ANÁLISIS DEL RETO

Sebastián Palma, <s.palma@uniandes.edu.co>, 202222498
Sara Leiva, <s.leivam@uniandes.edu.co>, 202220956.
Andrés Rodríguez, <a.rodiguezs@uniandes.edu.co>,202222586.

Carga de datos

Descripción

```
# Construccion de modelos

def new_data_structs():
    """
    Inicializa las estructuras de datos del modelo. Las crea de
    manera vacía para posteriormente almacenar la información.
    """

    data_structs = {'accidents': None,
                'dateIndex': None}
    data_structs['accidents'] = lt.newList('SINGLE_LINKED', compareIds)
    data_structs['dateIndex'] = om.newMap(omaptype='RBT',
                                          comparefunction=compareDates)
    return data_structs

# Funciones para agregar informacion al modelo

def add_data(data_structs, data):
    """
    Función para agregar nuevos elementos a la lista
    """

    lt.addLast(data_structs['accidents'], data)
    updateDateIndex(data_structs['dateIndex'], data)
    return data_structs

def updateDateIndex(map, data):

    occurreddate = data['FECHA_OCURRENCIA_ACC']
    accdate = datetime.datetime.strptime(occurreddate, '%Y/%m/%d')
    entry = om.get(map, accdate.date())
    if entry is None:
        datentry = newDataEntry(data)
```

```
        om.put(map, accdate.date(), datentry)
    else:
        datentry = me.getValue(entry)
    addDateIndex(datentry, data)
    addDateIndex2(datentry,data)
    return map

def newDataEntry(data):
    """
    Crea una entrada en el indice por fechas, es decir en el arbol
    binario.
    """
    entry = {'accidentIndex': None, 'lstaccidents': None}
    entry['accidentIndex'] = mp.newMap(numelements=30,
                                       maptype='PROBING',
                                       cmpfunction=compareAcc)
    entry['lstaccidents'] = lt.newList('SINGLE_LINKED', compareDates)
    return entry

def addDateIndex(datentry, data):

    lst = datentry['lstaccidents']
    lt.addLast(lst, data)
    accIndex = datentry['accidentIndex']
    accentry = mp.get(accIndex, data['CLASE_ACC'])
    accentry2 =  mp.get(accIndex, data['GRAVEDAD'])
    if (accentry is None):
        entry = newAccEntry(data['CLASE_ACC'], data)
        lt.addLast(entry['lstaccidents'], data)
        mp.put(accIndex, data['CLASE_ACC'], entry)
    else:
        entry = me.getValue(accentry)
        lt.addLast(entry['lstaccidents'], data)

    if (accentry2 is None):
        entry2 = newAccEntry2(data['GRAVEDAD'], data)
        lt.addLast(entry2['lstaccidents'], data)
        mp.put(accIndex, data['GRAVEDAD'], entry2)
    else:
        entry2 = me.getValue(accentry2)
        lt.addLast(entry2['lstaccidents'], data)
    return datentry

def newAccEntry(acc_class, data):
```

```
    accentry = {'accident': None, 'lstaccidents': None}
    accentry['accident'] = acc_class
    accentry['lstaccidents'] = lt.newList('SINGLE_LINKED', compareAcc)
    return accentry

def newAccEntry2(accgravedad, data):

    accentry = {'accident': None, 'lstaccidents': None}
    accentry['accident'] = accgravedad
    accentry['lstaccidents'] = lt.newList('SINGLE_LINKED', compareAcc)
    return accentry
```

Descripción del código y breve análisis de complejidad

new_data_structs():

Función encargada de construir la estructura de datos.

Pasos	**Complejidad**
Se crea la estructura de datos con llaves inicializadas en None para almacenar los datos.	O (1)
Para la llave ‘accidents’ se crea una lista encadenada con una función de comparación determinada.	O (1)
Para la llave de ‘dateIndex’ se crea un árbol RBT con una función de comparación determinada.	O (1)
TOTAL	O (1)

updateDateIndex():

Función encargada de actualizar el índice del árbol en el modelo creado. Este índice está compuesto por un mapa, donde su llave es la fecha del accidente y sus valores son los accidentes que ocurrieron en esta fecha.

Pasos	**Complejidad**
Se obtiene la fecha del accidente y se convierte a una fecha manejable a través del uso de la librería datetime.	O (1)
Se busca la fecha en el mapa y se evalúa si existe a través de una condicional.	O (logn)
Si la fecha no se encuentra cargada en un mapa, se llama a una función que va a crear un mapa con esta fecha y va a crear una nueva lista encadenada donde se le va a añadir el accidente.	O(1)
Poner la fecha en el mapa con la información de los accidentes que ocurrieron en esa fecha.	O (logn)
Si la fecha ya se encuentra cargada, obtener los	O(1)
Se llaman a otras funciones para añadir los índices en la estructura de datos.	O(1)
TOTAL	O (logn)

newDataEntry():

Función encargada de crear una entrada en el árbol binario. Aquella se encarga de poner la fecha ingresada en "accidentIndex" como la llave de un nuevo mapa y añadir el accidente a la lista total de los accidentes que ocurrieron en esa fecha.

Pasos	**Complejidad**
Se crea un diccionario inicializando sus valores en None para guardar la información en las llaves determinadas.	O (1)
Para la llave 'accidentIndex' se crea un mapa que tiene como llave la fecha ingresada.	O (1)
Para la llave de "lstaccidents" se crea una lista encadenada donde estarán todos los accidentes que ocurrieron en esa fecha.	O (1)
TOTAL	O (1)

addDateIndex():

Función encargada de actualizar los índices de cada fecha. Estos índices se encuentran ubicados en una tabla de hash donde la llave corresponde a la gravedad y a la clase del accidente.

Pasos	**Complejidad**
Entrar a la lista de accidentes que ocurrieron en esa fecha.	O (1)

Añadir al final el accidente recibido a la lista de accidentes que ocurrieron en esa fecha.	O (1)
Entrar a la tabla de hash que está organizada por la clase y gravedad de los accidentes que ocurrieron en esa fecha.	O (1)
Buscar si la llave está en la tabla de hash.	O (1)
Si la llave no está, entonces llamar a una función que crea una pareja llave-valor, donde la llave es la clase/gravedad del accidente y sus valores corresponden a los accidentes de esa clase/gravedad.	O (1)
Si la llave está, entrar al valor de la llave y añadir el accidente a la lista que tiene como valor.	O (1)
TOTAL	O (1)

newAccEntry() y newAccEntry2:

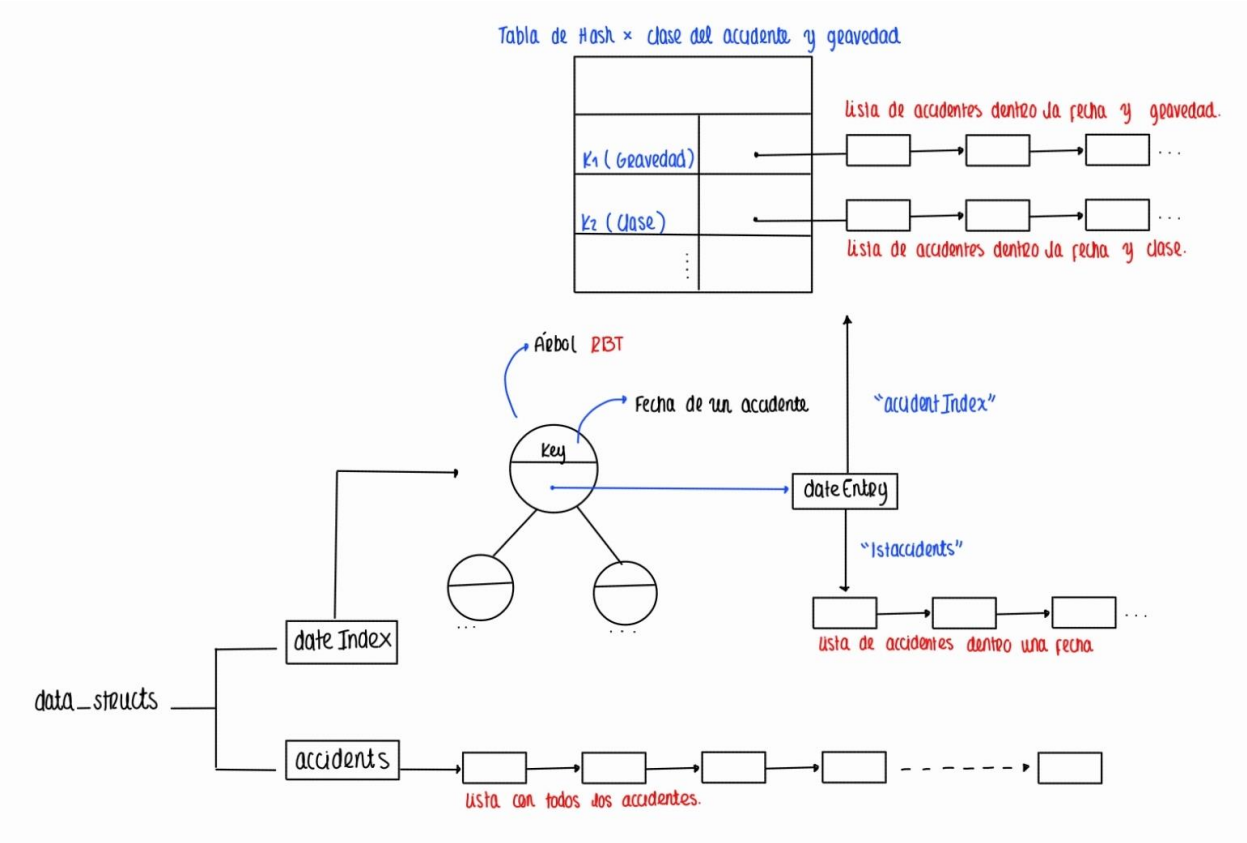
Función encargada de crear la pareja llave-valor por la llave entrada por parámetro.

