ANÁLISIS DEL RETO 3

Alejandro Pardo Sánchez, 202223709, a.pardos2@uniandes.edu.co

Joseph Eli Pulido Gómez, 202211365, je.pulidog1@uniandes.edu.co

Santiago González Serna, 202021226, s.gonzalezs@uniandes.edu.co

Carga de datos

Descripción

```
def new_data_structs(tipo_mapa, factor_carga, tipo_arbol):
    """
    Inicializa las estructuras de datos del modelo. Las crea de
    manera vacía para posteriormente almacenar la información.
    """
    #TODO: Inicializar las estructuras de datos
    data_structs = {"DATOS_TODOS": None,
                    "Fecha_Occur": None,
                    "Hora_Occur": None,
                    "Anios": None,
                    "Clase_accid": None
                    }
    data_structs["DATOS_TODOS"] = lt.newList("ARRAY_LIST")

    data_structs["Fecha_occur"] = om.newMap(omaptype=tipo_arbol,
                                            comparefunction=compareFecha)

    data_structs["Hora_occur"] = om.newMap(omaptype=tipo_arbol,
                                           comparefunction=compareHora)

    data_structs["Anios"] = mp.newMap(20,
                                      maptype=tipo_mapa ,
                                      loadfactor=factor_carga,
                                      cmpfunction=compare_by_anio)

    data_structs["Clase_accid"] = mp.newMap(20,
                                      maptype=tipo_mapa ,
                                      loadfactor=factor_carga,
                                      cmpfunction=compare_by_clase)
    return data_structs
```

Entrada	- El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente.
Salidas	Un diccionario
Implementado (Sí/No)	Sí

Mapa conceptual de la carga de datos

Realizamos una estructura compleja para resolver los requerimientos del reto, en la llave ["DATOS_TODOS"] existe una lista ADT que contiene todos los datos de accidentes.

La llave ["Fecha_occur"] tiene un mapa ordenado (arbol) que tiene como llaves las fechas y como valores los accidentes que ocurrieron en esa fecha.

El mapa no ordenado llamado ["Anios"] contiene llaves de los años, y valores los accidentes que ocurrieron en ese año, adicionalmente, tiene otra llave que es "Mes", esta llave es un mapa no ordenado que tiene llaves de meses y valores, los accidentes del mes correspondiente, teniendo en cuenta el año también, además esta llave de "Mes" tiene otra llave llamada "Hora_occur_om" que es un mapa ordenado (árbol) que tiene como llaves las horas y valores los accidentes que ocurrieron en esas horas, teniendo en cuenta, el mes y el año anterior, adicionalmente "Mes" tiene otra llave que se llama "Día" esta es un mapa ordenado (árbol) que organiza por días los accidentes, las llaves son los días, y los valores son los accidentes que ocurrieron en ese día, teniendo en cuenta el mes y el año anterior.

Por último, la llave ["Clase_accid"] es un mapa no ordenado que tiene como llaves las clases de accidentes y como valores los datos que tienen ese tipo de accidente.

En la imagen anterior falta la llave "Día" dentro del mapa de Anios

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

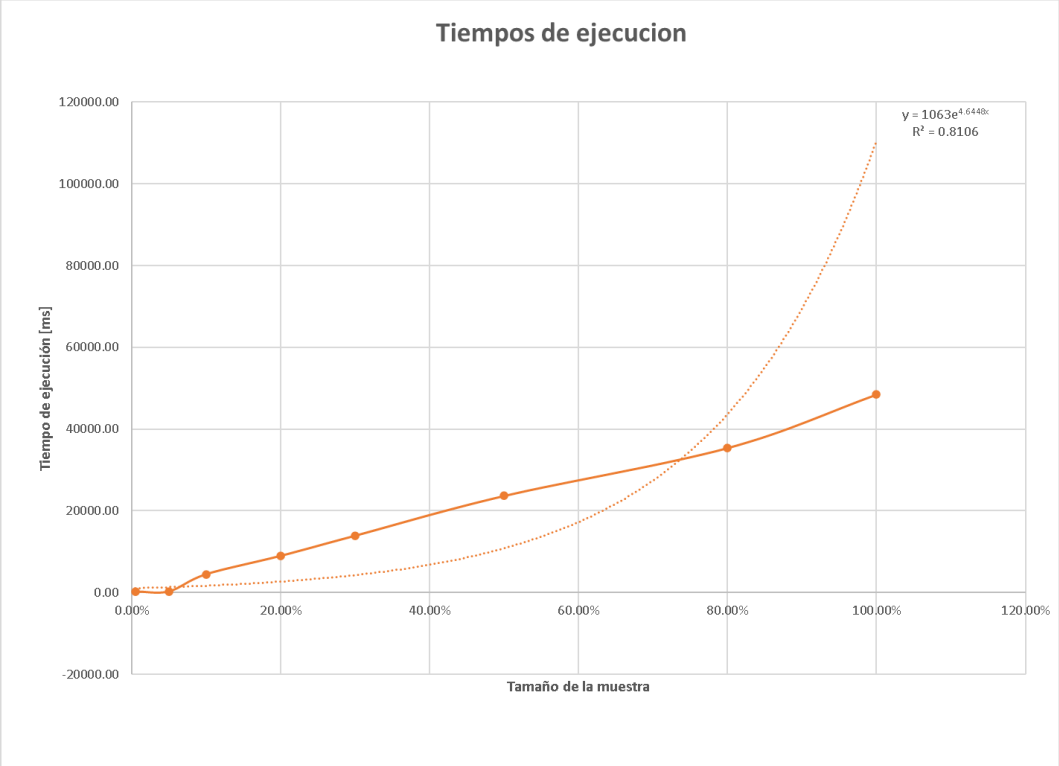
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	252.11
5.00%	23580.00	260.60
10.00%	47160.00	4469.12
20.00%	117900.00	8978.37
30.00%	235800.00	13914.54
50.00%	353700.00	23659.05
80.00%	542340.00	35404.92
100.00%	778141.00	48497.46

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.

