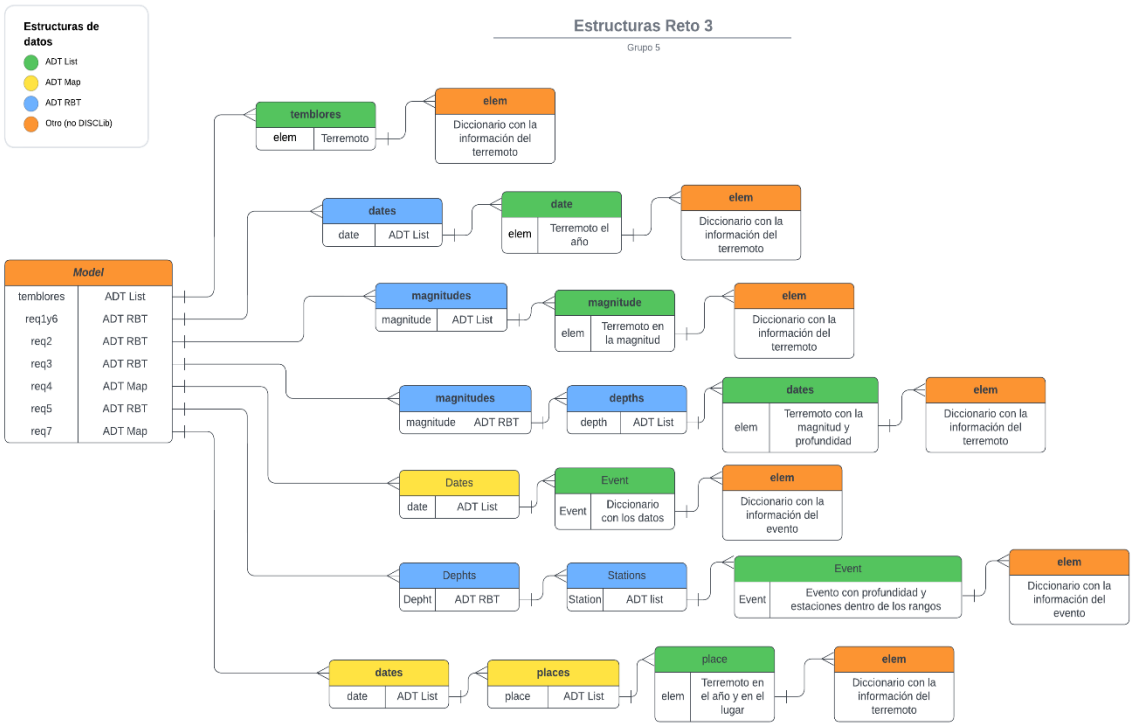
ANÁLISIS DEL RETO

Jhonny Armando Hortua Oyola, 202111749, j.hortuao@uniandes.edu.co

Gabriel Esteban González Carrillo, 202014375, g.gonzalezc@uniandes.edu.co

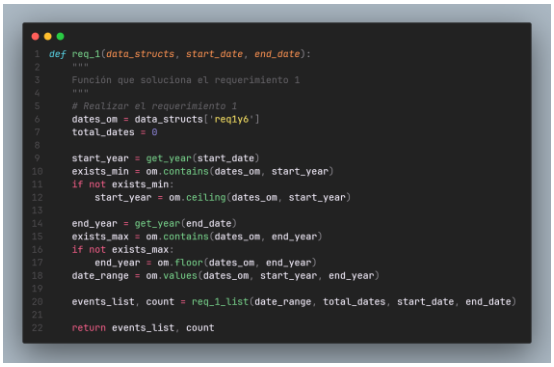
Adrian Esteban Velasquez Solano - 202222737, a.velasquezs@uniandes.edu.co

Estructuras de Datos



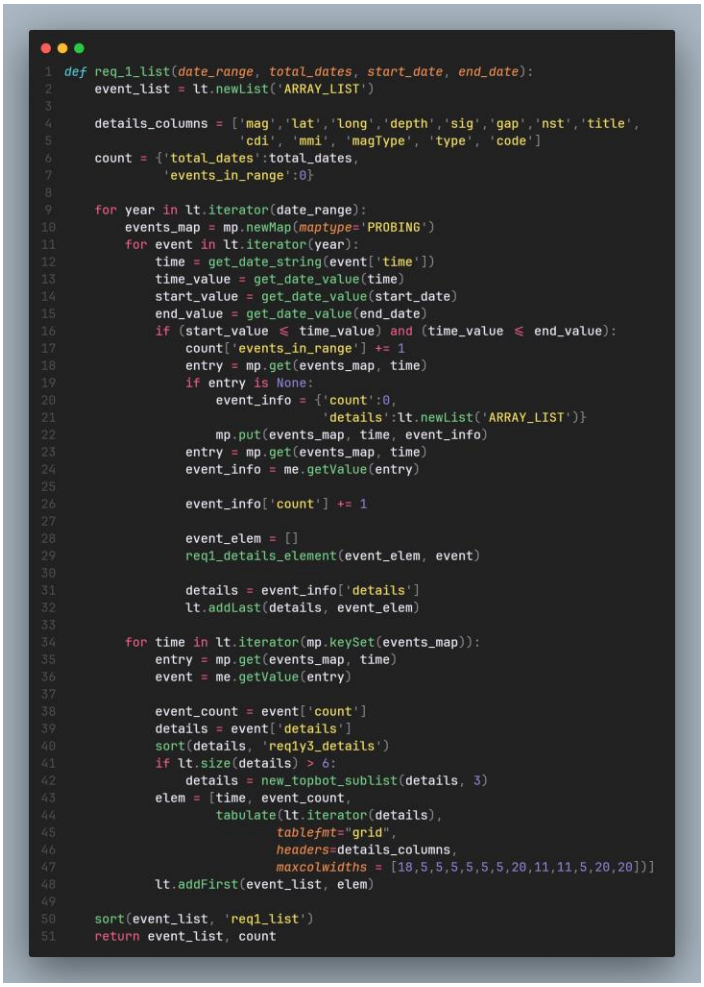
Para el reto 3, decidimos utilizar una mezcla de árboles, mapas y listas con el fin de reducir la complejidad de los requerimientos. Como se puede observar en el diagrama, la mayoría de las estructuras son árboles ordenados, mientras que algunas son mapas. Asimismo, principalmente por conveniencia, se utilizan ADT List para poder recorrer los eventos sísmicos con facilidad sin impactar mucho el tiempo de carga de los datos durante la inicialización. Por último, cada evento individual en todas las estructuras de datos es un diccionario de Python, con el cual se facilita el manejo de la información de cada evento por separado.