Pasos	**Complejidad**
Inicializar un diccionario con sus valores en None.	O (1)
Establecer la primera llave del diccionario como la clase/gravedad del accidente.	O (1)
Crear una lista encadenada con una función de comparación determinada donde se guardarán todos los accidentes que tienen determinada clase/gravedad.	O (1)
TOTAL	O (1)

TOTAL de la complejidad de todas las funciones que se usaron para establecer la carga:

- Gracias a la primera tabla tenemos que la complejidad temporal de la carga de datos es de **O(logn).**

Diagrama de la carga de datos:



Requerimiento 1

```
def req_1(data_structs, initialDate, finalDate):
    """
    Función que soluciona el requerimiento 1
    """

    l = om.values(data_structs['dateIndex'], finalDate, initialDate)
    lst = lt.newList("ARRAY_LIST")

    cod = []

    for date in lt.iterator(l):
        for a in lt.iterator(date['lstaccidents']):
            if a["CODIGO_ACCIDENTE"] not in cod:
                lt.addLast(lst, a)
                cod.append(a["CODIGO_ACCIDENTE"])
    tot = lt.size(lst)

    merg.sort(lst, compareTime)
```

```
    return tot, lst
```

Entrada	Data_structs, fecha inicial, fecha final.
Salidas	Lista con los accidentes y cantidad de accidentes que ocurrieron en los rangos de fecha dados.
Implementado (Sí/No)	Si, implementado por Andrés Rodríguez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Retornar todos los valores del data_structs en un rango con om.values() y guardar los valores en la variable “l”. Lo que contiene esa “l” es toda la información de los accidentes en cada fecha que entra en el rango dado.	O (1ogn + k) donde k es la cantidad de nodos en el rango.
Crear un nuevo arreglo llamado “lst”.	O (1)
Crear una lista normal de Python vacía llamada cod.	O (1)
Iterar date en l con lt.iterator(). Aquello itera sobre todas las fechas dentro del rango predeterminado que contienen una lista de accidentes que ocurrieron en estas.	O (n)
Iterar a en date[“lstaccidents”] con lt.iterator(). Acá se entra a la lista de accidentes dentro de cada fecha,	O (n)
Sí el código del accidente no se encuentra en “cod”, añadir el accidente a “lst”.	O (1)
Añadir el código del accidente a “cod”.	O (1)
Obtener el size de lst y guardarlo en la variable “tot”.	O (1)
Organizar lst con merg.sort() por fechas.	O (n*Log(n))
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	Intel® Core™ i5-9300H CPU @2.40GHz
Memoria RAM	8.0 GB
Sistema Operativo	Windows 10 Home Single – 64 bits

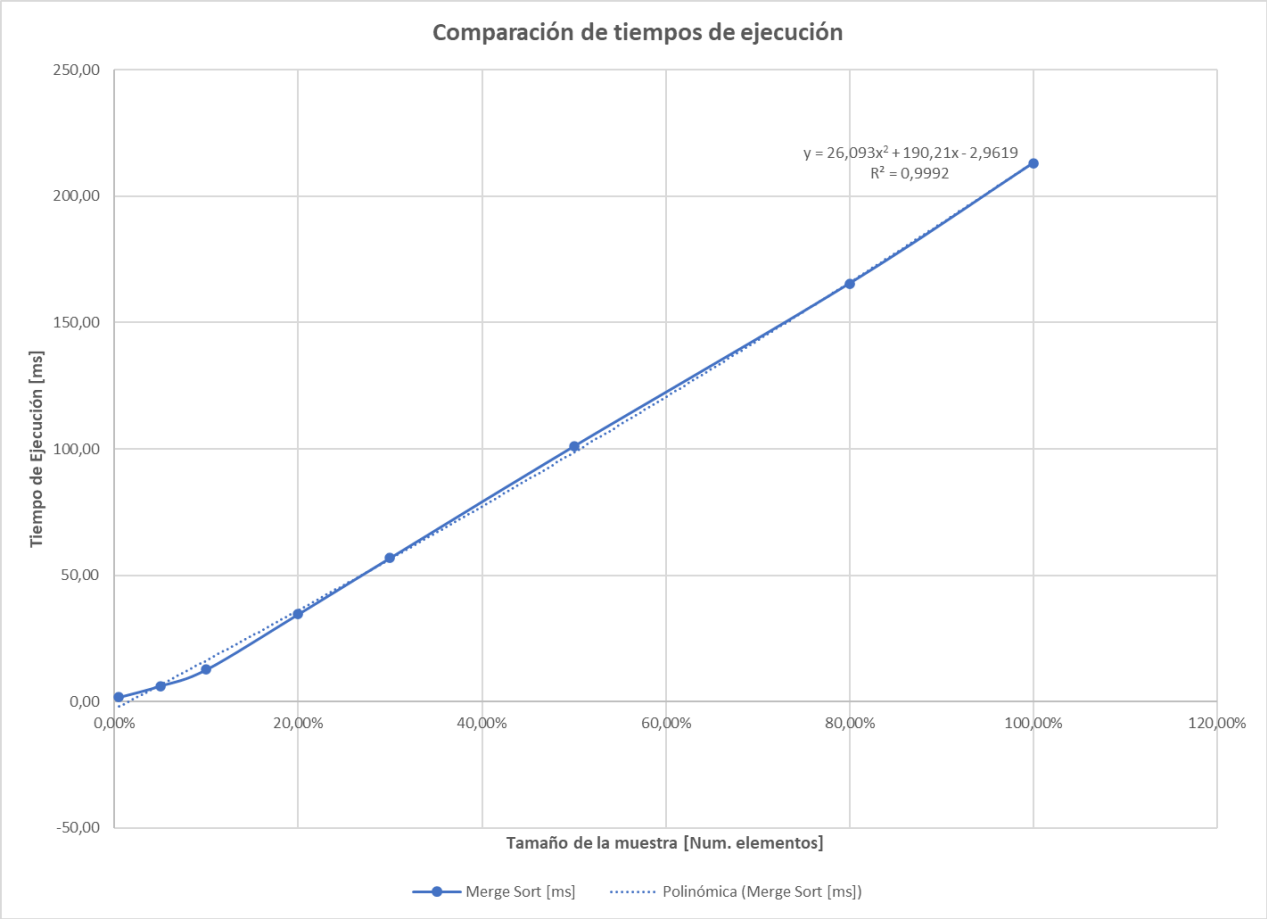
Entrada	**Tiempo (ms)**
small	1.7
5 pct	6.17

10 pct	12.62
20 pct	34.53
30 pct	56.81
50 pct	101.00
80 pct	165.47
large	213.1

Tablas de datos

Muestra	Salida	Tiempo (ms)
small	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	1.7
5 pct	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	6.17
10 pct	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	12.62
20 pct	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	34.53
30 pct	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	56.81
50 pct	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	101.00
80 pct	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	165.47
large	Una lista con todos los accidentes ocurridos dentro de un rango de fechas dado	213.1

Graficas



Análisis

En este requerimiento 1 gracias tanto a los árboles binarios y la forma en la que está construida la carga de datos, es posible obtener grandes cantidades de datos que se encuentran en un rango de fechas de manera mucho más sencilla. También, la implementación de la tabla de hash permitió que se organizara los accidentes por su gravedad, esto facilito mucho más las comparaciones y la búsqueda de los datos. Para este requerimiento se tiene una complejidad O(n^2), esto dado porque en el código se encuentra un ciclo dentro de un ciclo a la hora de buscar la información en la lista que se encuentra dentro del rango de fechas, esta complejidad se encuentra confirmada por la grafica

Requerimiento 2

```
def req_2(data_structs, initialHour, finalHour, dateMin, dateMax):
    """
    Función que soluciona el requerimiento 2
    """
    l = om.values(data_structs['dateIndex'], dateMax, dateMin)
    lst = lt.newList("ARRAY_LIST")
    cod = []
    for value in lt.iterator(l):
```

```
        for ls in lt.iterator(value['lstaccidents']):
            time = datetime.datetime.strptime(ls["HORA_OCURRENCIA_ACC"],
'%H:%M:%S').time()
            if (initialHour <= time <= finalHour) and (ls["CODIGO_ACCIDENTE"] not
in cod):
                lt.addLast(lst, ls)
                cod.append(ls["CODIGO_ACCIDENTE"])
    tot = lt.size(lst)


    merg.sort(lst, compareTime)

    return tot, lst
```

Entrada	Data_structs, hora y minutos iniciales, hora y minutos finales, año y mes de consulta
Salidas	Lista y cantidad de accidentes que ocurrieron en el rango de hora, minutos, año y mes dado.
Implementado (Sí/No)	Si, implementado por Andrés Rodríguez.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Retornar todos los valores del data_structs en un rango con om.values() y guardar los valores en la variable “l”. Lo que contiene esa “l” es toda la información de los accidentes en cada fecha que entra en el rango dado.	O (1ogn + k) donde k es la cantidad de nodos en el rango.
Hacer una nueva lista con lt.newlist y guardarla en “lst”.	O (1)
Crear una lista vacía de Python y guardarla en cod.	O (1)
Iterar sobre las fechas que se encuentran dentro del rango de fechas dadas con lt.iterator().	O (n)
Iterar sobre la lista de accidentes dentro de cada fecha que cumple con los parámetros ingresados con lt.iterator().	O (n)
Guardar la hora de ocurrencia de cada accidente dentro de esta lista en la variable “time”.	O (n)
Si “time” del accidente es menor o igual initialHour, mayor o igual a finalHour y el código del accidente no está en “cod”, agregar el accidente al arreglo creado y agregarle el código del accidente a “cod”.	O (1)
Obtener el tamaño del arreglo que contiene todos los accidentes que cumplen con los parámetros ingresados con lt.size () y guardarlo en la variable “tot”.	O (1)

