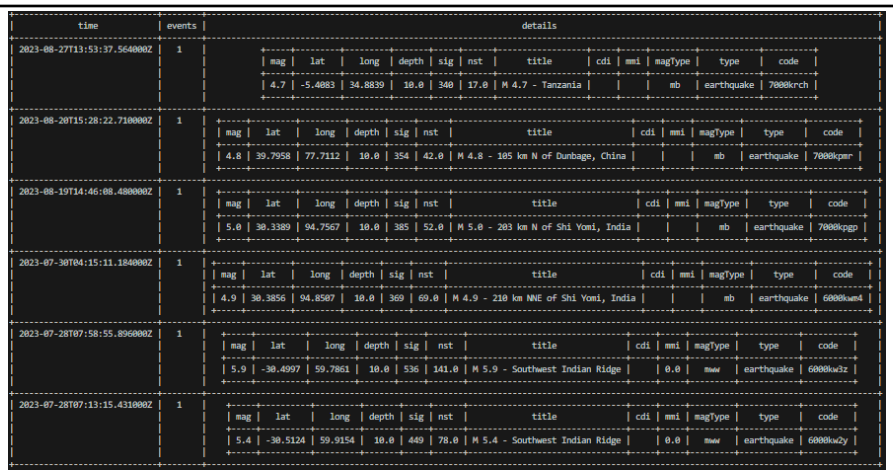
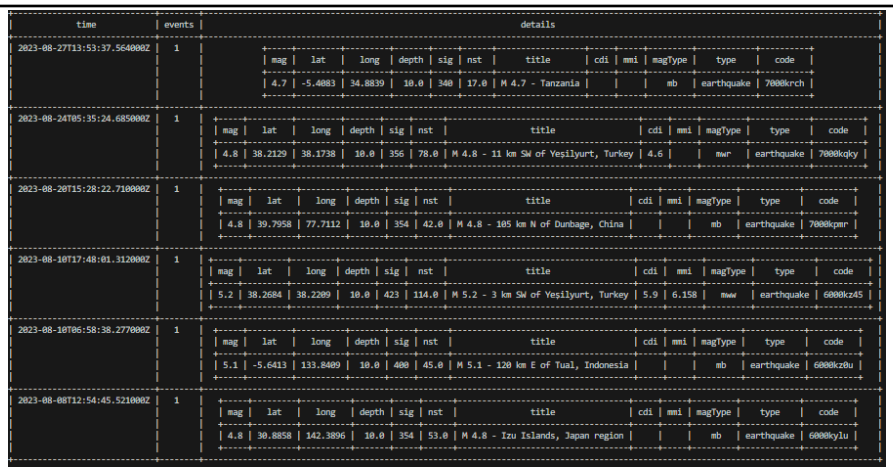
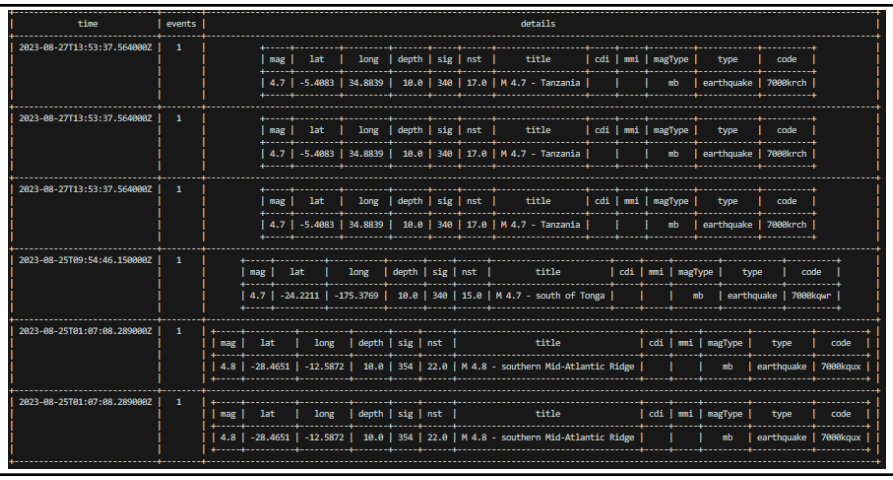
Procesadores	**8th Gen Intel(R) Core(TM) i7-8565U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

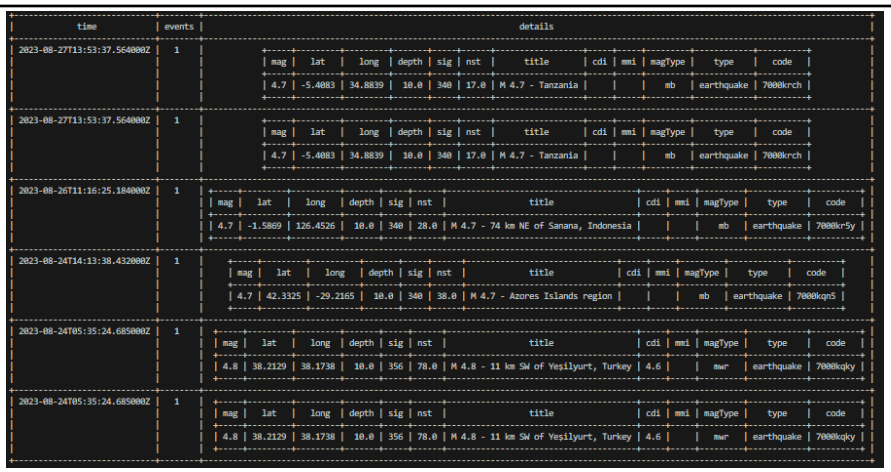
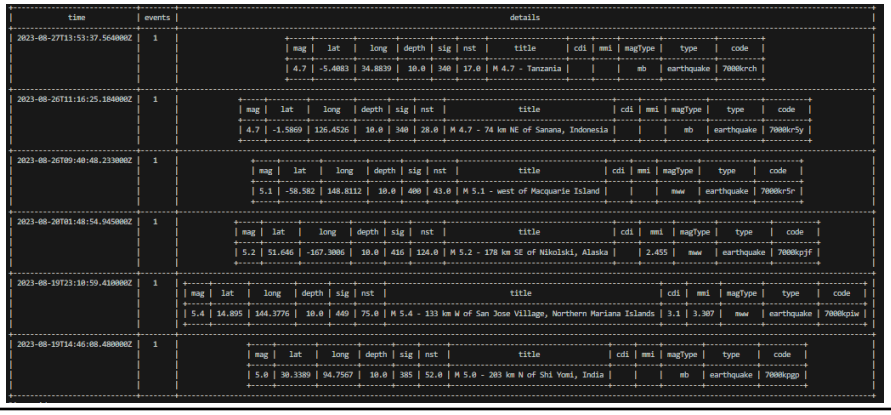
Entrada	**Tiempo (ms)**
small	8.81
5 pct	54.41
10 pct	130.05
20 pct	295.83
30 pct	512.05
50 pct	2416.42
80 pct	3438.75
large	6724.65