Requerimiento 1



```python
def req_1(data_structs, start_date, end_date):
    """
    Función que soluciona el requerimiento 1
    """
    # Realizar el requerimiento 1
    dates_om = data_structs['req1y6']
    total_dates = 0

    start_year = get_year(start_date)
    exists_min = om.contains(dates_om, start_year)
    if not exists_min:
        start_year = om.ceiling(dates_om, start_year)

    end_year = get_year(end_date)
    exists_max = om.contains(dates_om, end_year)
    if not exists_max:
        end_year = om.floor(dates_om, end_year)
    date_range = om.values(dates_om, start_year, end_year)

    events_list, count = req_1_list(date_range, total_dates, start_date, end_date)

    return events_list, count
```

F1.1: req_1()



```python
def req_1_list(date_range, total_dates, start_date, end_date):
    event_list = lt.newList('ARRAY_LIST')

    details_columns = ['mag','lat','long','depth','sig','gap','nst','title',
                       'cdi', 'mmi', 'magType', 'type', 'code']
    count = {'total_dates':total_dates,
             'events_in_range':0}

    for year in lt.iterator(date_range):
        events_map = mp.newMap(maptype='PROBING')
        for event in lt.iterator(year):
            time = get_date_string(event['time'])
            time_value = get_date_value(time)
            start_value = get_date_value(start_date)
            end_value = get_date_value(end_date)
            if (start_value <= time_value) and (time_value <= end_value):
                count['events_in_range'] += 1
                entry = mp.get(events_map, time)
                if entry is None:
                    event_info = {'count':0,
                                  'details':lt.newList('ARRAY_LIST')}
                    mp.put(events_map, time, event_info)
                entry = mp.get(events_map, time)
                event_info = me.getValue(entry)

                event_info['count'] += 1

                event_elem = []
                req1_details_element(event_elem, event)

                details = event_info['details']
                lt.addLast(details, event_elem)

        for time in lt.iterator(mp.keySet(events_map)):
            entry = mp.get(events_map, time)
            event = me.getValue(entry)

            event_count = event['count']
            details = event['details']
            sort(details, 'req1y3_details')
            if lt.size(details) > 6:
                details = new_topbot_sublist(details, 3)
            elem = [time, event_count,
                    tabulate(lt.iterator(details),
                             tablefmt="grid",
                             headers=details_columns,
                             maxcolwidths = [18,5,5,5,5,5,5,20,11,11,5,20,20])]
            lt.addFirst(event_list, elem)

    sort(event_list, 'req1_list')
    return event_list, count
```

F1.2: req_1_list()

```python
def req1_details_element(event_elem, event):
    event_elem.append(round(float(event['mag']),3))
    event_elem.append(round(float(event['lat']),3))
    event_elem.append(round(float(event['long']),3))
    event_elem.append(round(float(event['depth']),3))
    event_elem.append(event['sig'])
    gap = event['gap']
    if gap != '':
        event_elem.append(round(float(gap),3))
    else:
        event_elem.append(0.000)
    nst = event['nst']
    if nst != '':
        event_elem.append(float(nst))
    else:
        event_elem.append(1)
    event_elem.append(event['title'])
    cdi = event['cdi']
    if cdi != '':
        event_elem.append(round(float(cdi),3))
    else:
        event_elem.append('Unavailable')
    mmi = event['mmi']
    if mmi != '':
        event_elem.append(round(float(mmi),3))
    else:
        event_elem.append('Unavailable')
    event_elem.append(event['magType'])
    event_elem.append(event['type'])
    event_elem.append(event['code'])
```

F1.3: req1_details_element()

Descripción

Este requerimiento pretende buscar y ordenar todos los eventos sísmicos entre dos fechas. Para cumplir este objetivo, se emplea un árbol RBT de años, donde cada año contiene un ADT List de todos los eventos sísmicos de ese año. A pesar de que, hipotéticamente, el rango de búsqueda podría ser tan grande como N, en general la complejidad de los recorridos va a ser menor.

Entrada	Estructuras de datos del modelo, fecha inicial, fecha final.
Salidas	Lista de todos los eventos entre esas fechas
Implementado (Sí/No)	Si, Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
1.1	Obtener estructuras de datos	O(1)
1.1	Invocar 1.2	O(1)
1.2	Recorrer años en rango	O(y)
1.2	Recorrer eventos por año	O(e)
1.2	Recorrer eventos respuesta	O(k)
1.2	Sort	O(log(k))
Total		O(ye + k + log(k))

Pruebas Realizadas

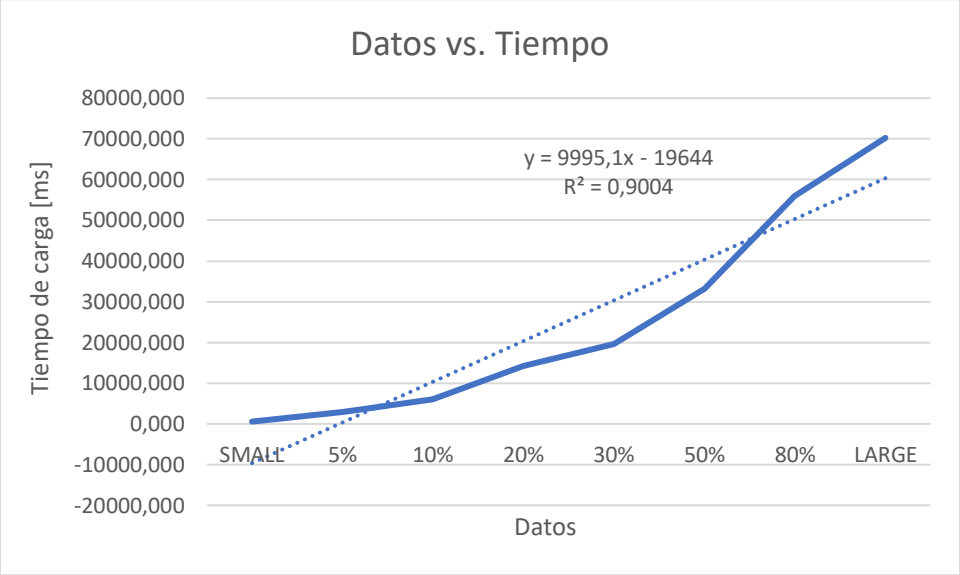
Las pruebas fueron realizadas con una máquina de las siguientes especificaciones. Los datos de entrada fueron **1999-03-21T05:00** y **2004-10-23T17:30**

Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	564.491
5%	2920.806
10%	6091
20%	14150.428
30%	19549.774
50%	33208.42
80%	55960.68
LARGE	70227.197

Graficas



Análisis

A pesar de que los tiempos de carga y de ejecución del requerimiento son más altos de lo esperado, es claro que la complejidad de este es cercana a la lineal. Esto se debe a la forma en la que se organizan los datos, ya que la cantidad de eventos por año es muy alta, y cuando el rango de fechas es muy grande, la complejidad se asemeja a O(N). Sin embargo, especialmente en rangos de fechas pequeños, se esperaría ver una gran ventaja en cuanto a los tiempos de ejecución del requerimiento.

Requerimiento 2

```python
def req_2(data_structs, start_mag, end_mag):
    """
    Función que soluciona el requerimiento 2
    """
    # Realizar el requerimiento 2
    mags_om = data_structs['req2']
    total_magnitudes = 0

    minimum = round(float(start_mag), 0)
    exists_min = om.contains(mags_om, minimum)
    if not exists_min:
        start_mag = om.ceiling(mags_om, minimum)

    maximum = round((float(end_mag)), 0)
    exists_max = om.contains(mags_om, maximum)
    if not exists_max:
        end_mag = om.floor(mags_om, maximum)

    mag_range = om.values(mags_om, minimum, maximum)

    events_list, count = req_2_list(mag_range, total_magnitudes, start_mag, end_mag)

    return events_list, count
```