Organizar lst con merg.sort por fechas	O (N*Log(N))
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	Intel® Core™ i5-9300H CPU @2.40GHz
Memoria RAM	8.0 GB
	Windows 10 Home Single – 64 bits

Entrada	**Tiempo (ms)**
small	0.93
5 pct	1.92
10 pct	3.4
20 pct	8.77
30 pct	12.3
50 pct	21.14
80 pct	34.14
large	47.29

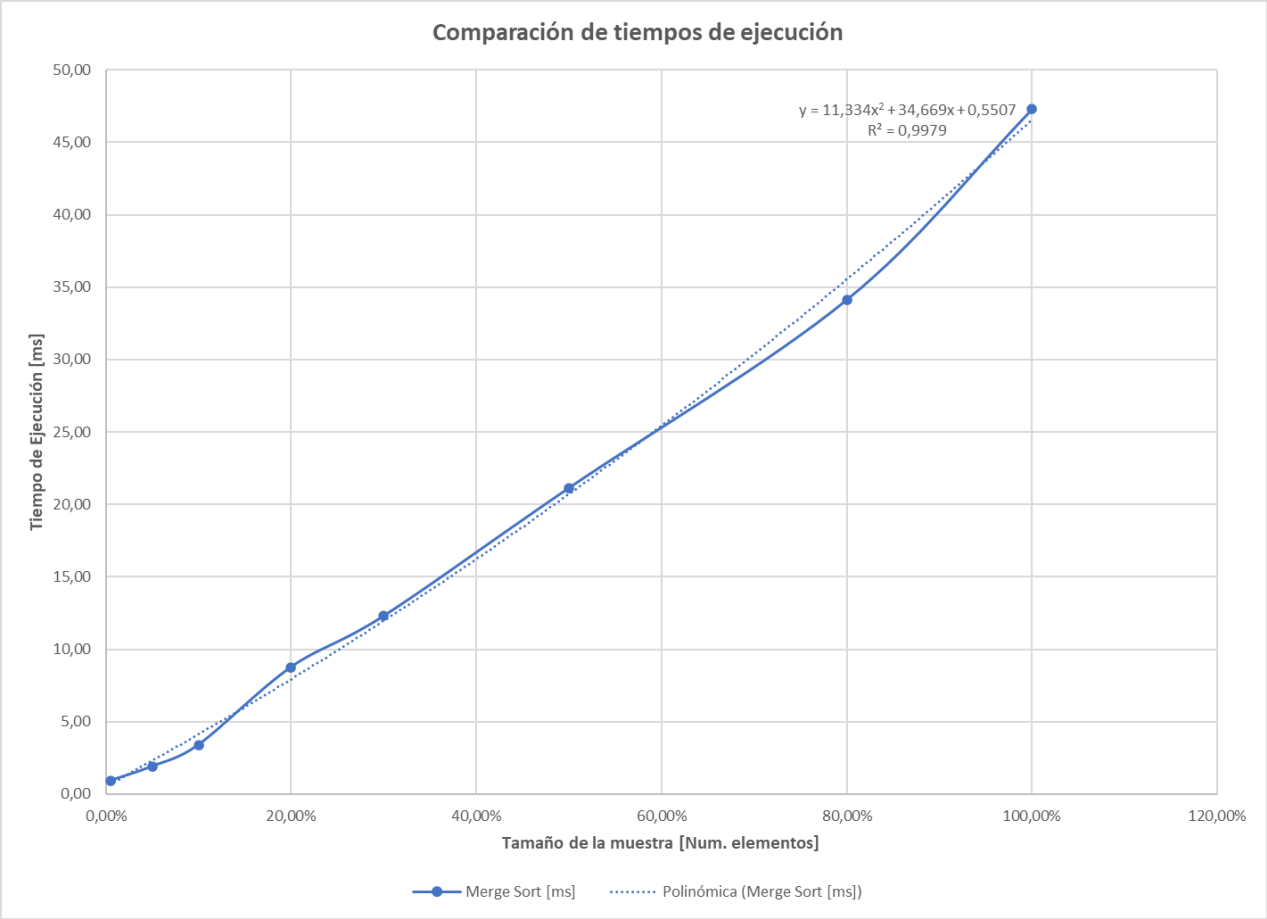
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	0.93
5 pct	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	1.92
10 pct	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	3.4
20 pct	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	8.77

30 pct	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	12.3
50 pct	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	21.14
80 pct	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	34.14
large	Una lista con todos los accidentes ocurridos dentro de un intervalo de hora dado, para un año y mes especifico	47.29

Graficas



Análisis

En este requerimiento 2 gracias tanto a los árboles binarios y la forma en la que está construida la carga de datos, es posible obtener grandes cantidades de datos que se encuentran en un rango de fechas de manera mucho más sencilla. También, la implementación de la tabla de hash permitió que se organizara los accidentes por su gravedad, esto facilito mucho más las comparaciones y la búsqueda de los datos. Para este requerimiento se tiene una complejidad O(n^2), esto dado porque en el código se encuentra un ciclo dentro de un ciclo a la hora de buscar la información en la lista que se encuentra dentro del rango de horas dado para un mes y un año en específico, esta complejidad se encuentra confirmada por la gráfica.

Requerimiento 3

```python
def req_3(data_structs, accidente, nombre_via):
    """
    Función que soluciona el requerimiento 3
    """
    accidentes =  data_structs["accidents"]
    lista = lt.newList("ARRAY_LIST")

    for elemento in lt.iterator(accidentes):
        if elemento["CLASE_ACC"].upper() == accidente.upper():
            if nombre_via.upper() in elemento["DIRECCION"].upper():
                lt.addLast(lista, elemento)

    merg.sort(lista, compareTime)

    largo = lt.size(lista)

    if largo < 3:
        sub_lista = lista
    else:
        sub_lista = lt.subList(lista, 1, 3)

    return largo, sub_lista
```

Entrada	Data_structs, tipo de accidente y el nombre de la via
Salidas	La cantidad de accidentes de un tipo ocurrido en una via y una lista con los 3 accidentes más recientes
Implementado (Sí/No)	Si, Implementado por Sebastian Palma Mogollón

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Guardar data_structs[“accidentes”] en accidentes	O (1)
Crear una nueva lista con lt.nerList y guardarla en lista	O (1)
Iterar elemento en accidentes con lt.iterator	O (n)
Si [“CLASE_ACC”] de elemento es igual a accidente y si nombre_via está dentro de [“DIRECCION”] de elemento, agregar elemento al final de lista con lt.addLast	O (1)
Organizar lista con merge.sort por fechas	O (n*Log(n))
Obtener el size de lista con lt.size y guarlarlo en largo	O (1)
Si largo es menor que 3, sub_lista es igual a lista	O (1)
Si no, hacer una sublista desde la posición 1a la 3 con lt.subList	O (1)
Retornar largo y sub_lista	O (1)
TOTAL	O (n*Log(n))

Pruebas Realizadas

Procesadores	Intel® Core™ i5-9300H CPU @2.40GHz
Memoria RAM	8.0 GB
Sistema Operativo	Windows 10 Home Single – 64 bits

Entrada	**Tiempo (ms)**
small	6.2
5 pct	44.4
10 pct	106.9
20 pct	225.3
30 pct	351.15
50 pct	638.29
80 pct	1056.24
large	1416.91