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

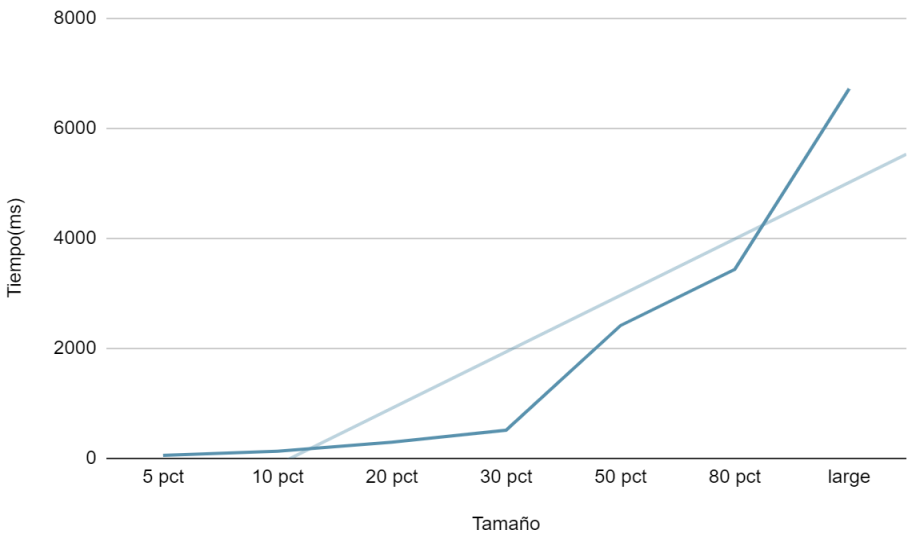
Muestra	Salida	Tiempo
small		8.81
5 pct		54.41
10 pct		130.05

20 pct		295.83
30 pct		512.05
50 pct		2416.42

80 pct		3438.75
large		6724.65

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *arbo*l tiene una complejidad lineal O(N), la implementación incurre en ciclos anidados, lo cual lo puede hacer complejo, O(Y^2).

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 exponencial esperado.

Requerimiento 4

Descripción

```
def req_4(sig,gap,analyzer):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento
    final = lt.newList('ARRAY_LIST')
    newLista = lt.newList('ARRAY_LIST')
    dic = {}
    data_structs = analyzer['sig']

    x = om.values(data_structs,float(sig), float(om.maxKey(data_structs)))

    for i in lt.iterator(x):

        f = om.values(i,float(om.minKey(i)),float(gap))
        for j in lt.iterator(f):
            for z in lt.iterator(j):
                if len(z['gap'])>0:

                 if float(z['gap'])>0:
                    lt.addFirst(newLista,z)
    sa.sort(newLista,compareDates3)
    a = lt.subList(newLista,1,17)
    for z in lt.iterator(a):
        time = z['time']



        dic[time] = {
            'time':time,
            'events':1,
            'details':z

             }
        lt.addLast(final,dic[time])
    return final
```

Este requerimiento se encarga de o consultar los 15 eventos sísmicos más recientes ocurridos que superen una significancia mínima y que sean menores a una distancia azimutal indicada.

Entrada	• La significancia mínima del evento (sig). • La distancia azimutal máxima del evento (gap).
Salidas	El número total de eventos sísmicos registrados mayores a la significancia y menores a la distancia azimutal indicada • Los quince (15) eventos cronológicamente más recientes que cumplan con los parámetros especificados. Cada uno de los eventos en la consulta
Implementado (Sí/No)	Si. Implementado por Luis Sebastián Contreras

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Creación de listas y diccionarios (tiempo constante): Esto incluye inicializar las estructuras de datos como listas y diccionarios.	O(1)
Buscar valores en un árbol binario	O(log(N))
Ciclo anidado for para recorrer la lista de valores	O(M)
Buscar valores en un árbol binario	O(log(M))
Ciclo anidado for para recorrer los valores	O(Y)
Verificación de condiciones y operaciones condicionales (tiempo constante)	O(1)
Ciclo anidado for para recorrer los valores	O(Y*Y)
Añadir al final de la lista los elementos	O(1)
Ordenamiento de la lista (sa.sort)	O(Y(log(Y))
TOTAL	***O(Y*(Y))***

Pruebas Realizadas

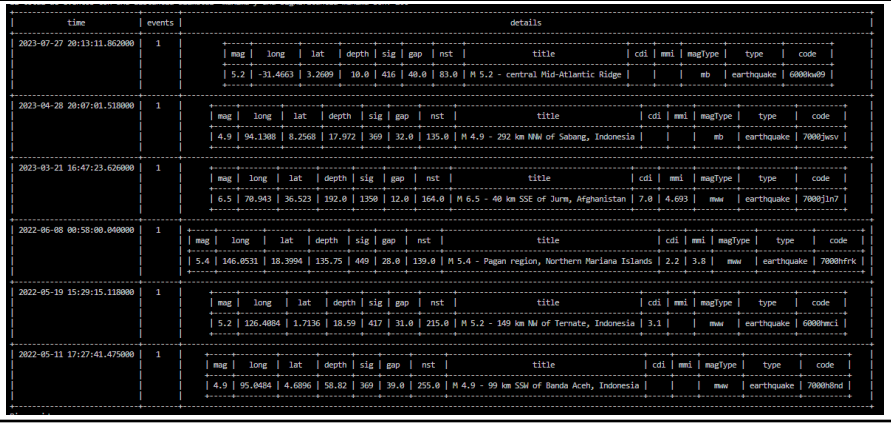
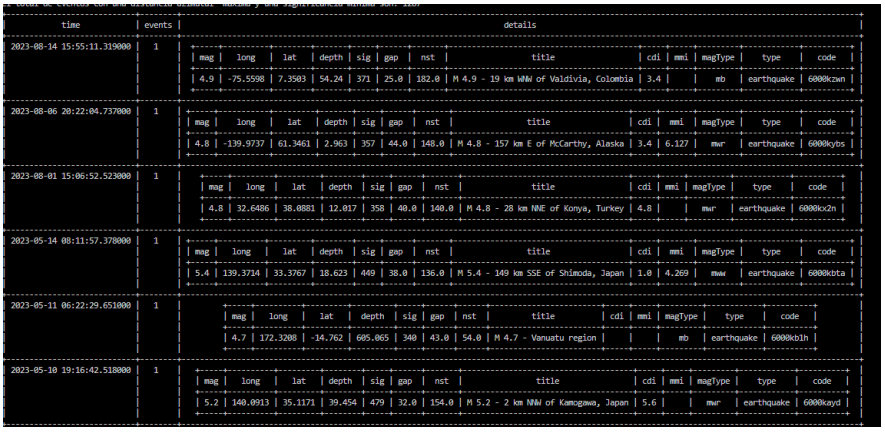
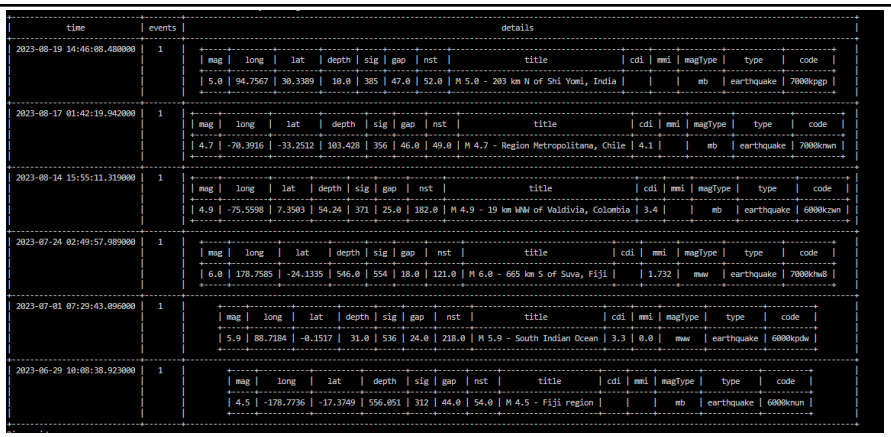
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron significancia mínima 300.0 y distancia azímutal máxima de 48.0.