F2.1: req_2()

```python
def req_2_list(mag_range, total_mags, start_mag, end_mag):
    event_list = lt.newList('ARRAY_LIST')

    details_columns = ['time','lat','long','depth','sig','gap','nst','title',
                       'cdi','mmi','magType','type','code']
    count = {'total_magnitudes':total_mags,
             'events_in_range':0}

    for mag in lt.iterator(mag_range):
        event_count = lt.size(mag)
        details = lt.newList()

        events_map = mp.newMap(maptype='PROBING')
        for event in lt.iterator(mag):
            magnitude = round(float(event['mag']), 3)

            if round(float(start_mag),3) <= magnitude and magnitude <= round(float(end_mag),3):
                count['events_in_range'] += 1
                entry = mp.get(events_map, magnitude)
                if entry is None:
                    event_info = {'count':0,
                                  'details':lt.newList('ARRAY_LIST')}
                    mp.put(events_map, magnitude, event_info)
                else:
                    event_info = me.getValue(entry)

                event_info['count'] += 1
                event_elem = []
                req2_details_element(event_elem, event)
                details = event_info['details']
                lt.addLast(details, event_elem)

        for magnitude in lt.iterator(mp.keySet(events_map)):
            entry = mp.get(events_map, magnitude)
            event = me.getValue(entry)

            event_count = event['count']
            details = event['details']
            sort(details, 'req2_details')
            if lt.size(details) > 6:
                details = new_topbot_sublist(details, 3)
            elem = [magnitude, event_count,
                tabulate(lt.iterator(details),
                         tablefmt="grid",
                         headers=details_columns,
                         maxcolwidths = [18,5,5,5,5,5,5,20,11,11,5,20,20])]
            lt.addFirst(event_list, elem)

    sort(event_list, 'req2_list')
    return event_list, count
```

F2.2: req_2_list()

```python
def req2_details_element(event_elem, event):
    event_elem.append(get_date_string(event['time']))
    event_elem.append(round(float(event['lat']),3))
    event_elem.append(round(float(event['long']),3))
    event_elem.append(round(float(event['depth']),3))
    event_elem.append(event['sig'])
    gap = event['gap']
    if gap != '':
        event_elem.append(round(float(gap),3))
    else:
        event_elem.append(0.000)
    nst = event['nst']
    if nst != '':
        event_elem.append(float(nst))
    else:
        event_elem.append(1)
    event_elem.append(event['title'])
    cdi = event['cdi']
    if cdi != '':
        event_elem.append(round(float(cdi),3))
    else:
        event_elem.append('Unavailable')
    mmi = event['mmi']
    if mmi != '':
        event_elem.append(round(float(mmi),3))
    else:
        event_elem.append('Unavailable')
    event_elem.append(event['magType'])
    event_elem.append(event['type'])
    event_elem.append(event['code'])
```

F2.3: req2_details_element()

Descripción

Este requerimiento pretende encontrar y ordenar todos los eventos sísmicos entre dos magnitudes. Para esto se utiliza un árbol RBT de magnitudes, donde cada nodo tiene un ADT List con todos los eventos de dicha magnitud. A pesar de que, hipotéticamente, el rango de búsqueda podría ser tan grande como N, en general la complejidad de los recorridos va a ser menor.

Entrada	Estructuras de datos del modelo, magnitud inicial, magnitud final
Salidas	Lista de todos los eventos entre las magnitudes.
Implementado (Sí/No)	Si, Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
2.1	Obtener estructuras de datos	O(1)
2.1	Invocar 2.2	O(1)
2.2	Recorrer magnitudes	O(m)
2.2	Recorrer eventos de la magnitud	O(e)
2.2	Recorrer eventos respuesta	O(k)
2.2	sort	O(log(k))
Total		O(me + k + log(k))

Pruebas Realizadas

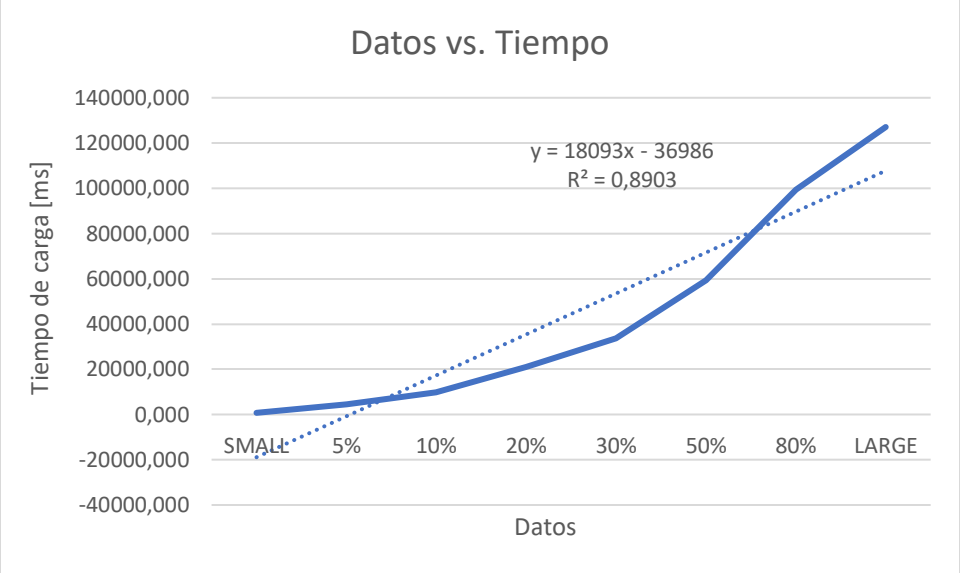
Las pruebas fueron realizadas con una máquina de las siguientes especificaciones. Del estudiante 2

Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	738.126
5%	4612.538
10%	9689.285
20%	21070.088
30%	33540.283
50%	59285.061
80%	99456.599
LARGE	127072.426

Graficas



Análisis

De forma similar al requerimiento 1, la cantidad de eventos por cada magnitud es muy grande, razón por la cual, en los rangos grandes de magnitudes, la complejidad de los recorridos se asemeja a O(N). Sin embargo, se esperaría tener tiempos de carga inferiores a los registrados con un rango de búsqueda menos amplio.

Requerimiento 3

```python
def req_3(data_structs, min_mag, max_depth):
    """
    Función que soluciona el requerimiento 3
    """
    # Realizar el requerimiento 3
    mags_om = data_structs['req3']
    exists_min = om.contains(mags_om, min_mag)
    if not exists_min:
        min_mag = om.ceiling(mags_om, min_mag)
    mag_range = om.values(mags_om, min_mag, om.maxKey(mags_om))

    events_list, count = req_3_list(mag_range, max_depth)

    return events_list, count
```

F3.1: req_3()

```python
def req_3_list(mag_range, max_depth):
    event_list = lt.newList('ARRAY_LIST')

    details_columns = ['mag','lat','long','depth','sig','gap','nst','title',
                       'cdi','mmi','magType','type','code']
    count = {'events_in_range':0}
    permanent_max_depth = max_depth

    for mag in lt.iterator(mag_range):
        exists_depth = om.contains(mag, permanent_max_depth)
        if not exists_depth:
            max_depth = om.floor(mag, permanent_max_depth)
        else:
            max_depth = permanent_max_depth
        depth_range = om.values(mag, om.minKey(mag), max_depth)
        details = lt.newList()

        for depth in lt.iterator(depth_range):
            events_map = mp.newMap(maptype='PROBING')

            for event in lt.iterator(depth):
                count['events_in_range'] += 1
                event_time = get_date_string(event['time'])

                entry = mp.get(events_map, event_time)
                if entry is None:
                    event_info = {'count':0,
                                  'details':lt.newList('ARRAY_LIST')}
                    mp.put(events_map, event_time, event_info)
                else:
                    event_info = me.getValue(entry)

                event_info['count'] += 1
                event_elem = []
                req3_details_element(event_elem, event)
                details = event_info['details']
                lt.addLast(details, event_elem)

            for time in lt.iterator(mp.keySet(events_map)):
                entry = mp.get(events_map, time)
                event = me.getValue(entry)

                event_count = event['count']
                details = event['details']
                sort(details, 'req1y3_details')
                if lt.size(details) > 6:
                    details = new_topbot_sublist(details, 3)
                elem = [time, event_count,
                        tabulate(lt.iterator(details),
                                 tablefmt="grid",
                                 headers=details_columns,
                                 maxcolwidths = [18,5,5,5,5,5,5,20,11,11,5,20,20])]
                lt.addFirst(event_list, elem)

    sort(event_list, 'req3_events')
    event_list = lt.subList(event_list, 1, 10)

    return event_list, count
```