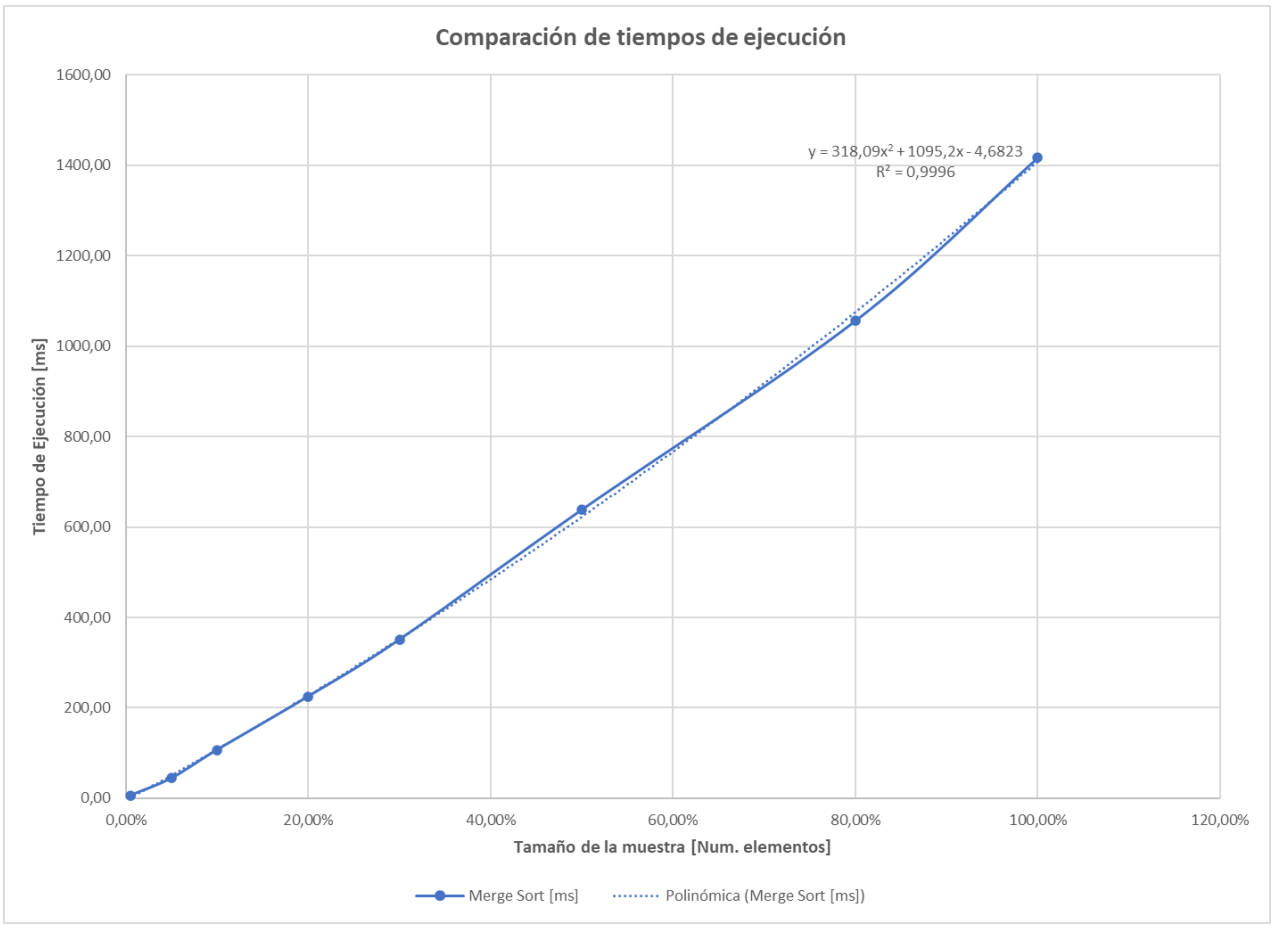
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	6.2
5 pct	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	44.4
10 pct	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	106.9
20 pct	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	225.3
30 pct	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	351.15
50 pct	Una lista con los 3 accidentes mas recientes	638.29

	ocurridos en una via y una clase en particular	
80 pct	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	1056.24
large	Una lista con los 3 accidentes mas recientes ocurridos en una via y una clase en particular	1416.91

Graficas


Comparación de tiempos de ejecución
1600,00
1400,00
1200,00
1000,00
800,00
600,00
400,00
200,00
0,00
Tiempo de Ejecución [ms]
y = 318,09x² + 1095,2x - 4,6823
R² = 0,9996
0,00%
20,00%
40,00%
60,00%
80,00%
100,00%
120,00%
Tamaño de la muestra [Num. elementos]
Merge Sort [ms]
Polinómica (Merge Sort [ms])

Análisis

En este requerimiento 3 gracias tanto a los árboles binarios y la forma en la que está construida la carga de datos, es posible obtener grandes cantidades de datos que se encuentran en un rango de fechas de manera mucho más sencilla. También, la implementación de la tabla de hash permitió que se organizara los accidentes por su gravedad, esto facilito mucho más las comparaciones y la búsqueda de los datos. Para este requerimiento se tiene una complejidad O (n*Log(n)), esto se debe al uso del algoritmo de orden merge sort que se usa para organizar todos los datos por fecha.

Requerimiento 4

Descripción

```
def req_4(data_structs, initialDate, finalDate,gravedad):
    """
    Función que soluciona el requerimiento 4
    """

    dates = om.values(data_structs['dateIndex'], finalDate, initialDate)
    new_list = lt.newList("ARRAY_LIST", compareIds)
    lista_codigos = []
    if lt.size(dates) != 0:
```

```
        for element in lt.iterator(dates):
            mapa = element['accidentIndex']
            entry = mp.get(mapa,gravedad)
            if entry is not None:
                lista_values = me.getValue(entry)['lstaccidents']

                for accidente in lt.iterator(lista_values):
                    if accidente["CODIGO_ACCIDENTE"] not in lista_codigos:
                        lt.addLast(new_list,accidente)
                        lista_codigos.append(accidente["CODIGO_ACCIDENTE"])

    size = lt.size(new_list)

    merg.sort(new_list, compareTime)

    if size < 5:
        sublist = new_list
    else:
        sublist = lt.subList(new_list,1,5)



    return size, sublist
```

Entrada	La estructura de datos con la información de los accidentes cargados, fecha inicial y final ingresadas por parámetro y gravedad de los accidentes a consultar.
Salidas	El tamaño de la lista con los accidentes que cumplen con los parámetros de entrada (el total de accidentes entre la fecha inicial y final con la gravedad dada) y los 5 accidentes más recientes que cumplen con los parámetros de entrada.
Implementado (Sí/No)	Si, implementado por Sara Leiva

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Sacar los accidentes entre la fecha inicial y final a partir de la función om.values() de los árboles, en donde el árbol está organizado a partir de las fechas de los accidentes como sus llaves.	O (1ogn + k) donde k es la cantidad de nodos en el rango.
Crear un nuevo arreglo con una función de comparación determinada.	O (1)
Crear una lista de Python llamada “lista_codigos”	O (1)

Condicional que pregunta si existen accidentes entre la fecha inicial y final dada.	O (1)
Se recorre la información dentro de “dates”. Es decir, se recorren las fechas que están en el intervalo dado.	O (n)
Se entra al mapa de la fecha que se encuentra dentro de “dates”.	O (1)
Se busca la pareja <llave, valor> dentro del mapa que contenga los accidentes como valores y la llave como la gravedad entrada por parámetro.	O (1)
Condicional que pregunta sí la pareja <llave, valor> existe dentro del mapa.	O (1)
Si la gravedad entrada por parámetro se encuentra en el mapa del año, sacar los valores que se encuentran dentro de la tabla de hash por tipo de gravedad. Aquello devuelve una lista con los accidentes que tienen esta gravedad y están dentro de la fecha.	O (1)
Iterar sobre la lista que contiene los accidentes.	O (n)
Se añade el accidente al arreglo creado sí se cumple la condicional.	O (1)
Se añade el código del accidente a la lista de Python para que no se añada el accidente dos veces al arreglo creado.	O (1)
Se saca el tamaño del arreglo con los accidentes encontrados.	O (1)
Se organizan los accidentes a partir de la fecha y la hora con merge.sort()	O(n*logn)
Si la lista de los accidentes es menor a 5, entonces se devuelve toda esta lista.	O(1)
Si la lista de los accidentes es mayor a 5 entonces se crea una sublista con los 5 accientes más recientes.	O(5)
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	Intel® Core™ i7-1165G7 @2.80GHz
Memoria RAM	15.6 GB
Sistema Operativo	Windows 11 Pro – 64 bits

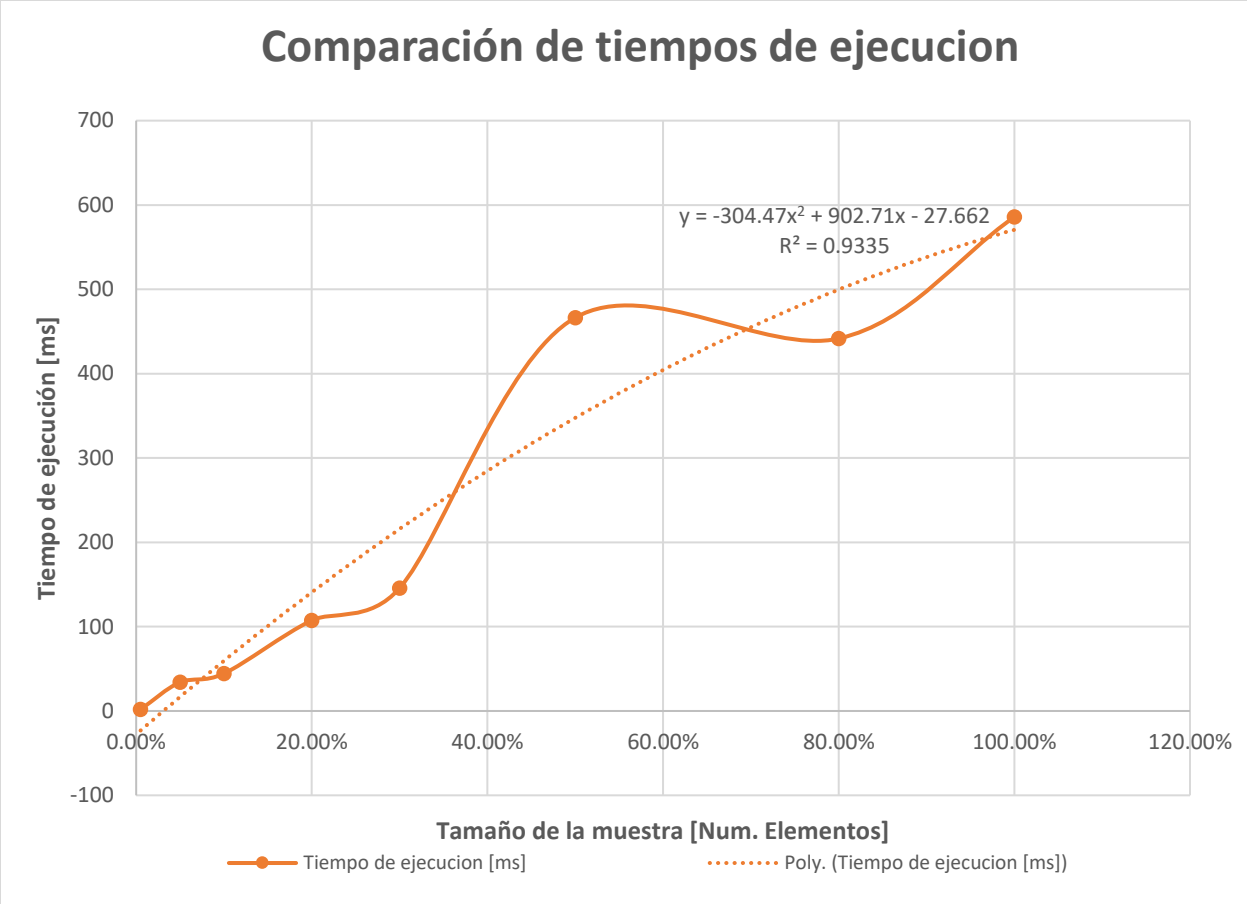
Entrada	**Tiempo (ms)**
small	1.91
5 pct	34.02
10 pct	44.4
20 pct	107.51
30 pct	145.75
50 pct	466.41