La línea de tendencia muestra un crecimiento constante, donde cada vez la pendiente aumenta, puede relacionarse a la complejidad: O(n) o a una O(log(n)).

Requerimiento 1

Descripción

```python
def req_1(data_structs, fecha1, fecha2):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    resultado = lt.newList("ARRAY_LIST")

    om_fechas = data_structs["Fecha_occur"]
    for fechas in lt.iterator(om.values(om_fechas, fecha1, fecha2)):
        for eleme in lt.iterator(fechas["Datos_fecha"]):
            lt.addLast(resultado, eleme)
    merg.sort(resultado, cmp_crimenes_by_reciente_antiguo)
    return resultado
```

Entrada	- El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. - Fecha de inicio del período. - Fecha final del período.
Salidas	Una lista ADT con los accidentes que ocurrieron en la ciudad durante un intervalo de fechas específico
Implementado (Sí/No)	Sí

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

K < n, debido a que no se recorren todos los datos sino solo los del intervalo especifico.

Pasos	**Complejidad**
Creacion Array_list	O(k)
Om.values	O(k) k = cantidad de datos en el intervalo
For	O(k)
addLast	O(1)
Merge sort	O(k*log(k))
TOTAL	***O(k*log(k))***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

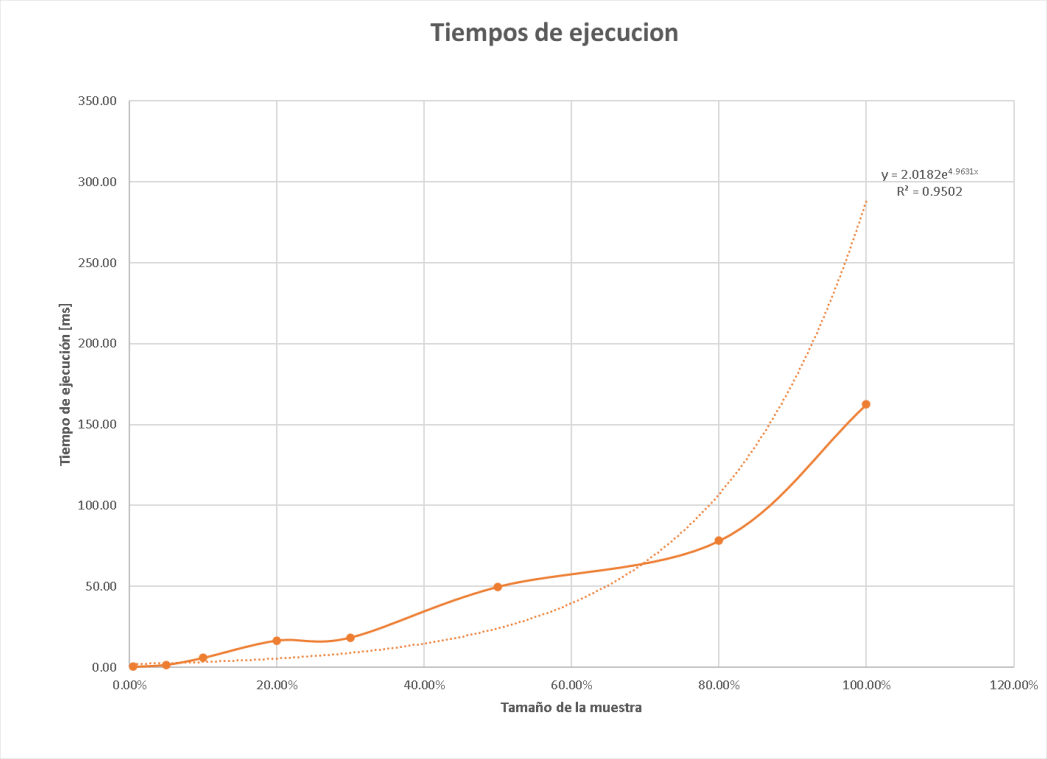
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	0.357
5.00%	23580.00	1.592
10.00%	47160.00	5.857
20.00%	117900.00	16.586
30.00%	235800.00	18.451
50.00%	353700.00	49.823
80.00%	542340.00	78.139
100.00%	778141.00	162.495

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.

La línea de tendencia muestra un crecimiento constante, donde cada vez la pendiente aumenta, puede relacionarse a la complejidad planteada: O(k*log(k)), por lo tanto, es posible inferir parcialmente que la complejidad que se calculó corresponde con la complejidad evidenciada en la gráfica.

Requerimiento 2

Descripción

```python
def req_2(data_structs, year, mes, tiempo1, tiempo2):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    resultado = lt.newList("ARRAY_LIST")

    dic_anio = mp.get(data_structs["Anios"], year)
    if not dic_anio:
        return False
    dic_anio=me.getValue(dic_anio)
    mes = mp.get(dic_anio["Mes"], mes)
    if not mes:
        return False
    lt_mes = me.getValue(mes)
    om_hora = lt_mes["Hora_occur_om"]
    for eleme in lt.iterator(om.values(om_hora, tiempo1, tiempo2)):
        for dato in lt.iterator(eleme):
            lt.addLast(resultado, dato)
    merg.sort(resultado, cmp_crimenes_by_reciente_antiguo)
    return resultado
```

Entrada	- El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. - El año - El mes - Tiempo inicial - Tiempo final
Salidas	Una lista de DISClib con los accidentes ocurridos en un intervalo de tiempo del día (desde una hora y minutos iniciales hasta una hora y minutos finales) para un año y mes dados
Implementado (Sí/No)	Sí

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

K < n, debido a que no se recorren todos los datos sino solo los del intervalo especifico.