F3.2: req_3_list()

```
def req3_details_element(event_elem, event):
    event_elem.append(round(float(event['mag']),3))
    event_elem.append(round(float(event['lat']),3))
    event_elem.append(round(float(event['long']),3))
    event_elem.append(round(float(event['depth']),3))
    event_elem.append(event['sig'])
    gap = event['gap']
    if gap != '':
        event_elem.append(round(float(gap),3))
    else:
        event_elem.append(0.000)
    nst = event['nst']
    if nst != '':
        event_elem.append(float(nst))
    else:
        event_elem.append(1)
    event_elem.append(event['title'])
    cdi = event['cdi']
    if cdi != '':
        event_elem.append(round(float(cdi),3))
    else:
        event_elem.append('Unavailable')
    mmi = event['mmi']
    if mmi != '':
        event_elem.append(round(float(mmi),3))
    else:
        event_elem.append('Unavailable')
    event_elem.append(event['magType'])
    event_elem.append(event['type'])
    event_elem.append(event['code'])
```

F3.3: req3_details_element()

Descripción

El requerimiento 3 pretende encontrar todos los eventos que ocurrieron en un rango de magnitudes y profundidades, y retornar los 10 eventos más recientes cuyas características coinciden con los parámetros de entrada. Para esto se utiliza un árbol RBT de magnitudes, donde cada magnitud tiene un árbol RBT de profundidades. Finalmente, cada profundidad tiene una lista con los eventos ocurridos en dicha magnitud a dicha profundidad. Esto se hace con el fin de reducir la complejidad de los recorridos.

Entrada	Estructuras de datos del modelo, magnitud mínima de consulta, profundidad máxima de consulta
Salidas	Lista con los 10 eventos más recientes ordenados cronológicamente con una magnitud mínima y profundidad máxima
Implementado (Sí/No)	Sí, Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
3.1	Obtener la lista de valores de magnitudes en el rango apropiado	O(1)
3.1	Invocar 3.2	O(1)
3.2	Recorrer lista de magnitudes	O(M)

3.2	Obtener lista de profundidades en el rango apropiado	O(1)
3.2	Recorrer lista de profundidades	O(D)
3.2	Recorrer lista de eventos	O(E)
3.2	Recorrer lista de times del mapa y crear el elemento de la lista respuesta	O(t)
3.2	sort	O(log(e))
Total		O(MDEt + log(e))

Pruebas Realizadas

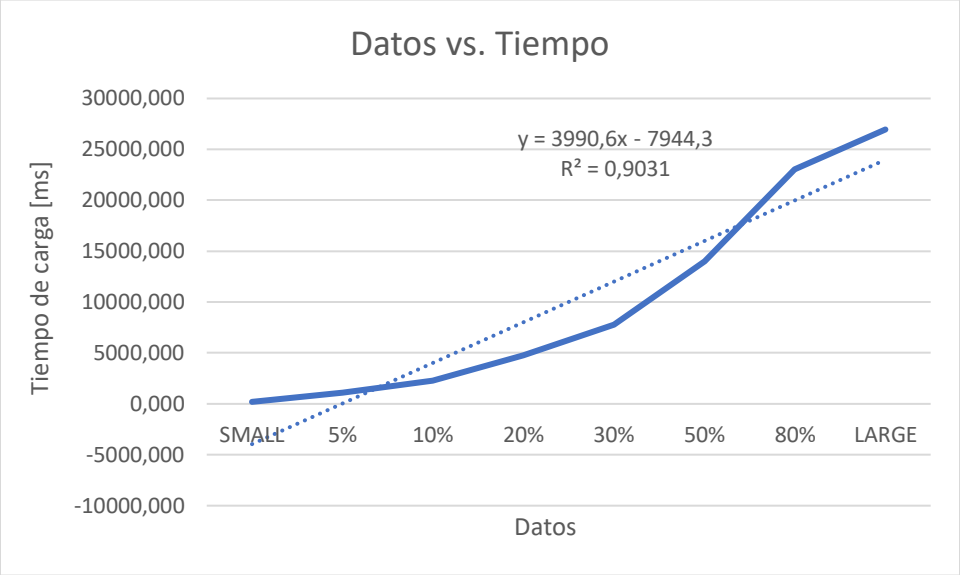
Las pruebas realizadas fueron realizadas con las siguientes especificaciones. Los datos de entrada fueron una magnitud de **4.7** y una profundidad de **10.0**

Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	199.900
5%	1090.686
10%	2271.045
20%	4751.706
30%	7804.597
50%	14016.073
80%	23020.54
LARGE	26953.615

Graficas



Análisis

A pesar de que se utilizan estructuras de datos diseñadas para acelerar los recorridos, debido a la gran cantidad de datos que se deben evaluar, la complejidad del requerimiento es alta. Sin embargo, su comportamiento no es cuadrático, como se puede evidenciar en los tiempos registrados, razón por la cual es beneficioso utilizar un algoritmo como el implementado para este requerimiento.

Requerimiento 4

```
def req4_data(data_structs):
    #for y in lt.iterator(data_structs["temblores"]):
    lst_req4 = lt.newList('SINGLE_LINKED')
    for data in lt.iterator(data_structs["temblores"]):
        lt.addLast (lst_req4,{"time":data["time"],
                    "events": 1,
                    "details":{"mag":data["mag"],
                               "lat":data["lat"],
                               "long":data["long"],
                               "depth":data["depth"],
                               "sig":data["sig"],
                               "gap":data["gap"],
                               "nst":data["nst"],
                               "title":data["title"],
                               "cdi":data["cdi"],
                               "mmi":data["mmi"],
                               "magType":data["magType"],
                               "type":data["type"],
                               "code":data["code"]
                               }
                    })
        #lt.addLast (lst_req4,capsule_lst)
    #print(capsule_lst)
    return lst_req4
```

```python
# ****************************** TRANSFORMAR TEXTO A FORMATO DE F
def time_operabile(date):
    time =date['time']
    format = '%Y-%m-%dT%H:%M'
    trans_date = dt.strptime(time,format)
    return trans_date
# ****************************** COMPARAR DOS TIEMPOS ***********
def sort_comparation_time (data_1, data_2):

    if time_operabile(data_1) != time_operabile(data_2):
        return time_operabile(data_1)>time_operabile(data_2)
    #else:
    #   return data_1['country']<data_2['country']
# ****************************** RETORNA LOS PRIMEROS Y ULTIMOS 3
def first_last(informacion):

    if lt.size(informacion) < 6:
        return informacion
    primeros = lt.subList(informacion,1,3)
    ultimos = lt.subList(informacion,lt.size(informacion)-2,3)
    respuesta = lt.newList('ARRAY_LIST')
    for i in lt.iterator(primeros):
        lt.addLast(respuesta,i)
    for i1 in lt.iterator(ultimos):
        lt.addLast(respuesta,i1)
    return respuesta
```

```python
def req_4(data_structs,min_sig,max_gap):
    """
    Función que soluciona el requerimiento 4
    """
    # Realizar el requerimiento 4


    data_lst = lt.newList('SINGLE_LINKED')
    data_data = req4_data(data_structs)
    for mg in lt.iterator(data_data):
        #print(mg["details"]["sig"])
        if min_sig < mg["details"]["sig"] and mg["details"]["gap"] < max_gap:
            mg["details"] = tabulate([mg["details"]],headers="keys",tablefmt="grid")
            lt.addLast(data_lst,mg)

    quk.sort(data_lst,sort_comparation_time)
    count = 0
    lst_fnd = lt.newList('SINGLE_LINKED')
```