80 pct	441.65
large	585.73

Tablas de datos

Muestra	Salida	Tiempo (ms)
small	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	1.91
5 pct	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	34.02
10 pct	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	44.4
20 pct	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	107.51
30 pct	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	145.75
50 pct	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	466.41
80 pct	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	441.65
large	Lista con los 5 accidentes más recientes dentro de un rango de fechas y su tamaño.	585.73

Graficas



Análisis

Los árboles nos permiten obtener ciertos elementos entre un rango de fechas al ser una estructura de datos organizada de manera jerárquica. Lo anterior permite entrar a un gran tamaño de datos y poder sacar la información deseada a partir de una llave inicial y final, en este caso fechas iniciales y finales. Además, la tabla de hash que se implementó en la carga de datos y organizó a los accidentes tomando a su gravedad como llave permite entrar una vez a la carga y poder obtener los datos de manera rápida. Por otra parte, código que se utilizó para resolver este requerimiento tiene una complejidad temporal de O(n^2) gracias a que hay un ciclo dentro de otro ciclo. Así mismo, esta complejidad se evidencia en la gráfica gracias a que los tiempos de comparación están acotados por una función polinomial de grado 2.

Requerimiento 5

Descripción

```
def req_5(data_structs, dateMin, dateMax, loc):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    l = om.values(data_structs['dateIndex'], dateMax, dateMin)
    lst = lt.newList("ARRAY_LIST")
    cod = []
    for value in lt.iterator(l):
        for ls in lt.iterator(value['lstaccidents']):
            if (ls["LOCALIDAD"] == loc) and (ls["CODIGO_ACCIDENTE"] not in cod):
                lt.addLast(lst, ls)
                cod.append(ls["CODIGO_ACCIDENTE"])
    tot = lt.size(lst)
    acc = []

    merg.sort(lst, compareTime)
    if lt.size(lst) <= 10:
        info = lst
    else:
        info = lt.subList(lst, 1, 10)
```

```
    for lstdate in lt.iterator(info):
        table = [
            lstdate["CODIGO_ACCIDENTE"],
            lstdate["FECHA_HORA_ACC"],
            lstdate["DIA_OCURRENCIA_ACC"],
            lstdate["DIRECCION"],
            lstdate["GRAVEDAD"],
            lstdate["CLASE_ACC"],
            lstdate["LATITUD"],
            lstdate["LONGITUD"]
            ]
        acc.append(table)
    return tot, acc
```

Entrada	La estructura de datos, fecha inicial del mes, fecha final del mes, localidad de Bogotá
Salidas	Número total de accidentes dentro del rango de fechas, lista de los 10 accidentes más recientes dentro del rango y la localidad
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Guardar los valores entre la fecha inicial y la fecha final con om.values() en l	O (log n)
Crear un array list llamado lst	O (1)
Crear la lista vacía cod	O (1)
Por cada value en l, recorrer la lista de accidentes en value con ls	O (n^2)
Si la localidad de ls es igual a loc y el codigo de accidente de ls no está en cod, agregar ls a lst y agregar el código del accidente de ls en cod	O (1)
Asignar a tot como el tamaño de lst	O (1)
Crear la lista vacía acc	O (1)
Organizar lst con respecto al tiempo con merge sort	O (n log n)
Si el tamaño de lst es menor o igual a 10 asignar info como lst	O (1)
En caso de que lst sea mayor a 10 asignar a info como la sublista de los primeros 10 elementos de lst	O (1)
Por cada lstdate en info, asignar a la lista table con los valores del código del accidente, fecha y hora, día, localidad, dirección, gravedad, clase de accidente, latitud y longitud de lstdate	O (n)
Agregar tabla a acc	O (1)
Retornar tot y acc	O (1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	AMD Ryzen 3 5300U with Radeon Graphics 2.60 GHz
Memoria RAM	8.0 GB (5.86 GB)
Sistema Operativo	Windows 11 Pro – 64 bits

Entrada	**Tiempo (ms)**
small	0.90
5 pct	3.87
10 pct	4.48
20 pct	8.97
30 pct	14.48
50 pct	18.51
80 pct	32.70
large	44.96

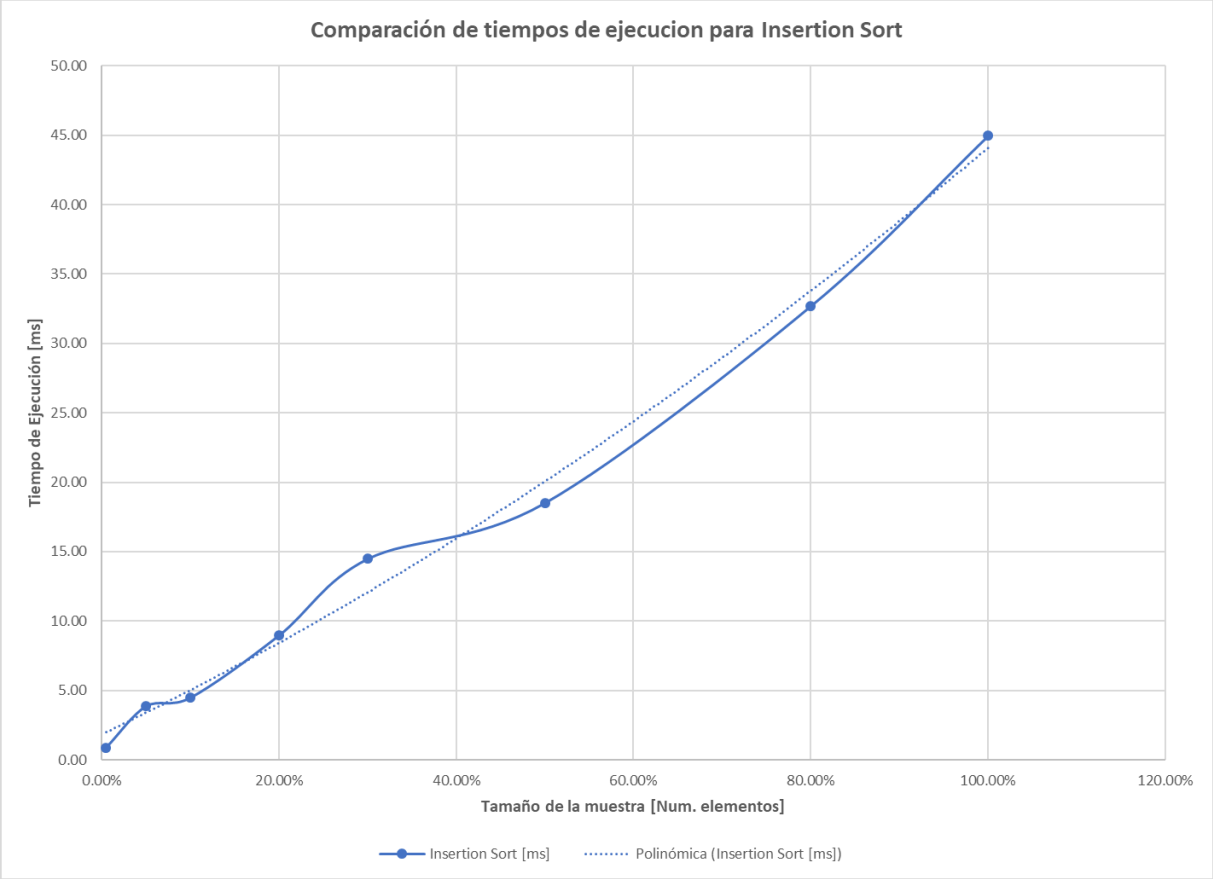
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	0.90
5 pct	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	3.87
10 pct	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	4.48
20 pct	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	8.97
30 pct	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	14.48
50 pct	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	18.51
80 pct	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	32.70
large	Lista con los 10 accidentes más recientes y el número total de accidentes en la localidad en el mes y año	44.96

Grafica

La gráfica con la representación de las pruebas realizadas.



Análisis

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

Gracias a la organización de los árboles, se puede acceder eficientemente a la sub lista que está dentro de un rango de valores. Por cada nodo en la lista del intervalo se recorre su lista de accidentes, generando la complejidad de O(n^2), los valores son luego agregado a una lista y organizados, pero la complejidad temporal de estas acciones es menor a n^2, dejando así la complejidad como O(n^2)

La grafica no presenta un comportamiento cuadrático tan marcado posiblemente debido a la posibilidad de que en las pruebas realizadas hubiera una cantidad menor de nodos o accidentes que redujeron los tiempos en las pruebas.