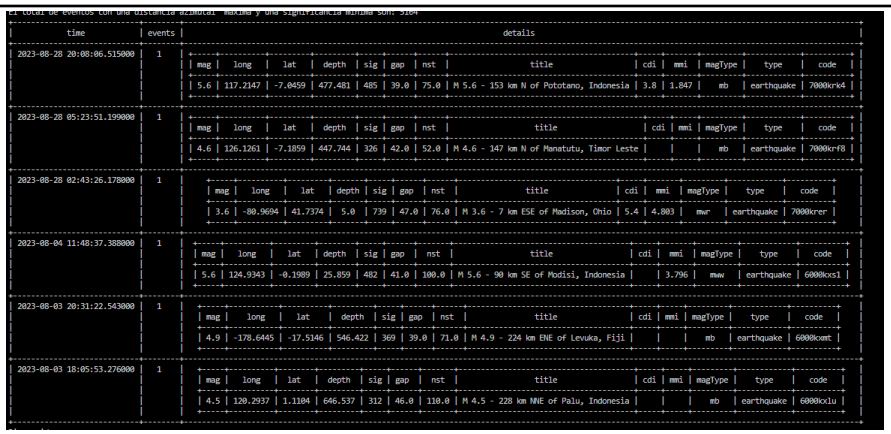
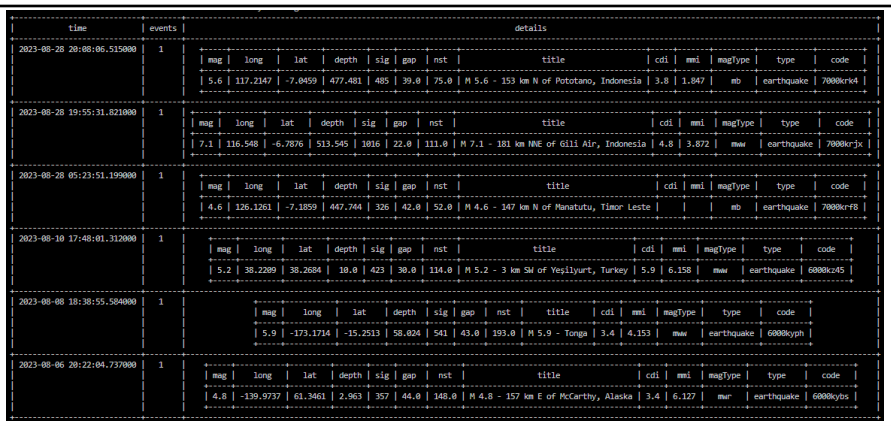
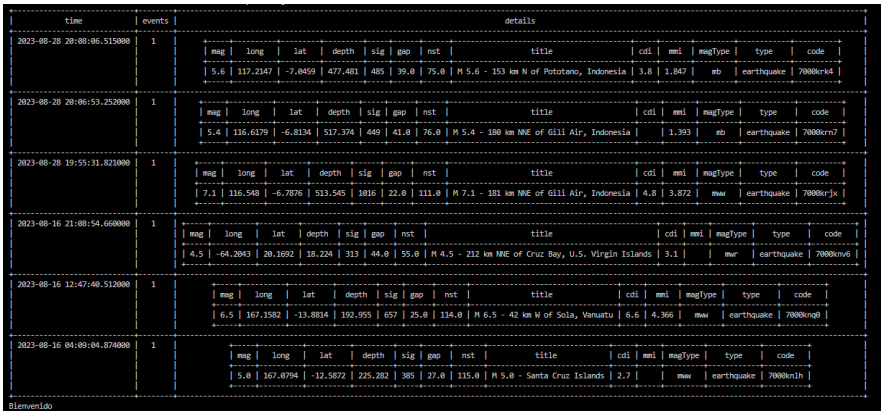
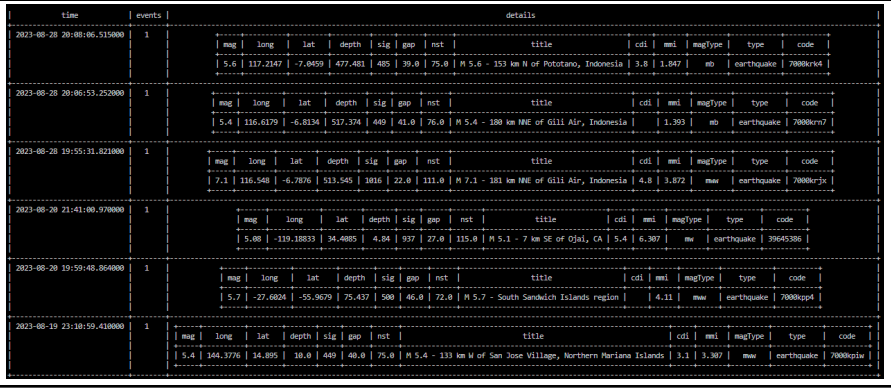
Procesadores	**12th Gen Intel(R) Core(TM) i5-1235U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

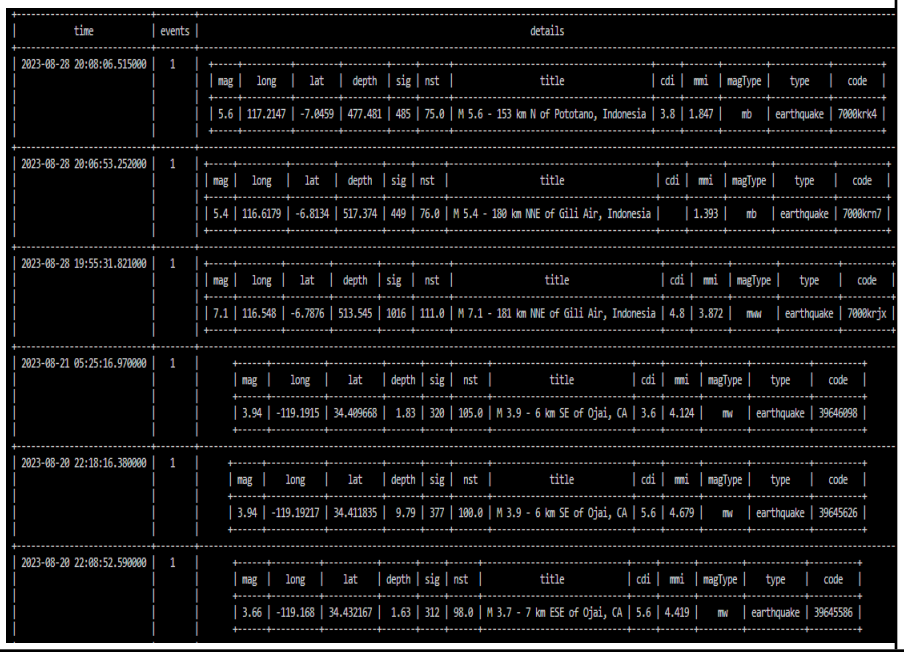
Entrada	Tiempo (ms)
small	10.58
5 pct	27.35
10 pct	77.78
20 pct	169.37
30 pct	283.64
50 pct	791.28
80 pct	12395.21
large	5907.25