Pasos	**Complejidad**
Creacion Array_list	O(1)
Get	O(1)
Om.values	O(k) k = cantidad de datos en el intervalo
For anidado	O(k^2)
addLast	O(1)
Merge sort	O(k*log(k))
TOTAL	***O(k^2)***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

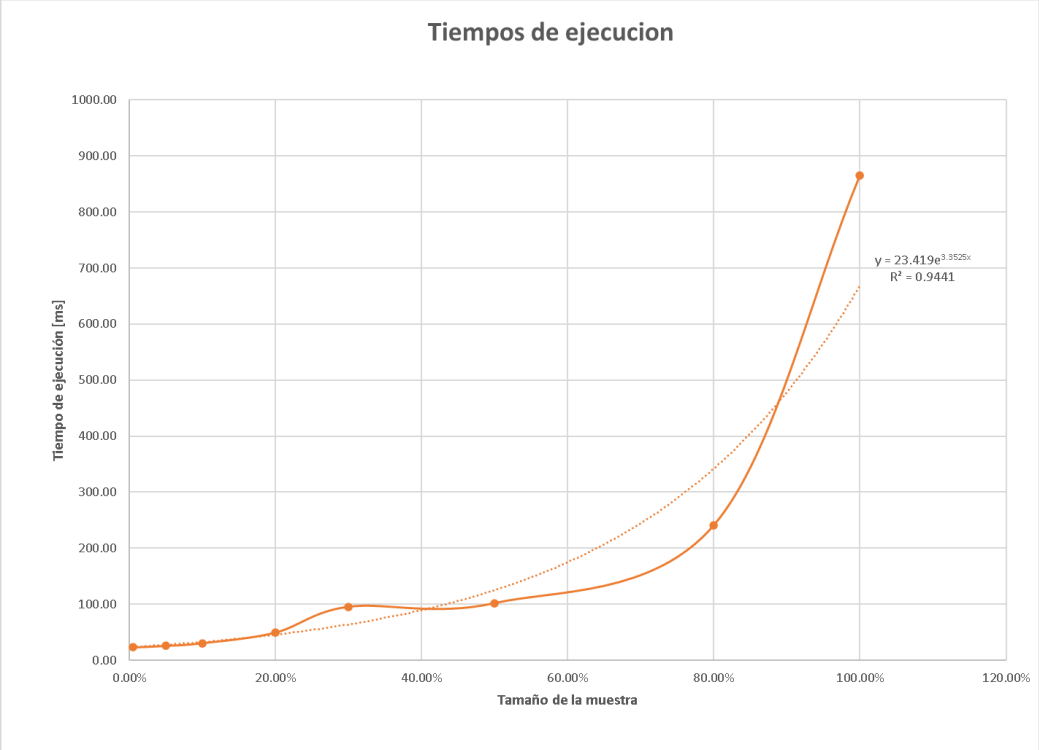
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	22.95
5.00%	23580.00	25.74
10.00%	47160.00	30.52
20.00%	117900.00	49.41
30.00%	235800.00	95.66
50.00%	353700.00	102.11
80.00%	542340.00	241.12
100.00%	778141.00	865.60

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.

La línea de tendencia muestra un crecimiento constante, donde cada vez la pendiente aumenta exponecialmente, puede relacionarse a la complejidad planteada: O(k^2), por lo tanto, es posible inferir que la complejidad que se calculó corresponde con la complejidad evidenciada en la gráfica.

Requerimiento 3

Descripción

```
def req_3(data_structs, clase, nombre_via):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    resultado = lt.newList("ARRAY_LIST")

    mapa_clase = mp.get(data_structs["Clase_accid"], clase)
    if not mapa_clase:
        return False
    clase = (me.getValue(mapa_clase))
    lt_clase = clase["Datos_clase"]
    for eleme in lt.iterator(lt_clase):
        if nombre_via in str(eleme["DIRECCION"]):
            lt.addLast(resultado, eleme)
    merg.sort(resultado, cmp_crimenes_by_reciente_antiguo)
    return resultado
```

Entrada	- El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. - La clase del accidente - El nombre de la via donde se quiere buscar
Salidas	Una lista de DISClib con los 3 accidentes de cierta clase ocurridos en una de las vías de la ciudad
Implementado (Sí/No)	Sí (Alejandro Pardo)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

K < n, debido a que no se recorren todos los datos sino solo los de la clase especifica.

Pasos	**Complejidad**
Creacion Array_list	O(1)
mp.get	O(1)
For	O(k)
Comparacion (todos)	O(k)
AddLast (todos)	O(1)
Merge sort (todos)	O(k*log(k))
TOTAL	***O(k*log(k))***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Tablas de datos