Requerimiento 6

Descripción

```
def req_6(data_structs, dateMin, dateMax, coord, rad, num):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    global point
    point = coord
    l = om.values(data_structs['dateIndex'], dateMax, dateMin)
    lst = lt.newList("ARRAY_LIST")
    cod = []
    for value in lt.iterator(l):
        for ls in lt.iterator(value['lstaccidents']):
            coordPoint = (float(ls["LATITUD"]), float(ls["LONGITUD"]))
            d = CalculateDistance(coordPoint, coord)
            if (d <= rad) and (ls["CODIGO_ACCIDENTE"] not in cod):
                lt.addLast(lst, ls)
                cod.append(ls["CODIGO_ACCIDENTE"])
    acc = []

    merg.sort(lst, compareDistance)
    if lt.size(lst) <= num:
        info = lst
    else:
        info = lt.subList(lst, 1, num)
    for lstdate in lt.iterator(info):
        table = [
            lstdate["CODIGO_ACCIDENTE"],
            lstdate["FECHA_HORA_ACC"],
            lstdate["DIA_OCURRENCIA_ACC"],
            lstdate["LOCALIDAD"],
            lstdate["DIRECCION"],
            lstdate["GRAVEDAD"],
            lstdate["CLASE_ACC"],
            lstdate["LATITUD"],
            lstdate["LONGITUD"]
            ]
        acc.append(table)
    return acc
```

Entrada	Estructura de datos, fecha inicial del mes, fecha final del mes, coordenadas del centro del área, radio del área, número de accidentes
Salidas	Lista de los accidentes occurridos dentro del radio del area.
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Declarar la variable global point	O (1)
Guardar en point coord	O (1)
Guardar los valores entre la fecha inicial y la fecha final con om.values() en l	O (log n)
Crear un array list llamado lst	O (1)
Crear la lista vacia cod	O (1)
Por cada value en l, recorrer la lista de accidentes en value con ls	O (n^2)
Crear la tupla coordPoint con la latitud y longitud de ls	O (1)
Calcular la distancia entre coordPoint y coord y guradarlo en d	O (1)
Si la d es menor o igual a rad y el codigo de accidente de ls no esta en cod, agregar ls a lst y agregar el codigo del accidente de ls en cod	O (1)
Crear la lista vacia acc	O (1)
Organizar lst con respecto a la distancia a coord con merge sort	O (n log n)
Si el tamano de lst es menor o igual a num, asignar info como lst	O (1)
En caso de que lst sea mayor a num, asignar a info como la sublista de los primeros elementos de lst hasta num	O (1)
Por cada lstdate en info, asignar a la lista table con los valores del codigo del accidente, fecha y hora, dia, localidad, direccion, gravedad, clase de accidente, latitud y longitud de lstdate	O (n)
Agregar tabla a acc	O (1)
Retornar acc	O (1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	AMD Ryzen 3 5300U with Radeon Graphics 2.60 GHz
Memoria RAM	8.0 GB (5.86 GB)
Sistema Operativo	Windows 11 Pro – 64 bits

Entrada	**Tiempo (ms)**
small	0.75
5 pct	5.08

10 pct	9.88
20 pct	12.42
30 pct	28.76
50 pct	36.35
80 pct	78.15
large	97.44

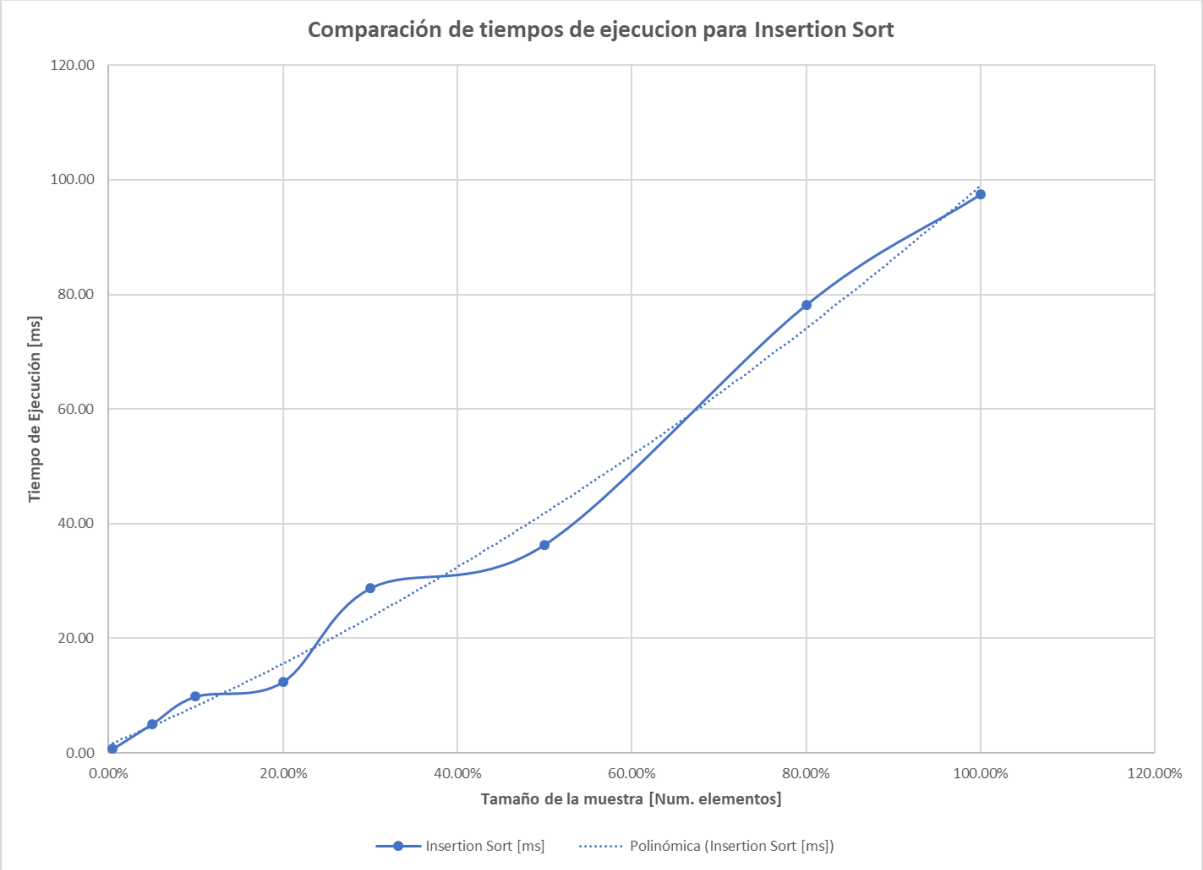
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Lista con los n elementos mas cercanos en el radio de la coordenada dada	0.75
5 pct	Lista con los n elementos mas cercanos en el radio de la coordenada dada	5.08
10 pct	Lista con los n elementos mas cercanos en el radio de la coordenada dada	9.88
20 pct	Lista con los n elementos mas cercanos en el radio de la coordenada dada	12.42
30 pct	Lista con los n elementos mas cercanos en el radio de la coordenada dada	28.76
50 pct	Lista con los n elementos mas cercanos en el radio de la coordenada dada	36.35
80 pct	Lista con los n elementos mas cercanos en el radio de la coordenada dada	78.15
large	Lista con los n elementos mas cercanos en el radio de la coordenada dada	97.44

Grafica

La gráfica con la representación de las pruebas realizadas.



Análisis

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

Gracias a la organización de los árboles, se puede acceder eficientemente a la sub lista que está dentro de un rango de valores. Por cada nodo en la lista del intervalo se recorre su lista de accidentes, generando la complejidad de O(n^2), para el cálculo de distancias solo se utilizan operaciones aritméticas por lo que tiene una complejidad de O(1), después de eso los valores son luego agregado a una lista y organizados, pero la complejidad temporal de estas acciones es menor a n^2, dejando así la complejidad como O(n^2)

La grafica no presenta un comportamiento cuadrático tan marcado posiblemente debido a la posibilidad de que en las pruebas realizadas hubiera una cantidad menor de nodos o accidentes que redujeron los tiempos en las pruebas.