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

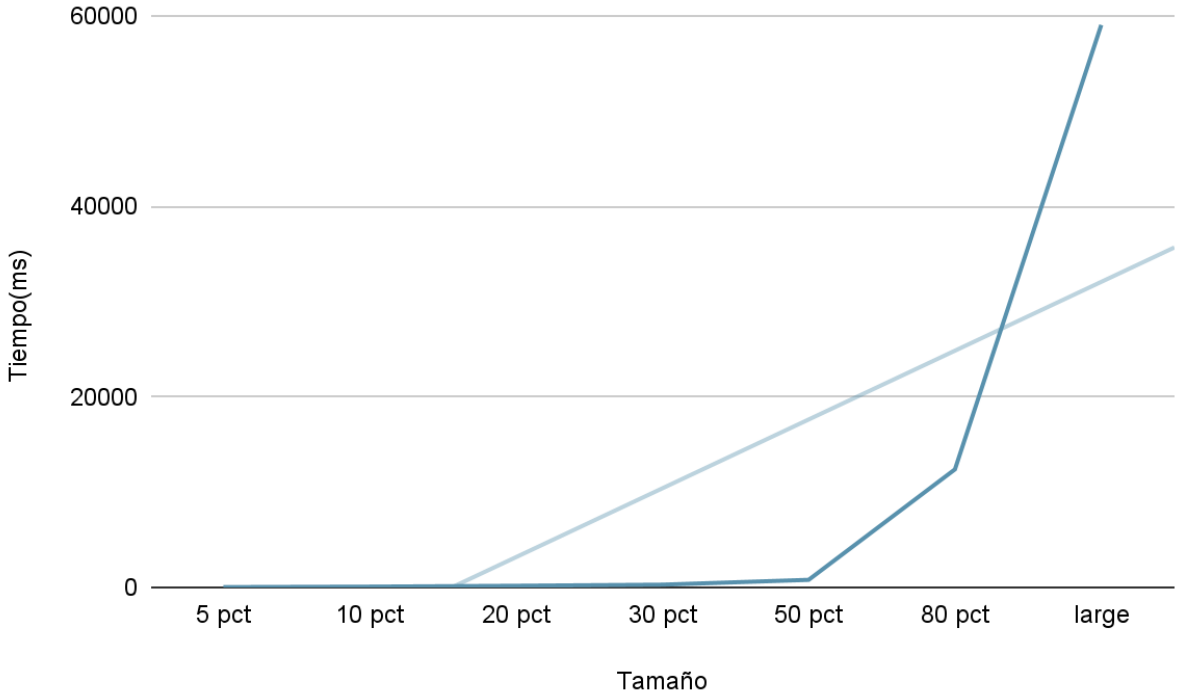
Muestra	Salida	Tiempo
small		10.58
5 pct		27.35
10 pct		77.78

20 pct		169.37
30 pct		283.64
50 pct		791.28
80 pct		12395.21

large		59070.25

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *arbo*l tiene una complejidad lineal O(N), la implementación incurre en ciclos anidados, lo cual lo puede hacer complejo, O(Y^2).

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 exponencial esperado.

Requerimiento 5

Descripción

```python
def req_6(depth,nst,analyzer):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6

    newLista = lt.newList('SINGLE_LINKED')
    data_structs = analyzer['depth']
    x = om.values(data_structs,depth, om.maxKey(data_structs))

    for i in lt.iterator(x):

        f = om.values(i,nst,om.maxKey(i))

        for j in lt.iterator(f):

            for z in lt.iterator(j):
                lt.addLast(newLista,z )


    merg.sort(newLista,compareDates2)

    return newLista
```

Este requerimiento se encarga de retornar los eventos sísmicos más recientes que superen una profundidad mínima y un número mínimo de estaciones de monitoreo. Sabemos que los datos están compuestos por un árbol el cual tiene una llave de profundidad y dentro del valor existe otro árbol que están el número mínimo de estaciones de monitoreo. Lo primero que hace es tomar los rangos de los dos árboles y ordenarlos de menor a mayor.

Entrada	La profundidad mínima del evento (depth). El número mínimo de estaciones que detectan el evento (nst).
Salidas	Un array list que cumpla con los requisitos anterios del mas reciente al mas antiguo.
Implementado (Sí/No)	Si. Implementado por Carlos Alberto Poveda Riaño

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
`newLista = lt.newList('SINGLE_LINKED')` Crea una lista encadenada	O(1)
`x = om.values(data_structs,depth, om.maxKey(data_structs))` Retorna todos los valores en una lista encadenada del arbol que se encuentren entre [keylo, keyhi]	O(n)
`for i in lt.iterator(x):` el valor de un arbol que es un arbol	O(n)
`f = om.values(i,nst,om.maxKey(i))` Retorna todos los valores en una lista encadenada del arbol que se encuentren entre [keylo, keyhi]	O(n)
`for z in lt.iterator(j):`	O(n)

`lt.addLast(newLista,z )` se agregan los elementos del arbol a una nueva lista	
`merg.sort(newLista,compareDates2)` ordena los datos del mas reciente al mas antiguo	O(n log(n))
TOTAL	**O(n log(n))**

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron profundidad: 23 y numero de estaciones 38.

Procesadores	**Intel(R) Core(TM) i5-6400 CPU**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

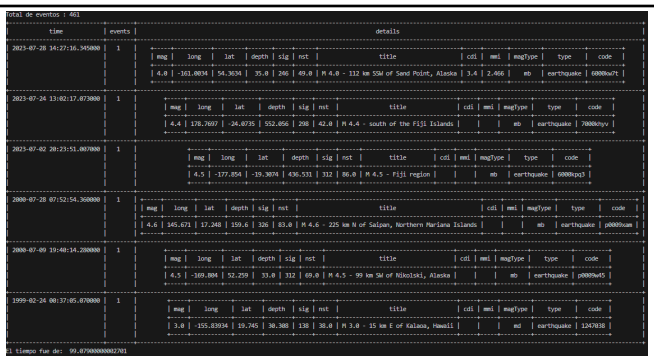
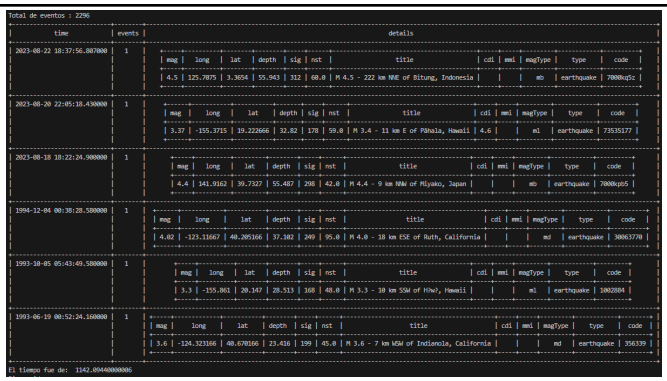
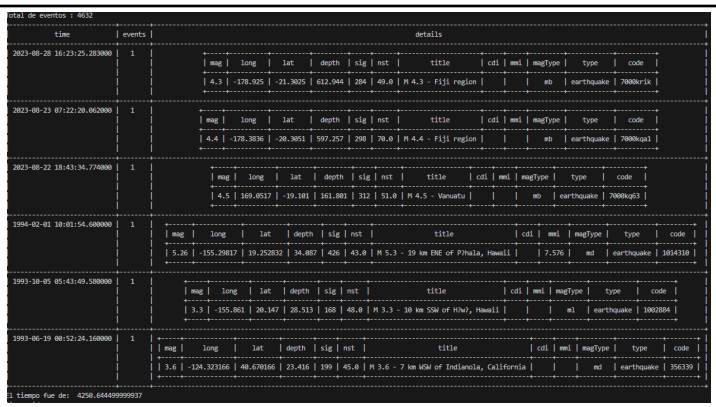
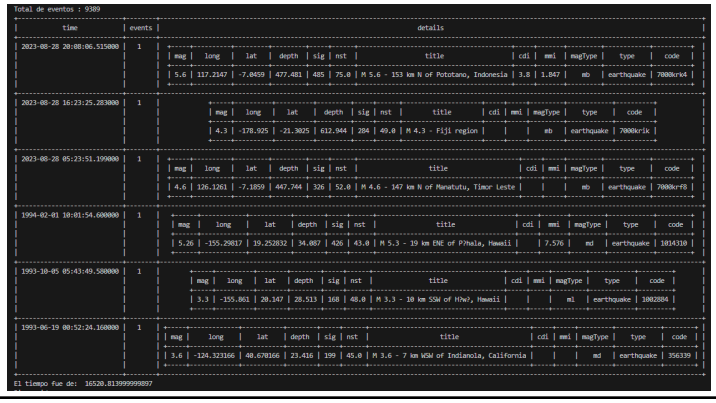
Carga de datos