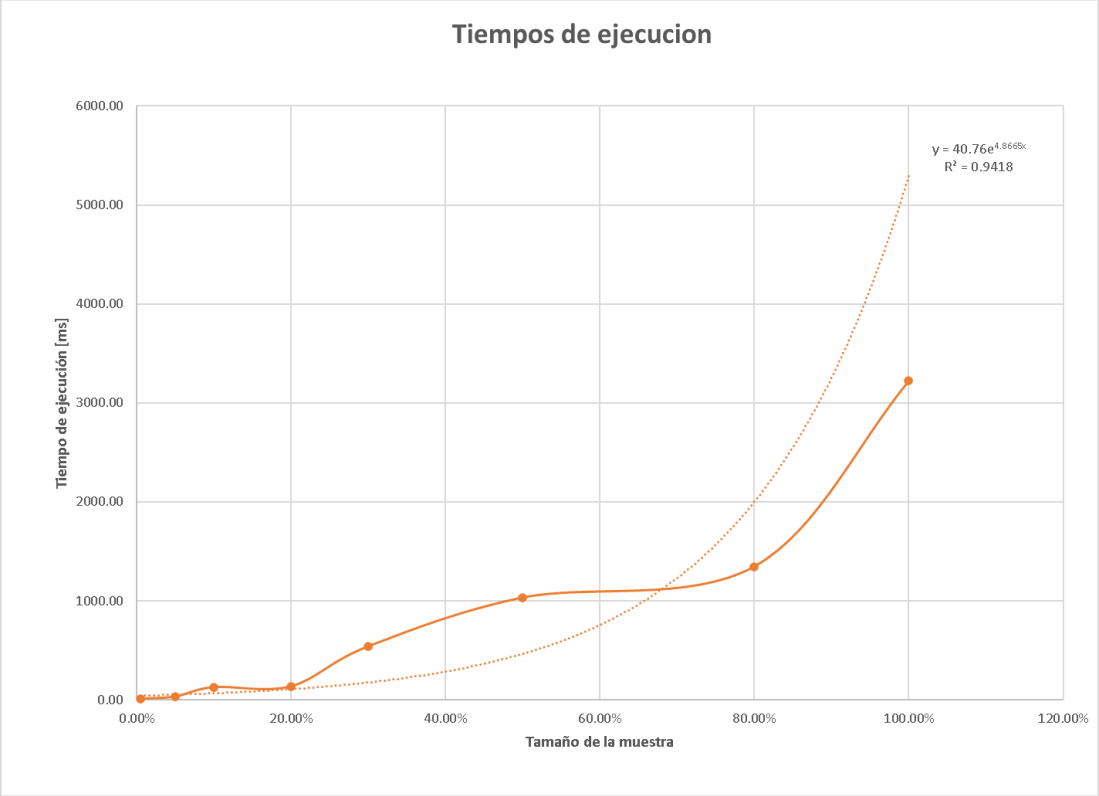
Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	10.32
5.00%	23580.00	31.48
10.00%	47160.00	126.69

20.00%	117900.00	135.20
30.00%	235800.00	538.11
50.00%	353700.00	1032.50
80.00%	542340.00	1346.27
100.00%	778141.00	3221.09

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.

La línea de tendencia muestra un crecimiento constante, donde cada vez la pendiente aumenta, puede relacionarse a la complejidad planteada: O(k*log(k)), por lo tanto, es posible inferir parcialmente que la complejidad que se calculó corresponde con la complejidad evidenciada en la gráfica.

Requerimiento 4

Descripción

```python
def req_4(data_structs, fecha_inicial, fecha_final, gravedad):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    resultado = lt.newList("ARRAY_LIST")

    om_fechas = data_structs["Fecha_occur"]
    for lts in lt.iterator(om.values(om_fechas, fecha_inicial, fecha_final)):
        datos_fecha = lts["Datos_fecha"]
        for dato in lt.iterator(datos_fecha):
            if dato["GRAVEDAD"] == gravedad:
                lt.addLast(resultado, dato)
    merg.sort(resultado, cmp_crimenes_by_reciente_antiguo)
    return resultado
```

Entrada	- El data_structs el cual contiene una llave llamada Anios que contiene todos los datos ordenados por año. - El año
Salidas	Una lista de DISClib con el subsector económico que tuvo los mayores costos y gastos de nómina para un año especifico (resultado). Las tres actividades económicas que menos aportaron y las tres actividades económicas que más aportaron al valor total de costos y gastos de nómina del subsector (filtro)
Implementado (Sí/No)	Sí (Joseph Pulido)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Para un k < n donde n es el número total de datos. Esto es porque no se recorren todos los datos, solamente el tramo dado por parámetro

Pasos	**Complejidad**
Creacion Array_list	O(1)
Om.values	O(k) k = cantidad de datos en el intervalo
For anidado	O(k^2)
Comparación	O(k)
addLast	O(1)
Merge sort	O(k*log(k))
TOTAL	***O(k^2)***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

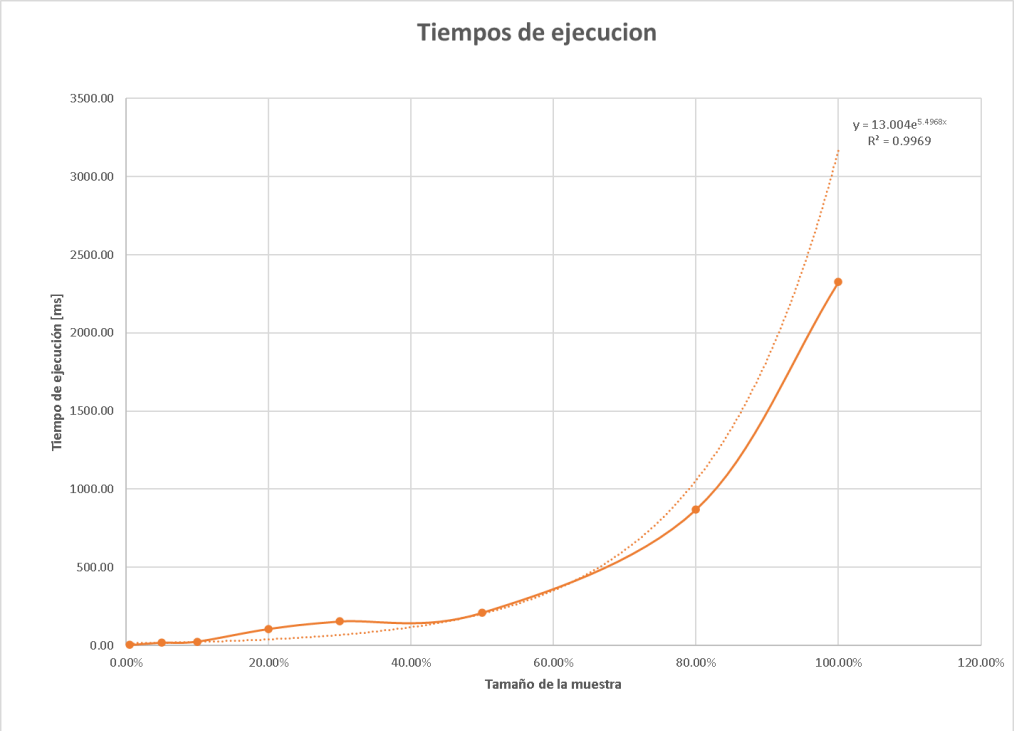
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	3.00
5.00%	23580.00	17.92
10.00%	47160.00	25.09
20.00%	117900.00	105.44
30.00%	235800.00	153.63
50.00%	353700.00	209.37
80.00%	542340.00	869.56
100.00%	778141.00	2326.31

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.