Descripción

En el requerimiento 4 tenemos que filtrar por valores que sean mayor a la significancia y menor a la distancia abismal, y de acuerdo a esto dar los últimos 15, y retornar los primeros 3 y ultimos 3 de esos datos, todo esto implementado las estructuras de datos vistas en clase

Entrada	significancia mínima, distancia azimutal máxima
Salidas	Tabla con los últimos 15 filtrado por esos parámetros
Implementado (Sí/No)	Si, Jhonny Hortua

Análisis de complejidad

Creación de la parte contenedora de la información	O(n)
Filtración por parámetros	O(log(n))
obtención de información de la carga de datos.	O(1)
Obtención y reasignación de información	O(log(n))
Pasos	**Complejidad**
Retornar primeros y ultimos 3	O(n)
TOTAL	***O(nlog(n))***

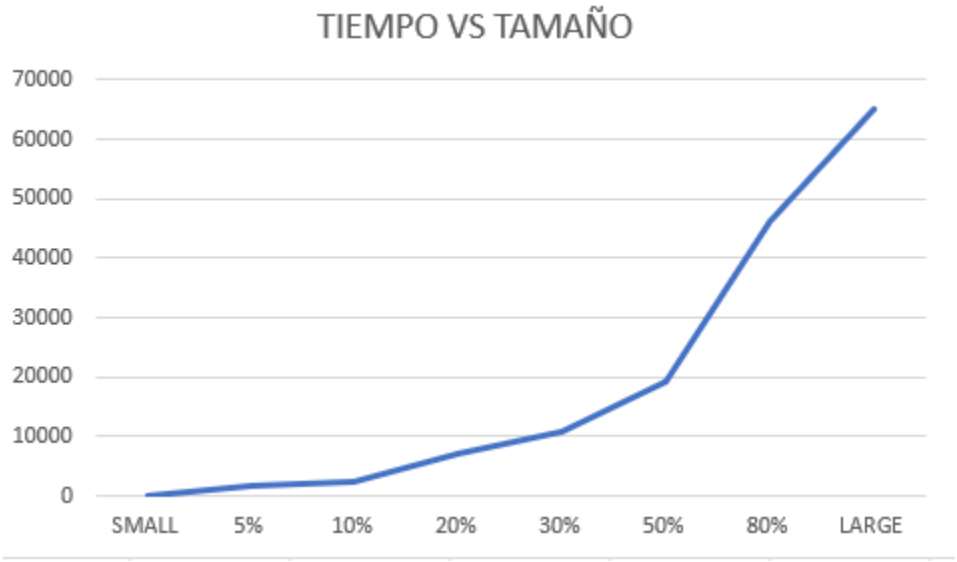
Pruebas Realizadas

Procesadores	**AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	175.651
5%	1543.283
10%	2354.397
20%	7253.263
30%	10695.648
50%	19253.395
80%	46365.187
LARGE	65064.958

Graficas



Análisis

Requerimiento 5

```
def req5_data(data_structs, data):
    stations_om = data_structs['req5']# Obtención mapa de estaciones
    if data['nst'] == '':
        data['nst'] = '1'
    data_station = round(float(data['nst']), 3)
    entry = om.get(stations_om, data_station)#defición del entry
    if entry is not None:
        mags_om = me.getValue(entry)
    else:
        mags_om = om.newMap(cmpfunction=req5_compare_mag)#creación mapa de profundidades
        om.put(stations_om, data_station, mags_om)#se agrega el mapa al mapa de estaciones

    data_mag = round(float(data['depth']), 3)
    entry = om.get(mags_om, data_mag)
    if entry is not None:
        date_list = me.getValue(entry)
    else:
        date_list = lt.newList('ARRAY_LIST')#Creación de la lista que contiene toda la información de los eventos
        om.put(mags_om, data_mag, date_list)
    lt.addLast(date_list, data)#Se agregan los datos del evento a la lista
```

Carga de datos requerimiento 5

```python
def req_5(data_structs, depth, stations):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5

    stations_om = data_structs['req5']#Obtención mapa de estaciones
    entry = om.get(stations_om, float(stations))
    if entry is not None:#busqueda de la estación mínima en el mapa
        min_stations = stations
    else:
        min_stations = om.ceiling(stations_om, stations)#en caso de que no se encuentre la cantidad mínima se busca el mínimo más cercano que
                                                        #se encuentre en el mapa
    stations_range = om.values(stations_om, min_stations, om.maxKey(stations_om)) #Obtención del rango de estaciones permitidas
    event_list = lt.newList('ARRAY_LIST')
    for station in lt.iterator(stations_range):#Recorrido estaciones en rango permitido
        min_depth = depth
        exist_depth = om.contains(station, min_depth)#Búsqueda de la profundidad mínima en el mapa de profundidades
        if not exist_depth:
            min_depth = om.ceiling(station, depth)#En caso de que no se encuentra la profundidad mínima se busca el mínimo más cercano que se
                                                  #se encuentre en el mapa
            if min_depth == None:
                continue
        depth_range = om.values(station, min_depth, om.maxKey(station))#Obtención del rango de profundidades permitidas
        for depth_1 in lt.iterator(depth_range):#Recorrido del rango de profundidades permitidas
            for event in lt.iterator(depth_1):
                lt.addLast(event_list, event)#Se agrega la información del evento a la lista de eventos

    event_list_sorted = sa.sort(event_list, req5_event_criteria)#Ordenamiento de la lista por fecha son método shellsort
    contador = lt.size(event_list_sorted)#Contador para cantidad de eventos encontrados en los rangos permitidos

    if contador > 20:
        top_twenty_list = lt.subList(event_list_sorted, contador-19, 20)#Obtención de los 20 eventos más recientes
    else:
        top_twenty_list = event_list_sorted
```

Implementación requerimiento 5

Descripción

El requerimiento 5 tiene como objetivo buscar los 20 eventos más recientes que sean mayores a una profundidad y cantidad de estaciones mínimas dadas por el usuario. Para realizar esta función se implementaron árboles de tipo RBT para reducir los tiempos de búsqueda. En primer lugar, se usó un árbol RBT donde las llaves son la cantidad de estaciones de los eventos. Luego se implementó otro árbol RBT donde las llaves son la profundidad de los eventos. Por último, se almacenó la información total de los eventos en listas tipo ARRAY_LIST para facilidad de ordenamientos.

Entrada	Profundidad mínima, cantidad mínima de estaciones
Salidas	20 eventos más recientes con los filtros requeridos
Implementado (Sí/No)	Si – Gabriel Esteban González Carrillo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Declaración de variables y obtención de información de la carga de datos.	O(1)
Recorrido árbol de cantidad de estaciones	O(log(n))
Obtención rango de estaciones permitido	O(log(n))
Recorrido árbol de profundidades	O(log(n))
Obtención rango de profundidades permitido	O(log(n))
Recorrido listas ARRAY_LIST	O(n)