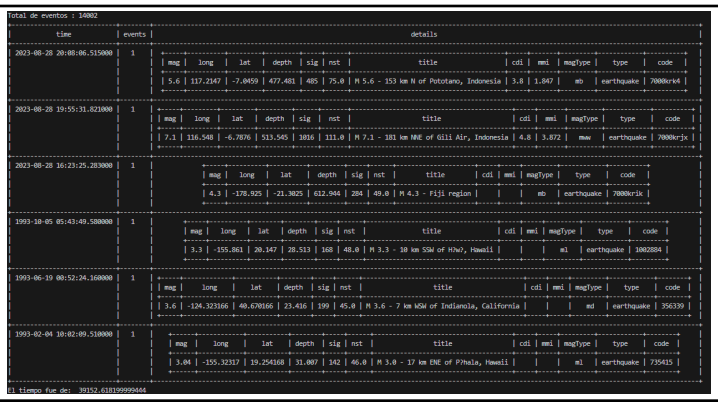
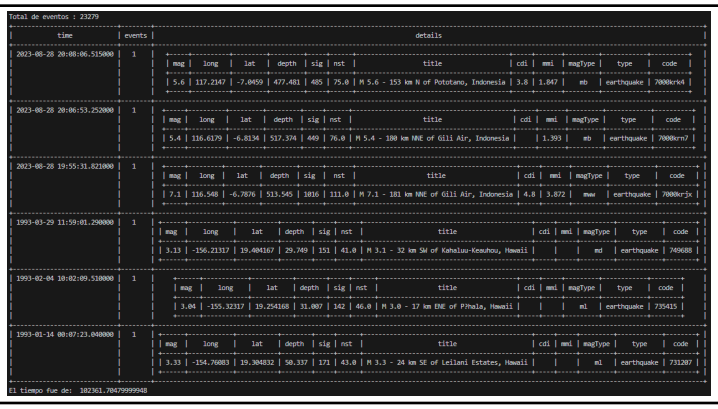
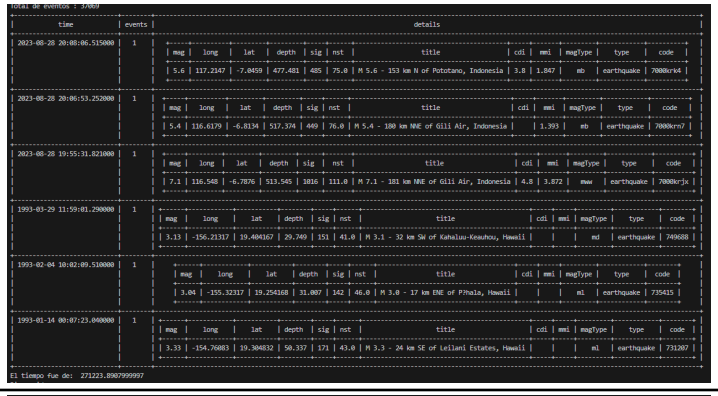
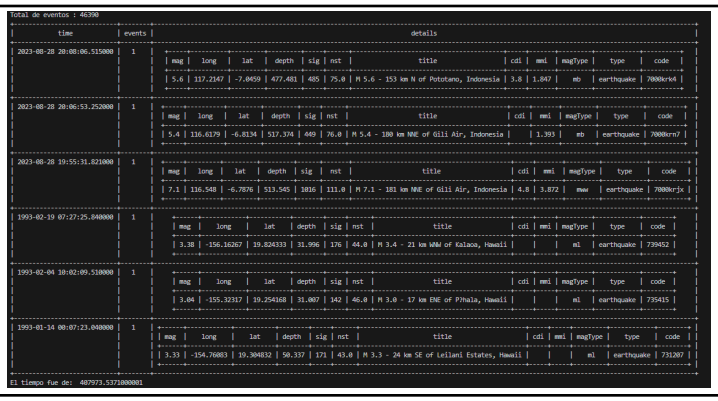
Entrada	Tiempo (ms)
small	1010.4
5 pct	5092.68
10 pct	11045.39
20 pct	20937.39
30 pct	34501.28
50 pct	59342.50
80 pct	95912.55
large	109279.5

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

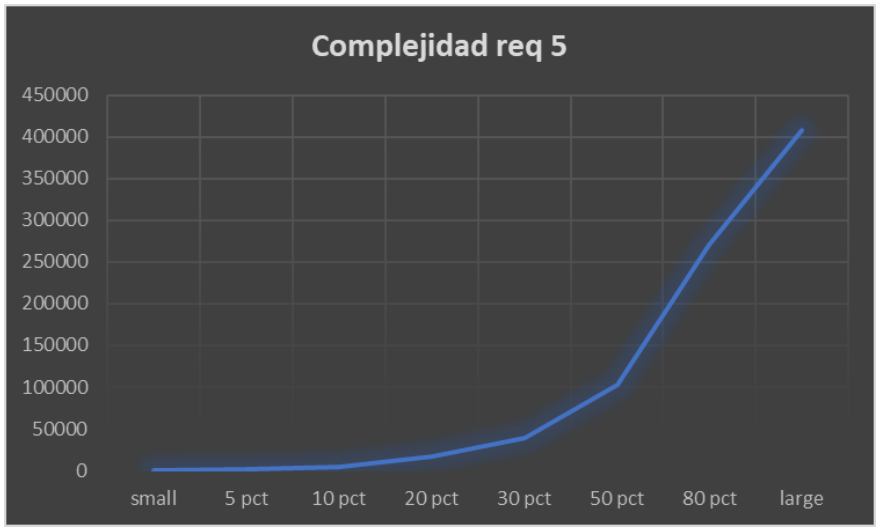
Muest	Salida	Tiempo

small		99.07
5 pct		1142.09
10 pct		4250.64
20 pct		16520.81

30 pct		39152.61
50 pct		102361.70
80 pct		271223.89
large		407973.53

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento tiene una complejidad **O(n log(n))** a pesar de que utilizamos estructura de árboles para optimizar el requerimiento, necesita de ordenamientos que influencia en la complejidad, al inicio tiene una complejidad constante, sin embargo, al acceso de los valores de los árboles y los ordenamientos tiende la complejidad a aumentar, De manera que se evidencia que la complejidad en el mejor va ser siempre **O(n log(n)).**

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 curvo esperado.

Requerimiento 6

Descripción

```
def req_5(year,lat,lon,radio, data_structs):
    a =0
    max ={}
    c = lt.newList('SINGLE_LINKED')
    array = lt.newList('ARRAY_LIST')
    array2 = lt.newList('ARRAY_LIST')
    temblor =m.get(data_structs,year)
    temblores = me.getValue(temblor)
    for j in lt.iterator(temblores):
        distancia = getdistance(lon,lat,j['long'],j['lat'])
        time = j['time']
        time2 = datetime.datetime.strptime(j['time'],'%Y-%m-%dT%H:%M:%S.%fZ')
        j['time'] = time2
        j['distancia'] = round(distancia,3)

        if j['distancia'] <radio:
            lt.addLast(array,j)
            if a <float(j['mag']):
                a =float(j['mag'])
                max = j
    lt.addLast(c,max)
    f = merg.sort(array,compareDates3)
    return f, c
```

Este requerimiento se encarga de retornar el sismo más significativo de un año dado dentro de un área circundante de una coordenada GPS designada, y los N eventos sísmicos más próximos cronológicamente. Se toma el valor del año de una tabla hash y se hacen los respectivos cálculos para obtener la distancia y así organizarlos.