La línea de tendencia muestra un crecimiento constante, donde cada vez la pendiente aumenta exponencialmente, puede relacionarse a la complejidad planteada: O(k^2), por lo tanto, es posible inferir que la complejidad que se calculó corresponde con la complejidad evidenciada en la gráfica.

Requerimiento 5

Descripción

```python
def req_5(data_structs, year, mes, localidad):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    resultado = lt.newList("ARRAY_LIST")

    dic_anio = mp.get(data_structs["Anios"], year)
    if not dic_anio:
        return False
    dic_anio=me.getValue(dic_anio)
    mes = mp.get(dic_anio["Mes"], mes)
    if not mes:
        return False
    lt_mes = me.getValue(mes)
    lt_mes = (lt_mes["Datos_mes"])
    for eleme in lt.iterator(lt_mes):
        if str(eleme["LOCALIDAD"]) == localidad:
            lt.addLast(resultado, eleme)
    merg.sort(resultado, cmp_crimenes_by_reciente_antiguo)
    return resultado
```

Entrada	- El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. - El año - El mes - La localidad
Salidas	Una lista de DISClib con los 10 accidentes más recientes ocurridos en un mes y un año en una localidad de la ciudad.
Implementado (Sí/No)	Sí (Santiago González)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Para un k < n donde n es el número total de datos. Esto es porque no se recorren todos los datos, solamente el tramo dado por parámetro

Pasos	**Complejidad**
Creacion Array_list	O(1)
mp.get	O(1)
For	O(k)
Comparacion (todos)	O(k)
AddLast (todos)	O(1)
Merge sort (todos)	O(k*log(k))
TOTAL	***O(k*log(k))***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

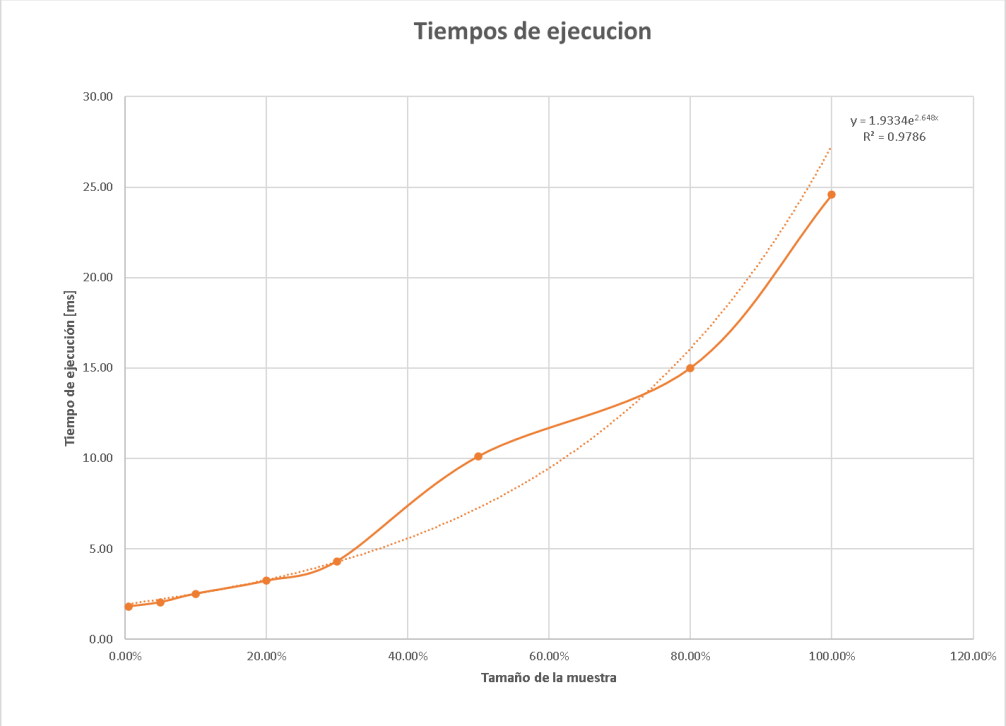
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	1.815
5.00%	23580.00	2.051
10.00%	47160.00	2.519
20.00%	117900.00	3.237
30.00%	235800.00	4.314
50.00%	353700.00	10.118
80.00%	542340.00	15.002
100.00%	778141.00	24.578

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.

La línea de tendencia muestra un crecimiento constante, donde cada vez la pendiente aumenta, puede relacionarse a la complejidad planteada: O(k*log(k)). La línea de tendencia puede ser más inclinada que un O(k), por lo tanto, es posible inferir parcialmente que la complejidad que se calculó corresponde con la complejidad evidenciada en la gráfica.

Requerimiento 6

Descripción

```python
#REQ 6
def radio_math (longitud_1, longitud_2, latitud_1, latitud_2):
    longitud_1, longitud_2, latitud_1, latitud_2 = map(mt.radians, [longitud_1, longitud_2, latitud_1, latitud_2])

    resta_longitud=(longitud_2-longitud_1)/2
    resta_latitudes=(latitud_2-latitud_1)/2
    cos_lat_1= mt.cos(latitud_1)
    cos_lat_2= mt.cos(latitud_2)
    raiz=mt.sqrt((mt.sin(resta_latitudes)**2)+(cos_lat_1*cos_lat_2*(mt.sin(resta_longitud)**2)))
    total=2*(mt.asin(raiz))*6371.009
    return total

def cmp_by_req6(crimen1, crimen2):
    if (crimen1["RADIO"]) < (crimen2["RADIO"]):
        return True
    else:
        return False

def req_6(data_structs, year, mes, longitud, latitud, radio, top):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    resultado = lt.newList("ARRAY_LIST")

    dic_anio = mp.get(data_structs["Anios"], year)
    if not dic_anio:
        return False
    dic_anio=me.getValue(dic_anio)
    mes = mp.get(dic_anio["Mes"], mes)
    if not mes:
        return False
    lt_mes = me.getValue(mes)
    lt_mes = (lt_mes["Datos_mes"])
    for eleme in lt.iterator(lt_mes):
        latitud2 = float(eleme["LATITUD"])
        longitud2 = float(eleme["LONGITUD"])
        radio_intervalo = radio_math(longitud, longitud2, latitud, latitud2)
        eleme["RADIO"] = radio_intervalo
        if radio_intervalo <= radio:
            lt.addLast(resultado, eleme)
    merg.sort(resultado, cmp_by_req6)
    if top < lt.size(resultado):
        lt_top = lt.subList(resultado,1,top)
    else:
        lt_top = resultado
    return lt_top
```