Lt.addlast()	
TOTAL	***O(nlog(n))***

Pruebas Realizadas

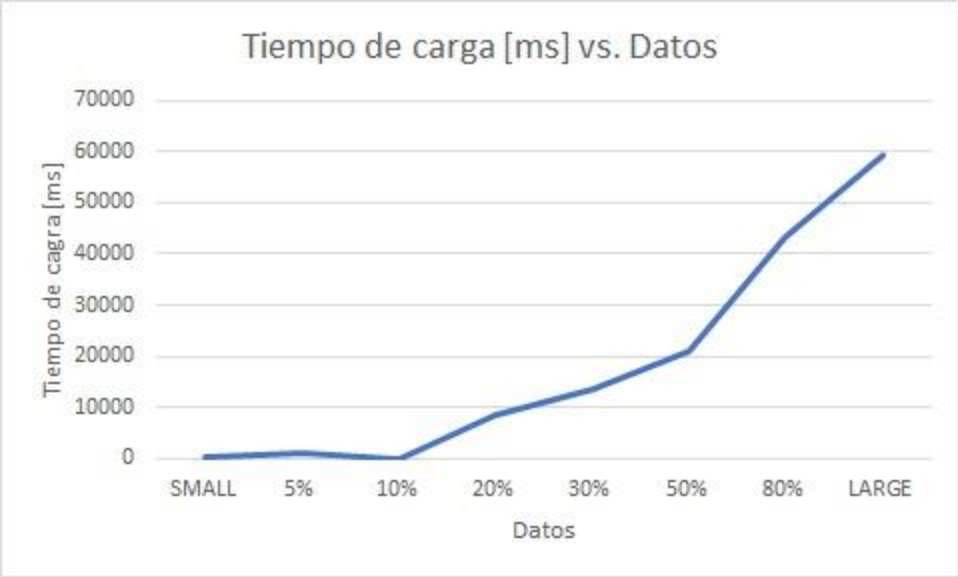
Las pruebas realizadas se implementaron con profundidad mínima de 23 y cantidad mínima de estaciones de 38. Al ingresar estos datos de entrada al código realizado para el requerimiento 5 se obtuvieron los siguientes tiempos al varias el porcentaje de cantidad de datos a revisar.

Procesadores	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.20 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	166.460
5%	1109.269
10%	2919.584
20%	8517.220
30%	13448.490
50%	21207.358
80%	43269.012
LARGE	59240.427

Gráficas



Gráfica cantidad de datos vs tiempo de carga.

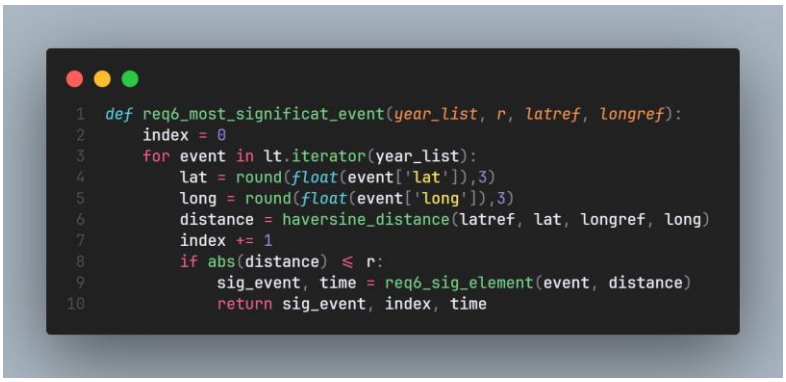
Análisis

En la gráfica representativa de los tiempos de carga se puede evidenciar como al tener una cantidad de datos baja los árboles reducen considerablemente dicho tiempo. Sin embargo, cuando se aumentan desde el 20% de la carga total empiezan a crecer con una pendiente bastante pronunciada. Esto refleja como a pesar de los árboles reducir en gran proporción los tiempos de carga, cuando la cantidad de datos aumenta no importa la estructura de datos que se use lo tiempos van a ser muy grandes.

Requerimiento 6

```python
def req_6(data_structs, year, r, n, latref, longref):
    """
    Función que soluciona el requerimiento 6
    """
    # Realizar el requerimiento 6
    dates_om = data_structs['req1y6']
    entry_year_list = om.get(dates_om, year)
    year_list = me.getValue(entry_year_list)
    r = int(r)
    n = int(n)
    latref = round(float(latref), 3)
    longref = round(float(longref), 3)

    sig_event, index, sig_time  = req6_most_significat_event(year_list, r, latref, longref)
    year_list = lt.subList(year_list, index+1, (lt.size(year_list) - index))

    event_list, count = req6_list(year_list, sig_time, r, n, latref, longref)

    return event_list, count, sig_event
```

F6.1: req_6()



```
def req6_most_significat_event(year_list, r, latref, longref):
    index = 0
    for event in lt.iterator(year_list):
        lat = round(float(event['lat']),3)
        long = round(float(event['long']),3)
        distance = haversine_distance(latref, lat, longref, long)
        index += 1
        if abs(distance) <= r:
            sig_event, time = req6_sig_element(event, distance)
            return sig_event, index, time
```

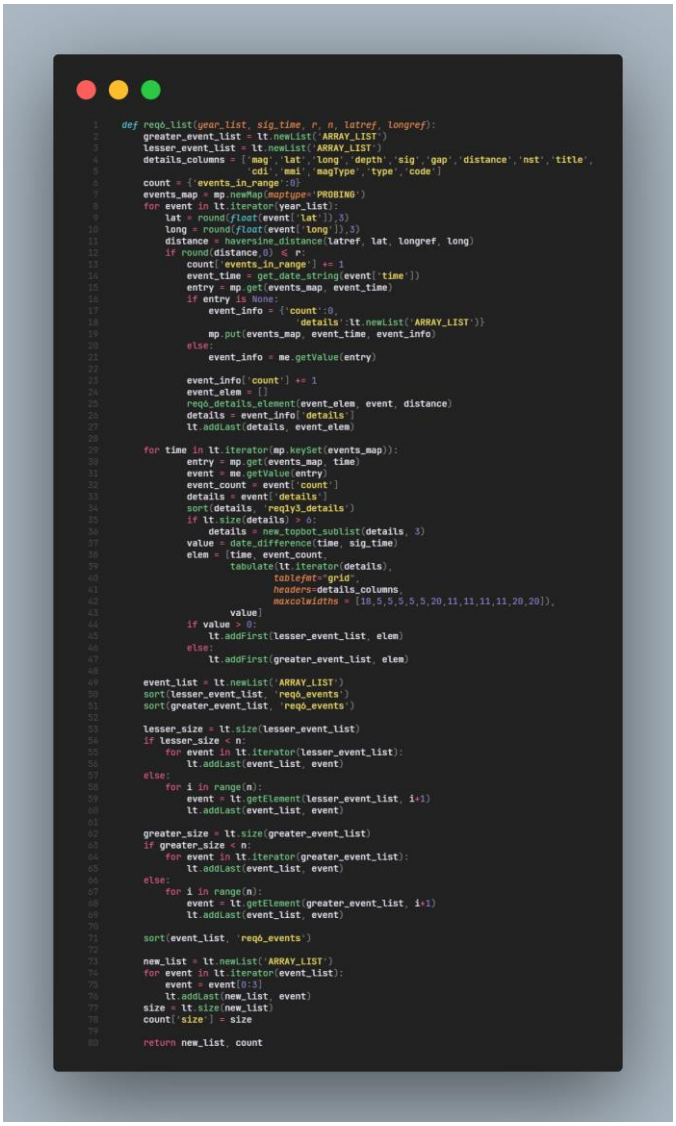
F6.2: req6_most_significant_event()

```
def haversine_distance(lat1, lat2, long1, long2):
    lat1 = m.radians(lat1)
    lat2 = m.radians(lat2)
    long1 = m.radians(long1)
    long2 = m.radians(long2)

    halflatdiff = (lat2 - lat1)/2
    first_term = m.sin(halflatdiff)**2

    coslat1 = m.cos(lat1)
    coslat2 = m.cos(lat2)
    halflongdiff = (long2 - long1)/2
    second_term = coslat1 * coslat2 * (m.sin(halflongdiff)**2)

    in_root_expression = first_term + second_term

    final_term = m.asin(m.sqrt(in_root_expression))

    distance = 2*final_term*6341

    return abs(distance)
```