Entrada	El año relevante (en formato “%Y”). La Latitud de referencia para el área de eventos (lat). La longitud de referencia para el área de eventos (long). El radio [km] del área circundante (float). El número de los N eventos de magnitud más cercana a mostrar.
Salidas	el evento maximo y un array list de los temblores mas recientes
Implementado (Sí/No)	Si. Implementado por Carlos Alberto Poveda Riaño

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
`array = lt.newList('ARRAY_LIST')` `Se crea un array list`	O(1)
`temblor =m.get(data_structs,year)` `Retorna la pareja llave, valor, cuya llave sea igual a key`	O(1)

`temblor =m.get(data_structs,year)` `Retorna el valor de una pareja de un Map`	O(1)
`for j in lt.iterator(temblores):`	O(n)
`merg.sort(array,compareDates3)`	O(n(log(n))
`lt.addLast(array,j)` `agrega el elemento a la lista`	O(1)
TOTAL	***O(n(log(n)))***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron año: 2022, latitud: 4.674, longitud: 74.068, radio :3000, numero de eventos: 5

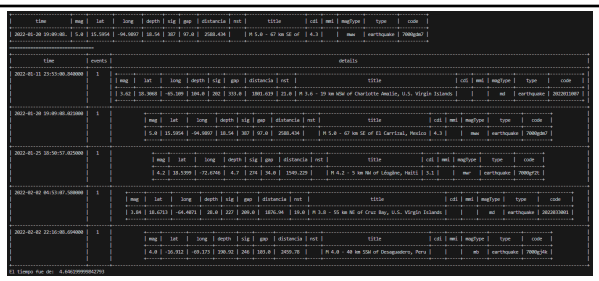
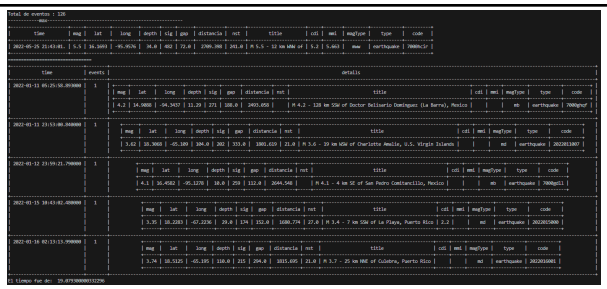
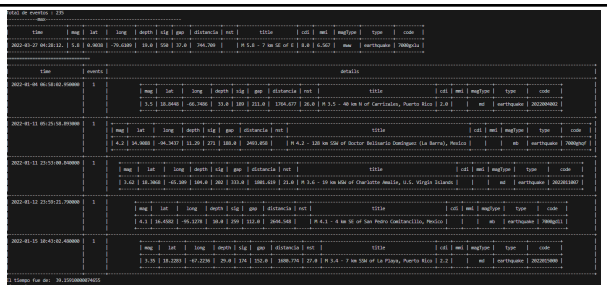
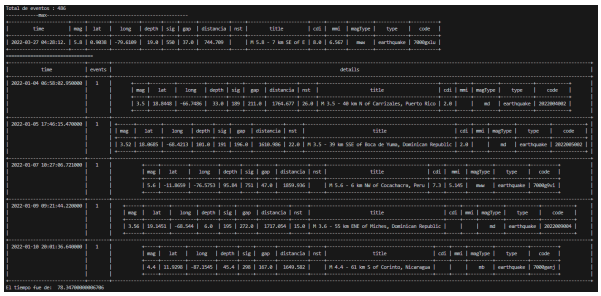
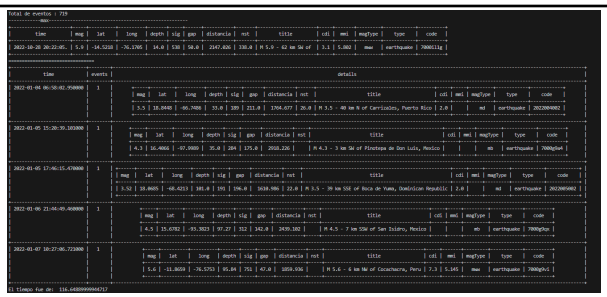
Procesadores	**Intel(R) Core(TM) i5-6400 CPU**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

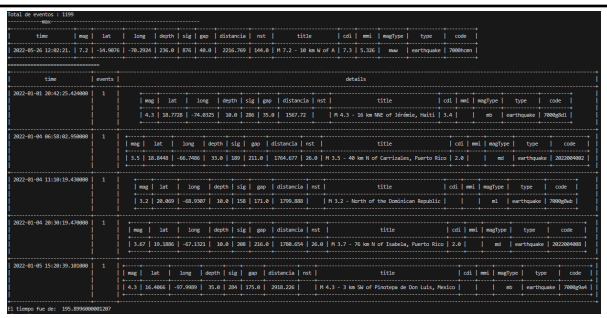
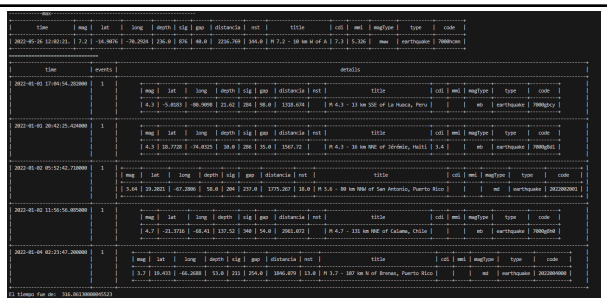
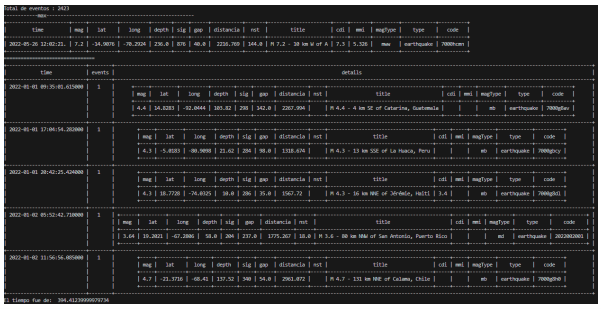
Entrada	**Tiempo (ms)**
small	1010.4
5 pct	5092.68
10 pct	11045.39
20 pct	20937.39
30 pct	34501.28
50 pct	59342.50
80 pct	95912.55
large	109279.5

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

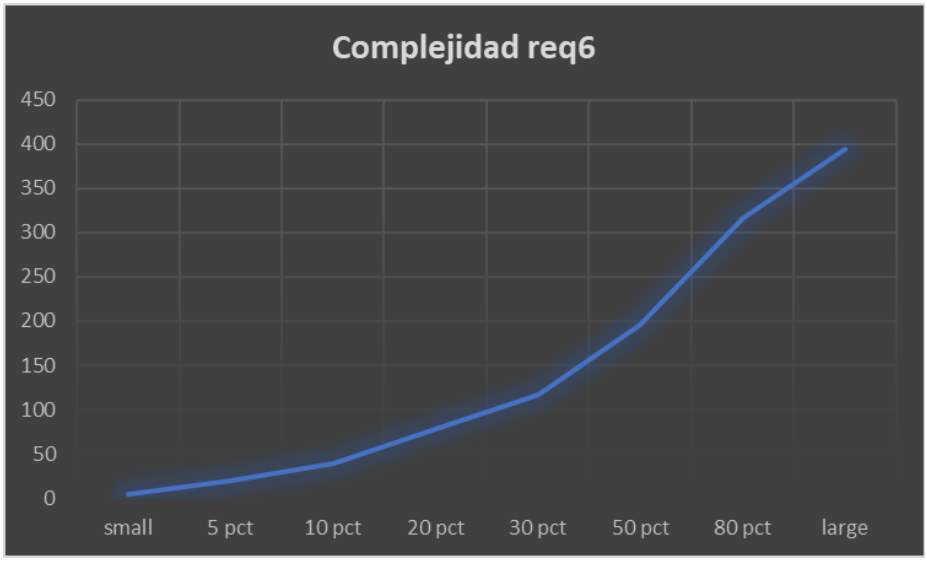
Muestra	**Salida**	**Tiemp**

small		4.64
5 pct		19.03
10 pct		39.15
20 pct		78.34
30 pct		116.64

50 pct		195.89
80 pct		316.86
large		394.41

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

En este requerimiento se utilizó la tabla hash lo que permitió la accesibilidad de los datos más rápido, aunque el ordenamiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace una búsqueda en la tabla de hash si el elemento hace parte del mapa. Sin embargo, al ordenarlo y al hacer un ciclo la complejidad es 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.