Entrada	El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. El año El mes La latitud La longitud El radio El top
Salidas	Una lista de DISClib con un número particular de accidentes que ocurrieron dentro de una zona circular específica de la ciudad para un mes y un año
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Para un k < n donde n es el número total de datos. Esto es porque no se recorren todos los datos, solamente el tramo dado por parámetro

Pasos	**Complejidad**
Creacion Array_list	O(1)
mp.get	O(1)
For	O(k)
Comparacion (todos)	O(k)
AddLast (todos)	O(1)
Merge sort (todos)	O(k*log(k))
TOTAL	***O(k*log(k))***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

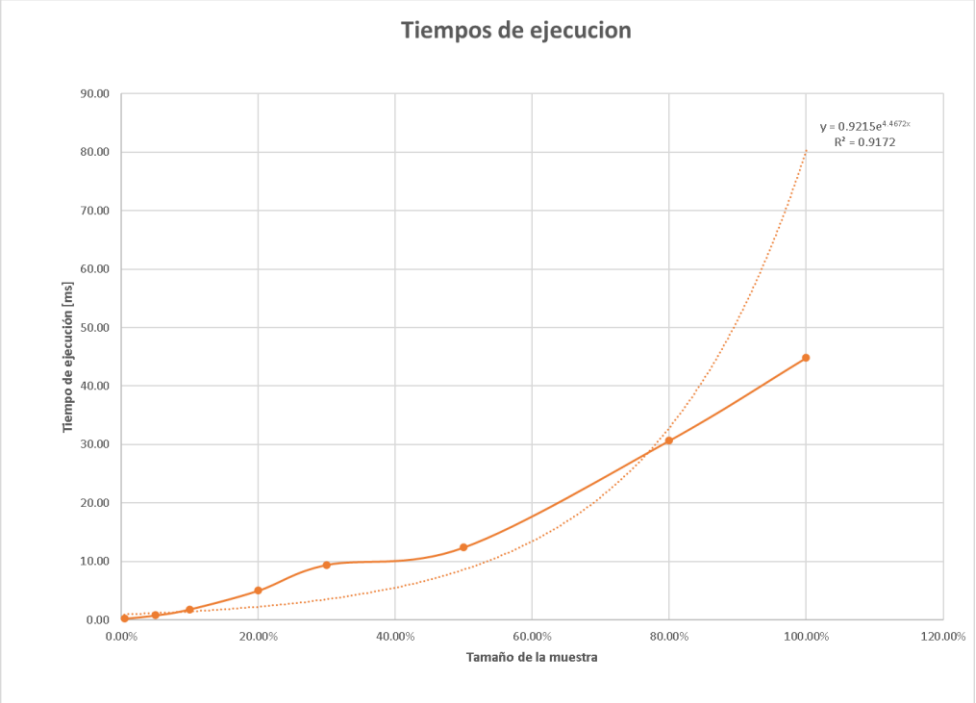
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	0.241
5.00%	23580.00	0.793
10.00%	47160.00	1.823
20.00%	117900.00	5.012
30.00%	235800.00	9.432
50.00%	353700.00	12.404
80.00%	542340.00	30.678
100.00%	778141.00	44.845

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.

Logramos apreciar que la línea de tendencia que se calcula a partir de la gráfica corresponde a la complejidad calculada y si la comparamos con las gráficas anteriores, observamos que son muy parecidas y esto tiene todo el sentido ya que en esta función también usamos un merge sort para ordenar los datos.

Requerimiento 7

Descripción

```python
def req_7(data_structs, year, mes, escala):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    resultado = lt.newList("ARRAY_LIST")

    dic_anio = mp.get(data_structs["Anios"], year)
    if not dic_anio:
        return False
    dic_anio=me.getValue(dic_anio)
    mes_mp = mp.get(dic_anio["Mes"], mes)
    if not mes_mp:
        return False
    lt_mes = me.getValue(mes_mp)
    om_fecha_dia = lt_mes["Dia"]
    unicos_dias = lt.newList("ARRAY_LIST")
    unico = 0
    for dias_dt in lt.iterator(om.keySet(om_fecha_dia)):
        if dias_dt != unico:
            unico = dias_dt
            lt.addLast(unicos_dias, unico)
    for dias_en_lista in lt.iterator(unicos_dias):
        dias = om.get(om_fecha_dia, dias_en_lista)
        dic_dias = me.getValue(dias)
        merg.sort(dic_dias, cmp_crimenes_by_reciente_antiguo)
        maximo = lt.firstElement(dic_dias)
        minimo = lt.lastElement(dic_dias)
        lt.addLast(resultado, minimo)
        lt.addLast(resultado, maximo)
    graf=[]
    for hora in range(0,24):
        cuantos=0
        for dato in lt.iterator(lt_mes["Datos_mes"]):
            dat= dato["HORA_OCURRENCIA_ACC"]
            dat= dat[0:2]
            dat= int(dat.replace(":",""))
            if dat == hora:
                cuantos+=1
        graf.append(cuantos)
    return resultado , graf, str(lt.size(lt_mes["Datos_mes"]))
```

Entrada	- El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. -Año -Mes -Escala del eje Y de la gráfica de barras
Salidas	El accidente más temprano y más tarde para cada día de un año y mes dados. Adicionalmente, generar el histograma con el número de accidentes por cada hora (entera) del día para los accidentes del mismo año y mes dados.
Implementado (Sí/No)	Sí

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Para un k<n siendo n el número total de datos. Esto es porque se tomó una cantidad de datos específicos, solo se accedió a los años y mes pasados por parámetro.