F6.3: haversine_distance()

```python
def req6_sig_element(event, distance):
    event_elem = []
    sig_event = lt.newList('ARRAY_LIST')
    time = get_date_string(event['time'])
    event_elem.append(time)
    event_elem.append(round(float(event['mag']),3))
    event_elem.append(round(float(event['lat']),3))
    event_elem.append(round(float(event['long']),3))
    event_elem.append(round(float(event['depth']),3))
    event_elem.append(event['sig'])
    gap = event['gap']
    if gap != '':
        event_elem.append(round(float(gap),3))
    else:
        event_elem.append(0.000)
    event_elem.append(distance)
    nst = event['nst']
    if nst != '':
        event_elem.append(float(nst))
    else:
        event_elem.append(1)
    event_elem.append(event['title'])
    cdi = event['cdi']
    if cdi != '':
        event_elem.append(round(float(cdi),3))
    else:
        event_elem.append('Unavailable')
    mmi = event['mmi']
    if mmi != '':
        event_elem.append(round(float(mmi),3))
    else:
        event_elem.append('Unavailable')
    event_elem.append(event['magType'])
    event_elem.append(event['type'])
    event_elem.append(event['code'])

    lt.addFirst(sig_event, event_elem)

    return sig_event, time
```

F6.4: req6_sig_element()



```
def req6_list(year_list, sig_time, r, n, latref, longref):
    greater_event_list = lt.newList('ARRAY_LIST')
    lesser_event_list = lt.newList('ARRAY_LIST')
    details_columns = ['mag','lat','long','depth','sig','gap','distance','nst','title',
                       'cdi','mmi','magType','type','code']
    count = {'events_in_range':0}
    events_map = mp.newMap(maptype='PROBING')
    for event in lt.iterator(year_list):
        lat = round(float(event['lat']),3)
        long = round(float(event['long']),3)
        distance = haversine_distance(latref, lat, longref, long)
        if round(distance,0) <= r:
            count['events_in_range'] += 1
            event_time = get_date_string(event['time'])
            entry = mp.get(events_map, event_time)
            if entry is None:
                event_info = {'count':0,
                              'details':lt.newList('ARRAY_LIST')}
                mp.put(events_map, event_time, event_info)
            else:
                event_info = me.getValue(entry)

            event_info['count'] += 1
            event_elem = []
            req6_details_element(event_elem, event, distance)
            details = event_info['details']
            lt.addLast(details, event_elem)

    for time in lt.iterator(mp.keySet(events_map)):
            entry = mp.get(events_map, time)
            event = me.getValue(entry)
            event_count = event['count']
            details = event['details']
            sort(details, 'req1y3_details')
            if lt.size(details) > 6:
                details = new_topbot_sublist(details, 3)
            value = date_difference(time, sig_time)
            elem = [time, event_count,
                    tabulate(lt.iterator(details),
                             tablefmt="grid",
                             headers=details_columns,
                             maxcolwidths = [18,5,5,5,5,5,20,11,11,11,11,20,20]),
                    value]
            if value > 0:
                lt.addFirst(lesser_event_list, elem)
            else:
                lt.addFirst(greater_event_list, elem)

    event_list = lt.newList('ARRAY_LIST')
    sort(lesser_event_list, 'req6_events')
    sort(greater_event_list, 'req6_events')

    lesser_size = lt.size(lesser_event_list)
    if lesser_size < n:
        for event in lt.iterator(lesser_event_list):
            lt.addLast(event_list, event)
    else:
        for i in range(n):
            event = lt.getElement(lesser_event_list, i+1)
            lt.addLast(event_list, event)

    greater_size = lt.size(greater_event_list)
    if greater_size < n:
        for event in lt.iterator(greater_event_list):
            lt.addLast(event_list, event)
    else:
        for i in range(n):
            event = lt.getElement(greater_event_list, i+1)
            lt.addLast(event_list, event)

    sort(event_list, 'req6_events')

    new_list = lt.newList('ARRAY_LIST')
    for event in lt.iterator(event_list):
        event = event[0:3]
        lt.addLast(new_list, event)
    size = lt.size(new_list)
    count['size'] = size

    return new_list, count
```

F6.5: req6_list()

```
def req6_details_element(event_elem, event, distance):
    event_elem.append(round(float(event['mag']),3))
    event_elem.append(round(float(event['lat']),3))
    event_elem.append(round(float(event['long']),3))
    event_elem.append(round(float(event['depth']),3))
    event_elem.append(event['sig'])
    gap = event['gap']
    if gap != '':
        event_elem.append(round(float(gap),3))
    else:
        event_elem.append(0.000)
    event_elem.append(distance)
    nst = event['nst']
    if nst != '':
        event_elem.append(float(nst))
    else:
        event_elem.append(1)
    event_elem.append(event['title'])
    cdi = event['cdi']
    if cdi != '':
        event_elem.append(round(float(cdi),3))
    else:
        event_elem.append('Unavailable')
    mmi = event['mmi']
    if mmi != '':
        event_elem.append(round(float(mmi),3))
    else:
        event_elem.append('Unavailable')
    event_elem.append(event['magType'])
    event_elem.append(event['type'])
    event_elem.append(event['code'])
```

F6.6: req6_details_element

Descripción

Este requerimiento pretende evaluar el evento más significativo en un área específica, al igual que los N elementos más cercanos cronológicamente en el área especificada. Para esto se utiliza un árbol RBT de fechas, el cual es idéntico al del requerimiento 1, donde cada fecha es un ADT List ordenado por significancia del evento. De esta manera, se reducen los tiempos de búsqueda de la información pertinente a una complejidad significativamente menor a O(N^2).

Entrada	Estructuras de datos del modelo, año, radio de búsqueda, número de eventos, latitud y longitud de referencia.
Salidas	El evento más significativo del año en el área y todos los eventos más significativos y cercanos cronológicamente al evento más significativo.
Implementado (Sí/No)	Sí. Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
6.1	Obtener la lista del año	O(1)
6.1	Invocar 6.2	O(1)
6.2	Hacer un recorrido parcial de la lista del año e invocar 6.3 y 6.4	O(Y - k)
6.1	Crear una sublista del año e invocar 6.5	O(1)

6.5	Recorrer la sublista del año e invocar 6.6	O(k)
6.5	Recorrer la lista respuesta para arreglar detalles y reducir su tamaño de acuerdo a los parámetros de entrada	O(r)
6.5	sort	O(log(r))
Total		O(Y + r + log(r))

Pruebas Realizadas

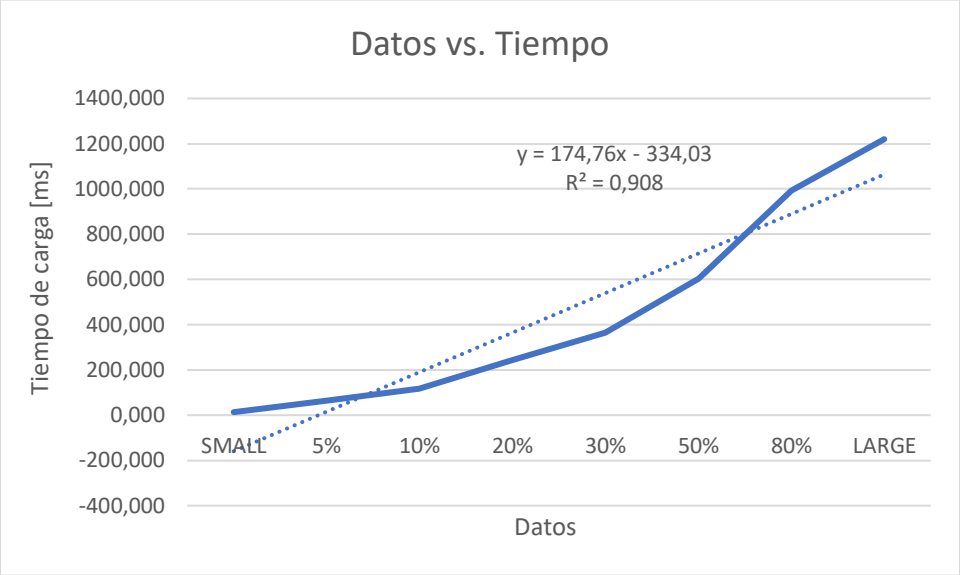
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el año **2022**, en un radio de **3000**, **5** eventos, con una latitud, longitud de **4.674, -74,068.**

Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	13.492
5%	63.003
10%	117.095
20%	242.641
30%	365.576
50%	604.442
80%	992.572
LARGE	1220.157

Graficas



Análisis

Gracias a las estructuras de datos implementadas, los tiempos de ejecución de este requerimiento son relativamente bajos, y tienen un comportamiento muy cercano al lineal. Sin embargo, debido a la gran cantidad de datos, es evidente que en los tamaños grandes los tiempos aumentan.