Requerimiento 7

Descripción

```
def req_7_histogram(year, title, prop, bins, analyzer):

    prop_values = lt.newList('ARRAY_LIST')


    year = me.getValue(m.get(analyzer['year'],int(year)))
    lista = lt.newList('ARRAY_LIST')

    prop_values2 = []


    # Iterar por las fechas
    for date in lt.iterator(year):



        if title in date['title']:
            if date[prop] is not None:
                lt.addLast(prop_values, date[prop])
                lt.addLast(lista, date)

    sa.sort(prop_values, compare_prop)
    sa.sort(lista, compareDates2)
    for a in lt.iterator(prop_values):
        prop_values2.append(a)

    mayor = lt.firstElement(prop_values)
    menor = lt.lastElement(prop_values)

    # Mostrar el resumen de eventos y rangos
    return lt.size(year), lt.size(lista), mayor, menor, prop_values2, lista
```

Este requerimiento se encarga de contabilizar 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).

Entrada	•El año relevante (en formato “%Y”). • El título de la región asociada (“title”). • La propiedad de conteo (magnitud, profundidad o significancia). • El número de segmentos o casillas (bins) en los que se divide el histograma.

Salidas	El número de eventos sísmicos dentro del periodo anual relevante. • El número de eventos sísmicos utilizados para crear el histograma de la propiedad. • Valor mínimo y valor máximo de la propiedad consultada en el histograma. • El histograma con la distribución de los eventos sísmicos según la propiedad. • Listado de los eventos que cumplen las condiciones de conteo para el histograma.
Implementado (Sí/No)	Si. Implementado por Luis Sebastián Contreras

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Inicialización de estructuras de datos y conversiones de fechas:	O(1)
Obtención de elementos según rango de fechas	O(N)
Ciclo anidado for para recorrer la lista de valores	O(N)
Creación y adición de elementos a una lista:	O(1)
Ordenar los elementos de la lista	O(Mlog(M))
Iterar los elementos de la lista	O(M)
Añadir a la lista un elemento	O(1)
TOTAL	***O(M(log(M))***

Pruebas Realizadas

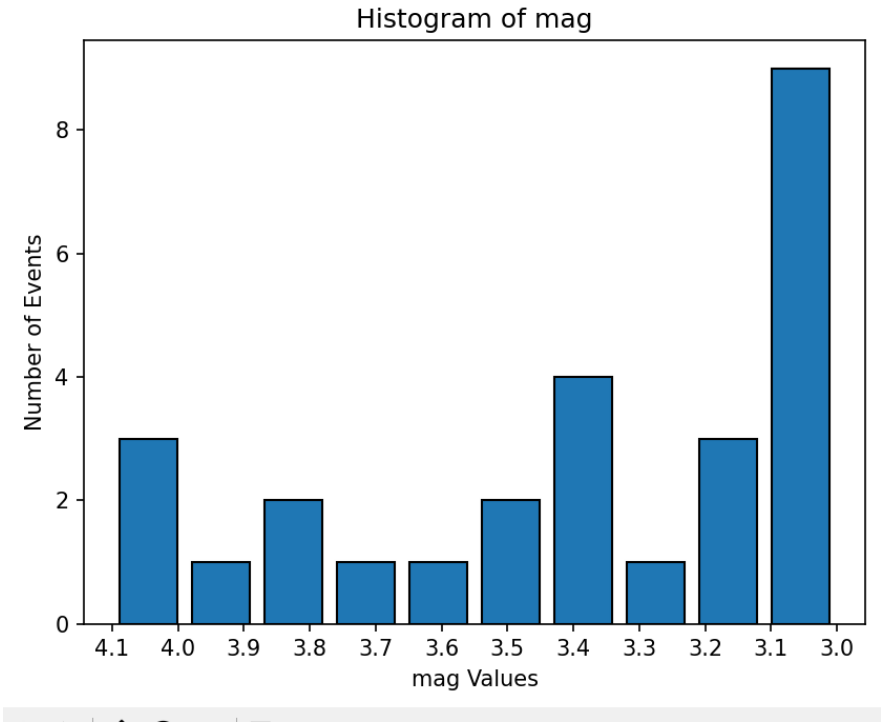
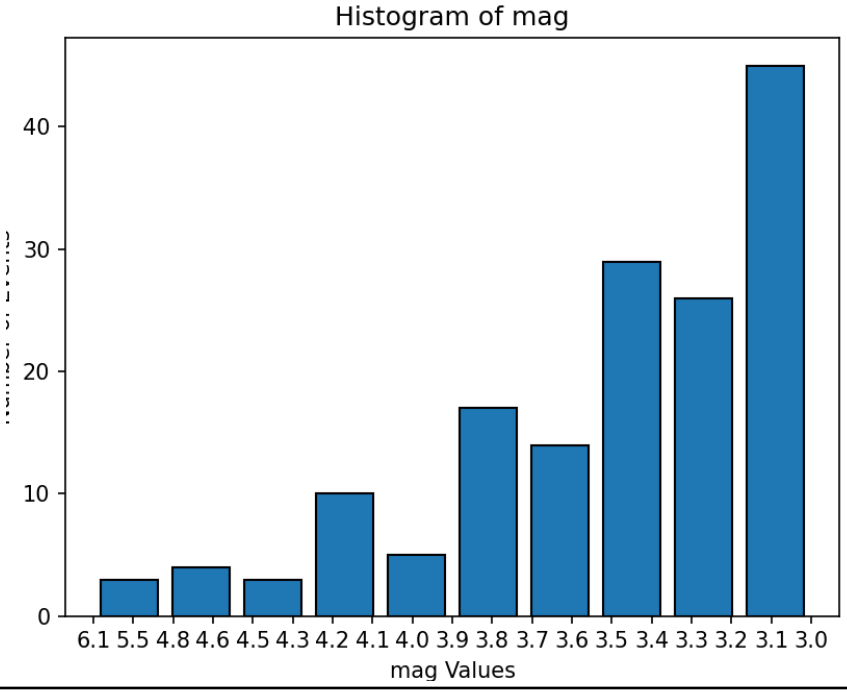
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron significancia mínima 300.0 y distancia azímutal máxima de 48.0.