Pasos	**Complejidad**
Creacion Array_list	O(1)
Get	O(1)
For (todos)	O(k)
addLast	O(1)
Comparacion	O(k)
TOTAL	***O(k)***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

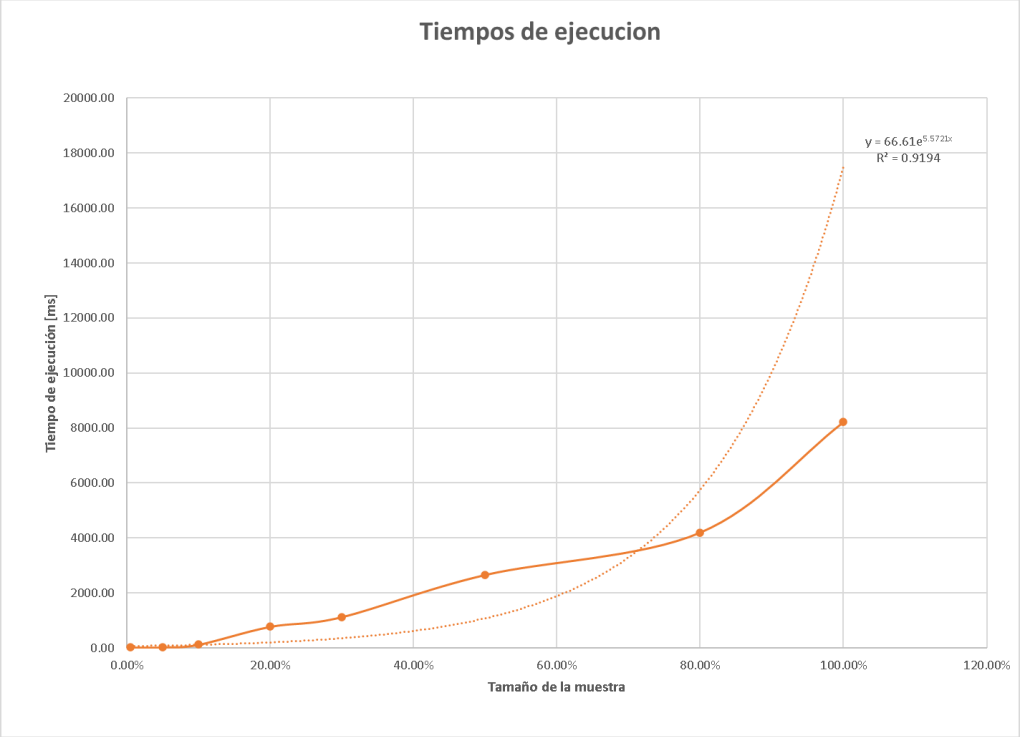
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	20.20
5.00%	23580.00	27.94
10.00%	47160.00	120.83
20.00%	117900.00	775.36
30.00%	235800.00	1127.51
50.00%	353700.00	2662.71
80.00%	542340.00	4204.13
100.00%	778141.00	8222.16

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

La complejidad calculada fue O(k), sin embargo; la gráfica tiende a un O(k*log(k)), esto puede ser debido a que se utilizó la librería matplotlib para realizar la grafica.

Requerimiento 8

Descripción

```
def req_8(data_structs, fecha1, fecha2, clase):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    resultado = lt.newList("ARRAY_LIST")

    om_fechas = data_structs["Fecha_occur"]
    lista_coo = []
    for fechas in lt.iterator(om.values(om_fechas, fecha1, fecha2)):
        for eleme in lt.iterator(fechas["Datos_Fecha"]):
            if eleme["CLASE_ACC"] == clase:
                lt.addLast(resultado, eleme)
                lista_coo.append(eleme["LATITUD"])
                lista_coo.append(eleme["LONGITUD"])
    if lt.size(resultado) <= 0 :
        return False
    m = folium.Map(location=lista_coo[0:2], zoom_start=16)
    marker_cluster = MarkerCluster(disableClusteringAtZoom=24).add_to(m)
    i = 2
    for eleme in lt.iterator(resultado):
        html="<p> Codigo de accidente: "+eleme["CODIGO_ACCIDENTE"]+"</p><p> Fecha de accidente: "+ eleme["FECHA_OCURRENCIA_ACC"]+"</p
        iframe=branca.element.IFrame(html=html, width=500, height=300)
        if eleme["GRAVEDAD"] == "SOLO DANOS":
            folium.Marker(location=lista_coo[i-2:i],icon=folium.Icon(color="green"),popup=folium.Popup(iframe, max_width=500)).add_to
            folium.Marker(location=lista_coo[i-2:i],icon=folium.Icon(color="green"),popup=folium.Popup(iframe, max_width=500)).add_to
        elif eleme["GRAVEDAD"] == "CON HERIDOS":
            folium.Marker(location=lista_coo[i-2:i],icon=folium.Icon(color="blue"),popup=folium.Popup(iframe, max_width=500)).add_to(
            folium.Marker(location=lista_coo[i-2:i],icon=folium.Icon(color="blue"),popup=folium.Popup(iframe, max_width=500)).add_to(
        else:
            folium.Marker(location=lista_coo[i-2:i],icon=folium.Icon(color="red"),popup=folium.Popup(iframe, max_width=500)).add_to(m
            folium.Marker(location=lista_coo[i-2:i],icon=folium.Icon(color="red"),popup=folium.Popup(iframe, max_width=500)).add_to(m
        i += 2
        if i > lt.size(resultado):
            break
    m.save('mapa.html') # para abrir y visualizar el mapa, debes abrir el archivo mapa.html que se encuentra en la carpeta del reto
    #webbrowser.open_new_tab('mapa.html')
    return lt.size(resultado)
```