Requerimiento 7

Descripción

```
def req_7(data_structs, mes, anio):
    """
    Función que soluciona el requerimiento 7
    """

    dia = 1
    primeros_ultimos = lt.newList("ARRAY_LIST")
    lista_total = lt.newList('ARRAY_LIST',compareIds)


    if mes in ['1','3', '5','7','8','10','12']:
        while dia <= 31:
            new_list = lt.newList('ARRAY_LIST',compareIds)
            fecha = anio + "/" + mes + "/" + str(dia)
            fecha = datetime.datetime.strptime(fecha, '%Y/%m/%d')
            fecha = fecha.date()
            accident_date = om.get(data_structs['dateIndex'], fecha)

            if accident_date != None:
                map_values = me.getValue(accident_date)['accidentIndex']
                values = mp.valueSet(map_values)
                for accident_lista in lt.iterator(values):
                    for accident in lt.iterator(accident_lista['lstaccidents']):
                        lt.addLast(new_list,accident)
                        if lt.isPresent(lista_total,accident) == 0:
                            lt.addLast(lista_total,accident)

                merg.sort(new_list,compareTime2)
                lt.addLast(primeros_ultimos,lt.getElement(new_list,1))
                lt.addLast(primeros_ultimos,lt.getElement(new_list,lt.size(new_li
st)))


            dia +=1

    if mes in ['4', '6', '9', '11']:
        while dia <= 30:
            new_list = lt.newList('ARRAY_LIST',compareIds)
            fecha = anio + "/" + mes + "/" + str(dia)
            fecha = datetime.datetime.strptime(fecha, '%Y/%m/%d')
            fecha = fecha.date()
            accident_date = om.get(data_structs['dateIndex'], fecha)
```

```
            if accident_date != None:
                map_values = me.getValue(accident_date)['accidentIndex']
                values = mp.valueSet(map_values)
                for accident_lista in lt.iterator(values):
                    for accident in lt.iterator(accident_lista['lstaccidents']):
                        lt.addLast(new_list,accident)
                        if lt.isPresent(lista_total,accident) == 0:
                            lt.addLast(lista_total,accident)

                merg.sort(new_list,compareTime2)
                lt.addLast(primeros_ultimos,lt.getElement(new_list,1))
                lt.addLast(primeros_ultimos,lt.getElement(new_list,lt.size(new_li
st)))


            dia +=1

    else:
        if anio == '2020' or anio == '2016':
            while dia <= 29:
                new_list = lt.newList('ARRAY_LIST',compareIds)
                fecha = anio + "/" + mes + "/" + str(dia)
                fecha = datetime.datetime.strptime(fecha, '%Y/%m/%d')
                fecha = fecha.date()
                accident_date = om.get(data_structs['dateIndex'], fecha)

                if accident_date != None:
                    map_values = me.getValue(accident_date)['accidentIndex']
                    values = mp.valueSet(map_values)
                    for accident_lista in lt.iterator(values):
                        for accident in
lt.iterator(accident_lista['lstaccidents']):
                            lt.addLast(new_list,accident)
                            if lt.isPresent(lista_total,accident) == 0:
                                lt.addLast(lista_total,accident)

                    merg.sort(new_list,compareTime2)
                    lt.addLast(primeros_ultimos,lt.getElement(new_list,1))
                    lt.addLast(primeros_ultimos,lt.getElement(new_list,lt.size(ne
w_list)))


                dia +=1
        else:
```

```
            while dia <= 28:
                new_list = lt.newList('ARRAY_LIST',compareIds)
                fecha = anio + "/" + mes + "/" + str(dia)
                fecha = datetime.datetime.strptime(fecha, '%Y/%m/%d')
                fecha = fecha.date()
                accident_date = om.get(data_structs['dateIndex'], fecha)

                if accident_date != None:
                    map_values = me.getValue(accident_date)['accidentIndex']
                    values = mp.valueSet(map_values)
                    for accident_lista in lt.iterator(values):
                        for accident in
lt.iterator(accident_lista['lstaccidents']):
                            lt.addLast(new_list,accident)
                            if lt.isPresent(lista_total,accident) == 0:
                                lt.addLast(lista_total,accident)

                    merg.sort(new_list,compareTime2)
                    lt.addLast(primeros_ultimos,lt.getElement(new_list,1))
                    lt.addLast(primeros_ultimos,lt.getElement(new_list,lt.size(ne
w_list)))


                dia +=1

    size = lt.size(lista_total)

    return primeros_ultimos, lista_total, size
```

Entrada	La estructura de datos con la información de los accidentes cargados, el mes y el año ingresados por parámetro.
Salidas	La lista de los primeros y últimos accidentes por día, la lista con todos los accidentes dentro del mes y el total de accidentes dentro del mes que cumplen con los parámetros.
Implementado (Sí/No)	Si, implementado por Sara Leiva.

Análisis de complejidad

Pasos	**Complejidad**

Aginarle un valor a la variable “día” que corresponde al primer día que se va a buscar.	O (1)
Crear un nuevo arreglo llamado “primeros_ultimos” que contendrá el primer y último accidente de cada mes.	O (1)
Crear un nuevo arreglo llamado “lista_total” que contendrá todos los accidentes por mes.	O (1)
Condicional para verificar sí el mes tendrá cierta cantidad de días.	O (1)
Ciclo que se detiene cuando la variable “día” es mayor a 31.	O (31)
Crear un nuevo arreglo (“new_list”) que contendrá los accidentes por día del mes y año indicados.	O (1)
Asignarle a una variable la fecha que se está buscando.	O (1)
Convertir la fecha a través de la librería datetime para que se pueda buscar en el árbol RBT.	O (1)
Obtener una pareja llave-valor de la estructura de datos donde la llave es la fecha establecida y su valor es un mapa con los accidentes que ocurrieron en esta fecha.	O (1ogn)
Condicional que permite que el código siga sí hay accidentes en la fecha establecida.	O (1)
Obtener el mapa de los accidentes que ocurrieron en esta fecha.	O (1)
Obtener la lista de listas de accidentes dentro del mapa encontrado.	O (1)
Iterar sobre esta lista de listas de accidentes que se encuentra en el mapa.	O (n)
Iterar sobre cada lista de accidentes.	O (n)
Añadir el accidente a “new_list”.	O (1)
Si el accidente no se encuentra en la lista de todos los accidentes que ocurrieron en ese mes, añadirlo a la lista total de accidentes.	O (1)
Organizar los accidentes por fecha con merge.sort()	O (n*logn)
Añadir el accidente más y menos reciente a la lista “primeros_ultimos”.	O (1)
Sumarle 1 a la variable “día”	O (1)
Si el mes ingresado por parámetro no cumple con la anterior condicional, hacer exactamente lo mismo pero con una cantidad de días diferente. Lo anterior solo cambia el “while” y su complejidad será de O(30) porque será un día menos.	
Si el mes ingresado por parámetro no cumple con ninguna de las anteriores condicionales hacer exactamente lo mismo pero cambia la cantidad de días evaluada. En este último caso se genera otra condicional y, dependiendo del año, lo único que cambia es la complejidad del “while” que será de	

O(29) u O(28) dependiendo de hasta donde llegue el día.	
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	Intel® Core™ i7-1165G7 @2.80GHz
Memoria RAM	15.6 GB
Sistema Operativo	Windows 11 Pro – 64 bits

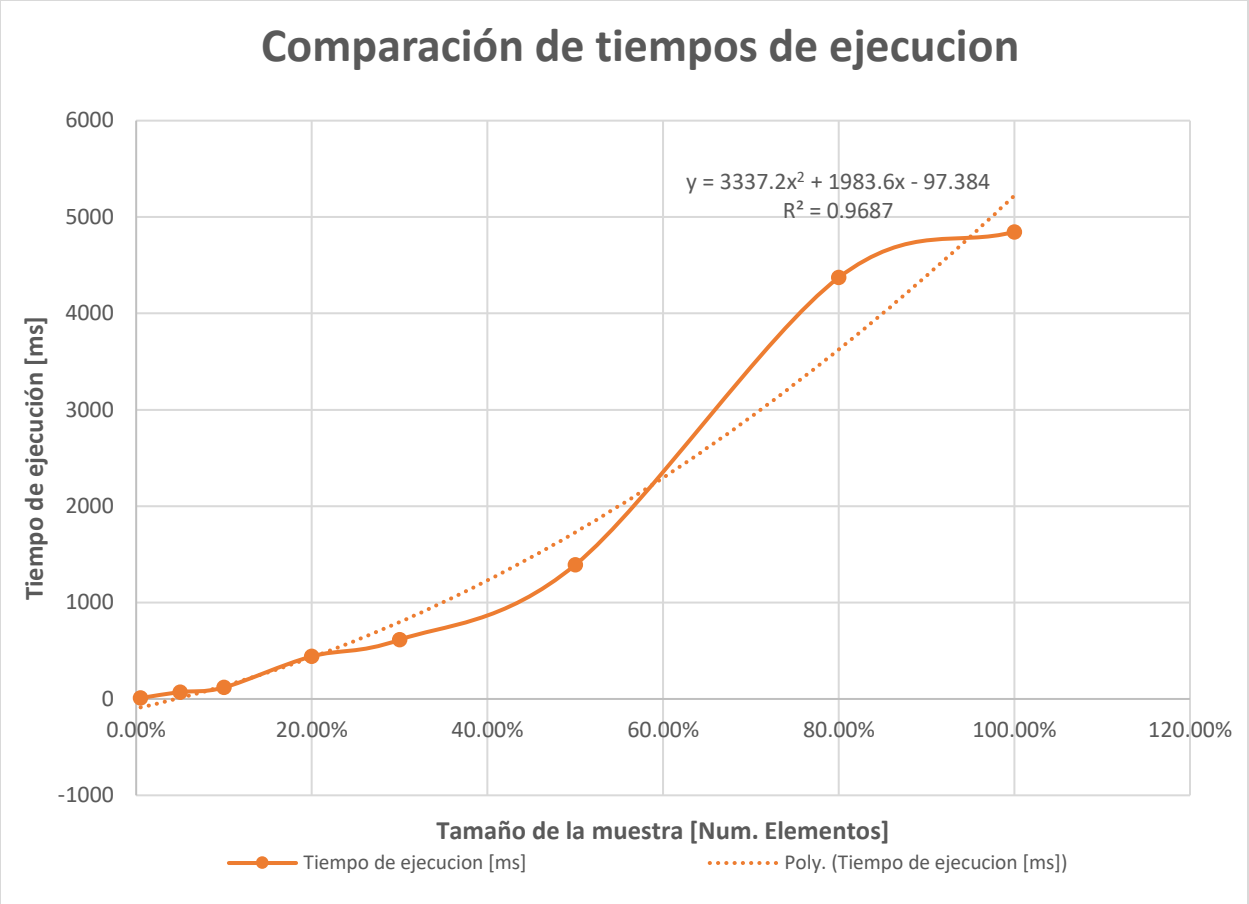
Entrada	**Tiempo (ms)**
small	9.07
5 pct	71.86
10 pct	119.1
20 pct	441.61
30 pct	614.23
50 pct	1391.59
80 pct	4371.66
large	4846.11

Tablas de datos

Muestra	**Salida**	**Tiempo (ms)**
small	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	9.07
5 pct	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	71.86
10 pct	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	119.1

20 pct	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	441.61
30 pct	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	614.23
50 pct	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	1391.59
80 pct	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	4371.66
large	Lista de todos los accidentes dentro de un mes y su tamaño y lista con el accidente más reciente y menos reciente por cada día.	4846.11

Graficas



Análisis

El código que se utilizó para resolver este requerimiento tiene una complejidad de O(n^2) gracias a que hay un ciclo dentro de otro ciclo y siendo n el número de elementos que se iteran tenemos que el peor caso del algoritmo es n*n. Por otra parte, el árbol RBT implementado en la carga de datos nos permitió obtener de manera rápida todos los datos que se encuentran en los parámetros dados, filtrando mejor los datos y mejorando el tiempo de ejecución.