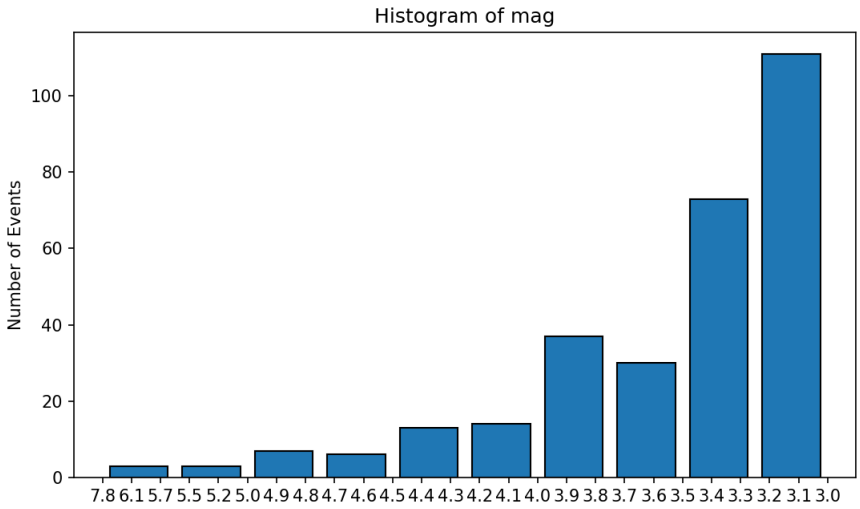
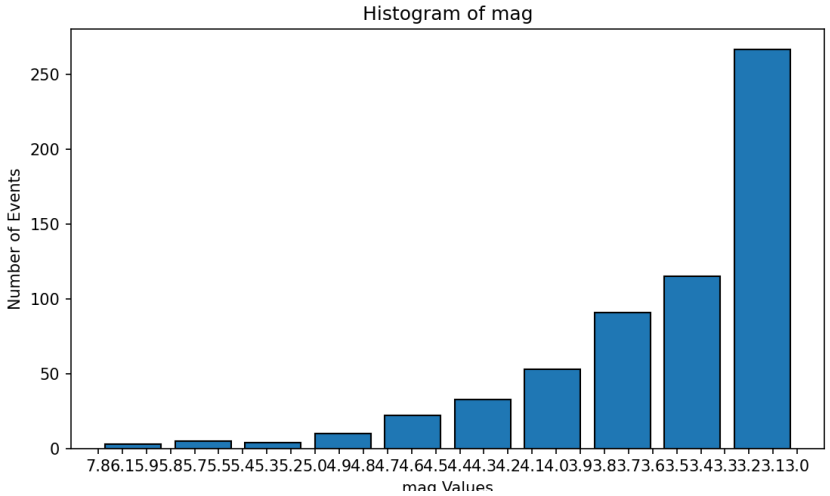
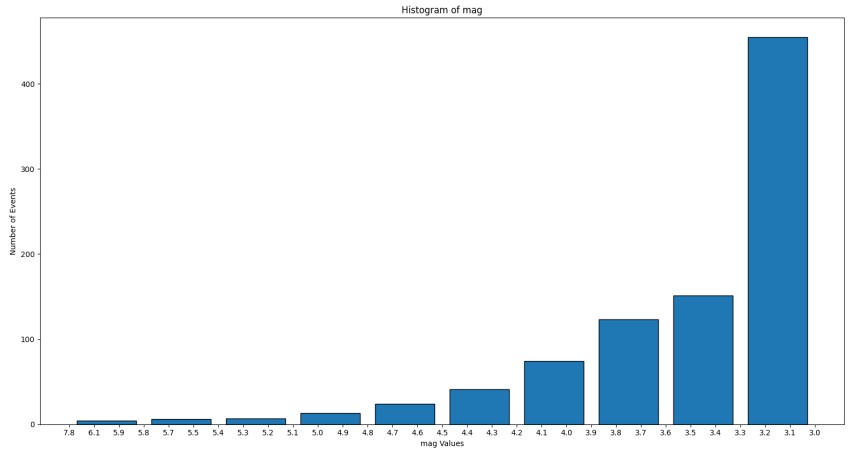
Procesadores	**12th Gen Intel(R) Core(TM) i5-1235U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

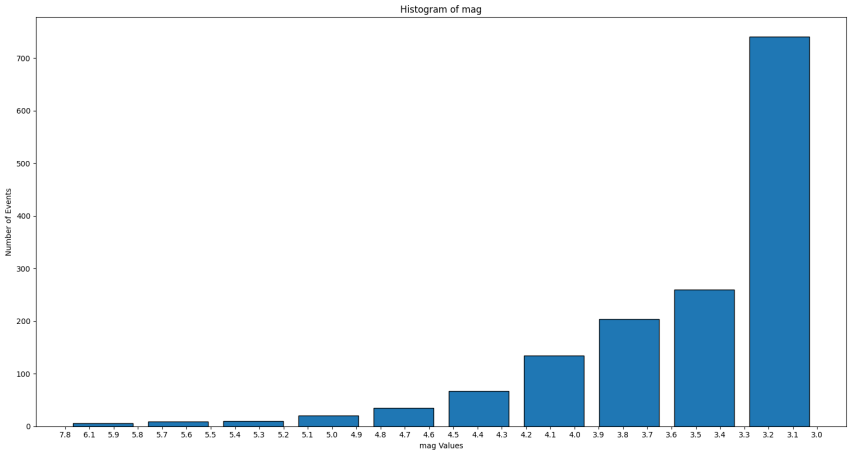
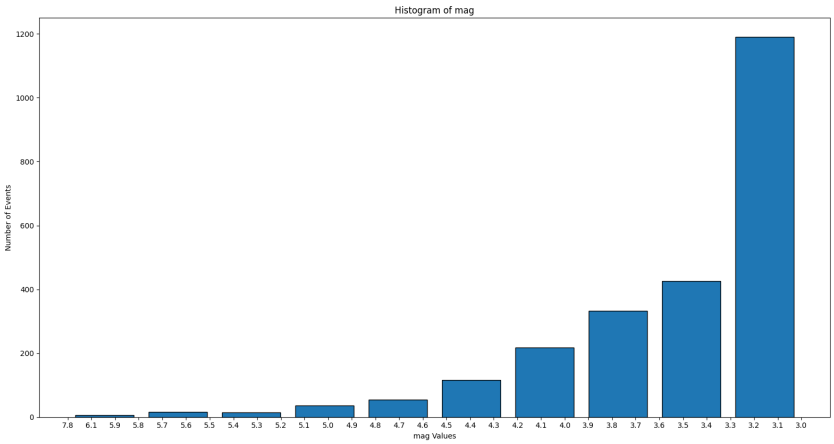
Entrada	Tiempo (ms)
small	0.75
5 pct	4.44
10 pct	12.70
20 pct	51.34
30 pct	36.59
50 pct	70.96
80 pct	126.23
large	167.297

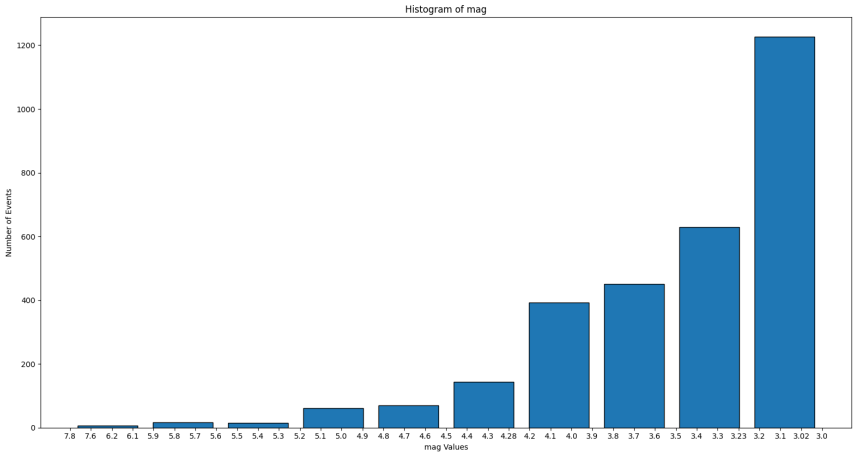
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo
small		0.75
5 pct		4.44

10 pct		12.70
20 pct		51.34
30 pct		36.59

50 pct		70.96
80 pct		126.23

large		167.297

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

La obtención de los elementos en un Hashmap puede resultar eficiente al tener una complejidad de N, sin embargo para poder obtener los datos correctamente se debe hacer un ordenamiento, esto puede conllevar a un complejidad de MlogM. Sin embargo, esto no refleja un cambio muy contundente, en

términos generales el código es eficiente.