Entrada	El data_structs, que contiene 5 llaves: DATOS_TODOS, que tiene un ADT list con todos los datos; Fecha_occur, que tiene un mapa con las fechas como llaves; Hora_occur, que tiene un mapa con las horas como llaves; Anios, que contiene un mapa con los años como llaves y Clase_accid que es también un mapa con llaves la clase de accidente. Fecha inicial Fecha final Clase del accidente
Salidas	Retorna la cantidad de datos en el intervalo de fechas. Además, guarda un archivo HTML llamado “mapa” para visualizar gráficamente en un mapa los puntos en los que ocurrieron accidentes por tipo para un rango de fechas especificado
Implementado (Sí/No)	Sí, para abrir y visualizar el mapa, debes abrir el archivo mapa.html que se encuentra en la carpeta del reto

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Para un k<n siendo n el número total de datos. Esto es porque se tomó una cantidad de datos específicos.

Pasos	**Complejidad**
Creacion Array_list	O(1)

For 1	O(k^2)
.append	O(1)
For 2	O(k)
AddLast	O(1)
TOTAL	***O(k^2)***

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).

Procesadores	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

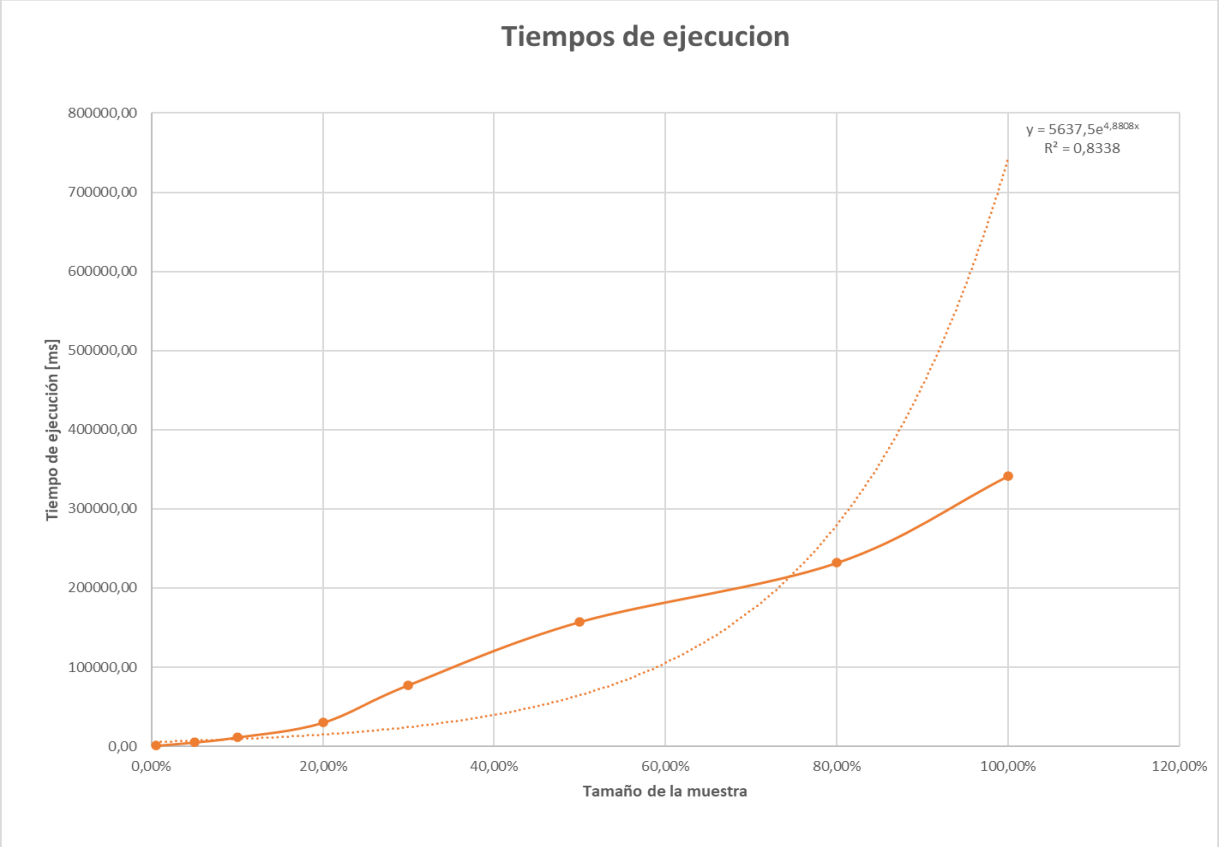
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Porcentaje de la muestra [pct]	Tamaño de la muestra	Merge Sort [ms]
0.50%	2358.00	1066,70
5.00%	23580.00	5259,92
10.00%	47160.00	11544,59
20.00%	117900.00	30015,06
30.00%	235800.00	77491,06
50.00%	353700.00	157262,81
80.00%	542340.00	231717,85
100.00%	778141.00	341104,56

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

Este requerimiento implementaba una librería para poder graficar el mapa llamado folium lo que explica los tiempos tan excesivos para cargar el requerimiento, lo que se hizo en el para solucionar el reto fue recorrer cierta cantidad de elemento, por eso en la complejidad aparece k y no n, para así poder comparar y agregar a una lista los resultados para luego imprimir. Se uso dos fors debido a que cargamos los datos en mapas que contienen listas o mapas, debido a estos fors se estableció la complejidad O(k^2) y tiene todo el sentido según la gráfica obtenida ya que la línea de tendencias se acopla muy bien a esta complejidad.