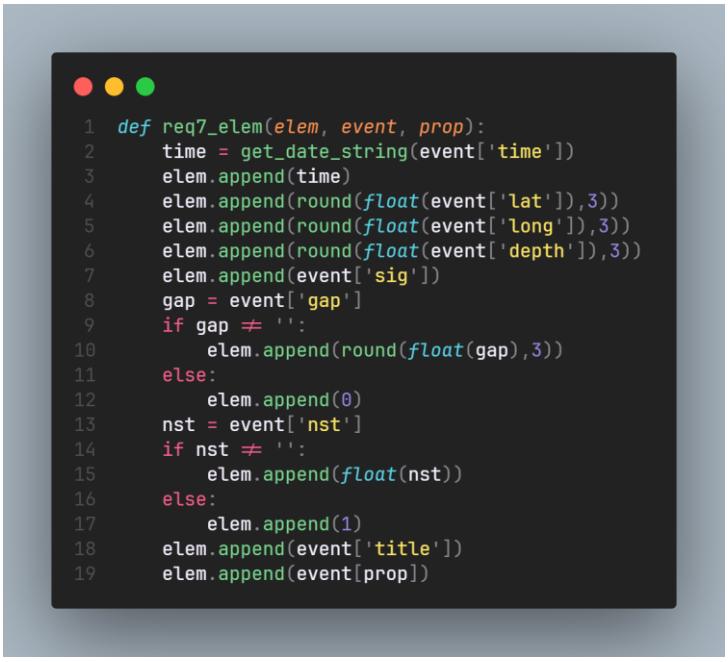
Requerimiento 7

```python
def req_7(data_structs, year, title, prop, bins):
    """
    Función que soluciona el requerimiento 7
    """
    #Realizar el requerimiento 7
    dates_mp = data_structs['req7']
    entry = mp.get(dates_mp, year)
    places_mp = me.getValue(entry)
    entry = mp.get(places_mp, title)
    place_list = me.getValue(entry)

    count_entry = mp.get(places_mp, 'count')
    count = me.getValue(count_entry)
    year_count = lt.size(count)

    event_list, count, histogram_list = req7_list(place_list, prop, year_count)

    return event_list, count, histogram_list
```

F7.1: req_7()

```
def req7_list(place_list, prop, year_count):
    event_list = lt.newList()
    histogram_list = []

    count = {'total_events':year_count,
             'events_in_range':0}

    for event in lt.iterator(place_list):

        if event[prop] != '':
            event_prop = round(float(event[prop]),3)
        else:
            event_prop = 'Unkown'

        if event_prop != 'Unkown':
            count['events_in_range'] += 1
            elem = []
            req7_elem(elem, event, prop)
            lt.addLast(event_list, elem)
            histogram_list.append(event_prop)

    sort(event_list, 'req7_events')
    count['max'] = lt.firstElement(event_list)[-1]
    count['min'] = lt.lastElement(event_list)[-1]

    return event_list, count, histogram_list
```

F7.2: req7_list()



```
def req7_elem(elem, event, prop):
    time = get_date_string(event['time'])
    elem.append(time)
    elem.append(round(float(event['lat']),3))
    elem.append(round(float(event['long']),3))
    elem.append(round(float(event['depth']),3))
    elem.append(event['sig'])
    gap = event['gap']
    if gap != '':
        elem.append(round(float(gap),3))
    else:
        elem.append(0)
    nst = event['nst']
    if nst != '':
        elem.append(float(nst))
    else:
        elem.append(1)
    elem.append(event['title'])
    elem.append(event[prop])
```

F7.3: req7_elem()

Descripción

Este requerimiento pretende encontrar todos los eventos en un año y un lugar, y crear un histograma con su información. Para esto, se hace un mapa de años, donde cada año tiene un mapa de lugares. Cada lugar tiene una lista con la información pertinente.

Entrada	Estructuras de datos del modelo, año de consulta, lugar de consulta, propiedad a evaluar, y número de bins.
Salidas	Una lista con la información de todos los eventos con propiedad conocida en el año y el lugar especificados junto a un histograma con la información.
Implementado (Sí/No)	Sí. Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
7.1	Obtener la lista del lugar e invocar 7.2	O(1)
7.2	Recorrer la lista del lugar e invocar 7.3	O(P)
7.2	sort	O(log(P))
7.2	sort	O(P + log(P))

Pruebas Realizadas

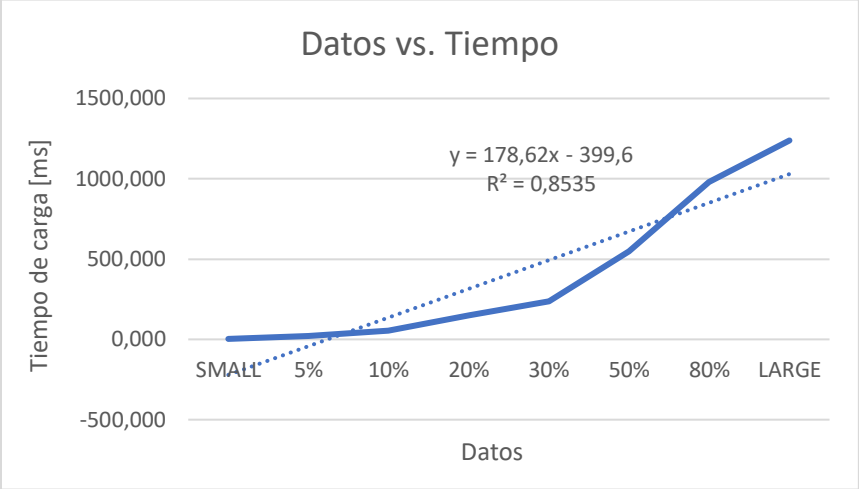
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el año **2020**, en **Alaska**, **Magnitud** y con **10** bins.

Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Tablas de datos

Datos vs. Tiempo	
Datos	**Tiempo de carga [ms]**
SMALL	2.965
5%	22.484
10%	54.471
20%	150.463
30%	236.095
50%	548.498
80%	980.046
LARGE	1238.459

Graficas



Análisis

Al realizar las pruebas para este requerimiento, es evidente que su complejidad no es tan cercana a la lineal como se esperaba. Sin embargo, esta sigue siendo menor a la cuadrática. Cabe resaltar que los tiempos de carga con las cantidades más pequeñas de datos, los tiempos de carga son muy bajos, pero con las cantidades grandes se demora significativamente más.

Requerimiento 8

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

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

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(...)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(...)***

Pruebas Realizadas

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

Entrada	**Tiempo (s)**

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

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