Requerimiento 8

```python
def req_8(data_structs,initialDate, finalDate, clase):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    dates = om.values(data_structs['dateIndex'], finalDate, initialDate)
    new_list = lt.newList("ARRAY_LIST")
    if lt.size(dates) != 0:
        for element in lt.iterator(dates):
            mapa = element['accidentIndex']
            entry = mp.get(mapa,clase)
            if entry is not None:
                lista_values = me.getValue(entry)['lstaccidents']
                for accidente in lt.iterator(lista_values):
                    lt.addLast(new_list,accidente)

    merg.sort(new_list,compareTime)

    mapa = folium.Map(location = [4.636057, -74.110094], zoom_start=13)

    for elementos in lt.iterator(new_list):
        if str(elementos["GRAVEDAD"]) == "SOLO DANOS":
            folium.Marker([float(elementos["LATITUD"]), float(elementos["LONGITUD"])], popup = "SOLO DANOS", icon=folium.Icon(color="green",

        if str(elementos["GRAVEDAD"]) == "CON HERIDOS":
            folium.Marker([float(elementos["LATITUD"]), float(elementos["LONGITUD"])], popup = "CON HERIDOS", icon=folium.Icon(color="blue",

        if str(elementos["GRAVEDAD"]) == "CON MUERTOS":
            folium.Marker([float(elementos["LATITUD"]), float(elementos["LONGITUD"])], popup = "CON MUERTOS", icon=folium.Icon(color="red",

    size = lt.size(new_list)

    return mapa, size
```

Entrada	Data_structs, fecha inicial, fecha final, clase de accidente
Salidas	Cantidad de accidentes de la clase de accidentes ocurridos entre la fecha inicial y la fecha final con un mapa iterativo donde se pueden observar
Implementado (Sí/No)	Si,

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Retornar todos los valores del data_structs en un rango con om.values y guardar los valores en dates	O (1ogn + k) donde k es la cantidad de nodos en el rango.
Hacer una nueva lista con lt.newlist y guardarla en new_list	O (1)
Si el size, obtenido con lt.size, de dates es diferente de 0	O (1)
Iterar element en dates con lt.iterator	O (n)
Guardar element[“accidentIndex”] en mapa	O (1)
Buscar si clase está dentro de mapa con mp.get y guardarlo en entry	O (n)
Si entry no es None, obtener el valor de entry con me.getValue y guardarlo en lista_values	O (n)
Iterar accidente en lista_values con lt.iterator	O (n)
Agregar accidente al final de new_list con lt.addLast	O (1)
Organizar new_list con merg.sort por fechas	O (n*Log(n))
Crear un mapa de Bogota con la librería folium con folium.Map y guardarlo en mapa	O (1)

Iterar elementos en new_list con lt..iterator	O (n)
Si [“GRAVEDAD”] de elementos es igual a “SOLO DANOS”, se crea un marcador de color verde con título “SOLO DANOS” y se agrega al mapa	O (n)
Si [“GRAVEDAD”] de elementos es igual a “CON HERIDOS”, se crea un marcador de color azul con título “CON HERIDOS” y se agrega al mapa	O (n)
Si [“GRAVEDAD”] de elementos es igual a “CON MUERTO”, se crea un marcador de color rojo con título “CON MUERTOS” y se agrega al mapa	O (n)
Obtener el size de new_list con lt.size y guardarlo en size	O (n)
Retornar mapa y size	O (1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	Intel® Core™ i5-9300H CPU @2.40GHz
Memoria RAM	8.0 GB
Sistema Operativo	Windows 10 Home Single – 64 bits

Entrada	**Tiempo (ms)**
small	7.28
5 pct	33.08
10 pct	73.05
20 pct	152.72
30 pct	242.92
50 pct	435.55
80 pct	703.30
large	1371.58

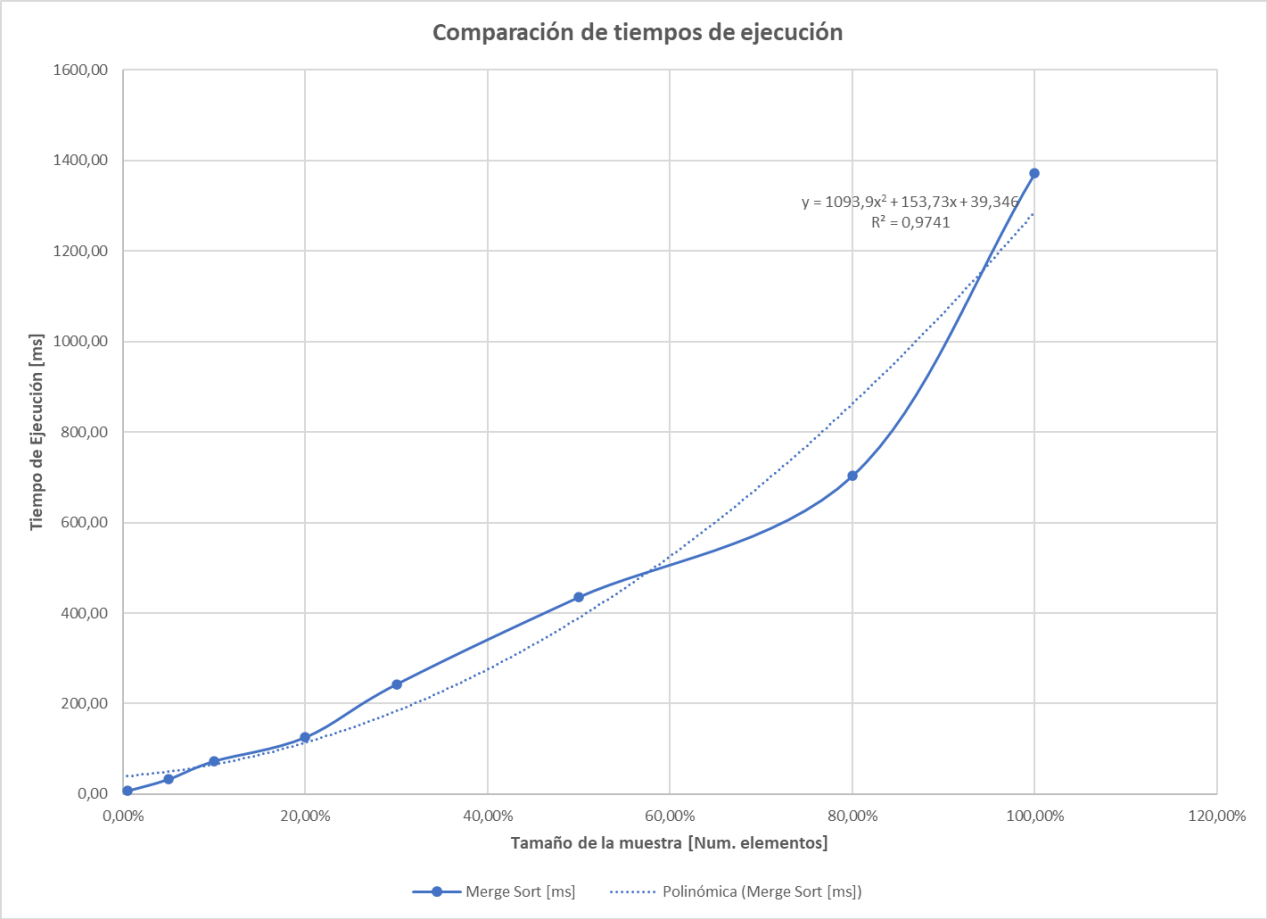
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	7.28
5 pct	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un	33.08

	rango de fechas y una clase dada	
10 pct	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	73.05
20 pct	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	152.72
30 pct	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	242.92
50 pct	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	435.55
80 pct	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	703.30
large	Un mapa iterativo donde es posible visualizar los accidentes ocurridos en un rango de fechas y una clase dada	1371.58

Graficas



Análisis

En este requerimiento 8 gracias tanto a los árboles binarios y la forma en la que está construida la carga de datos, es posible obtener grandes cantidades de datos que se encuentran en un rango de fechas de manera mucho más sencilla. También, la implementación de la tabla de hash permitió que se organizara los accidentes por su gravedad, esto facilito mucho más las comparaciones y la búsqueda de los datos. Para este requerimiento se tiene una complejidad O(n^2), esto dado porque en el código se encuentra un ciclo dentro de un ciclo a la hora de buscar la información en la lista que se encuentra dentro del rango de fechas, esta complejidad se encuentra confirmada por la grafica