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

ANÁLISIS DEL RETO

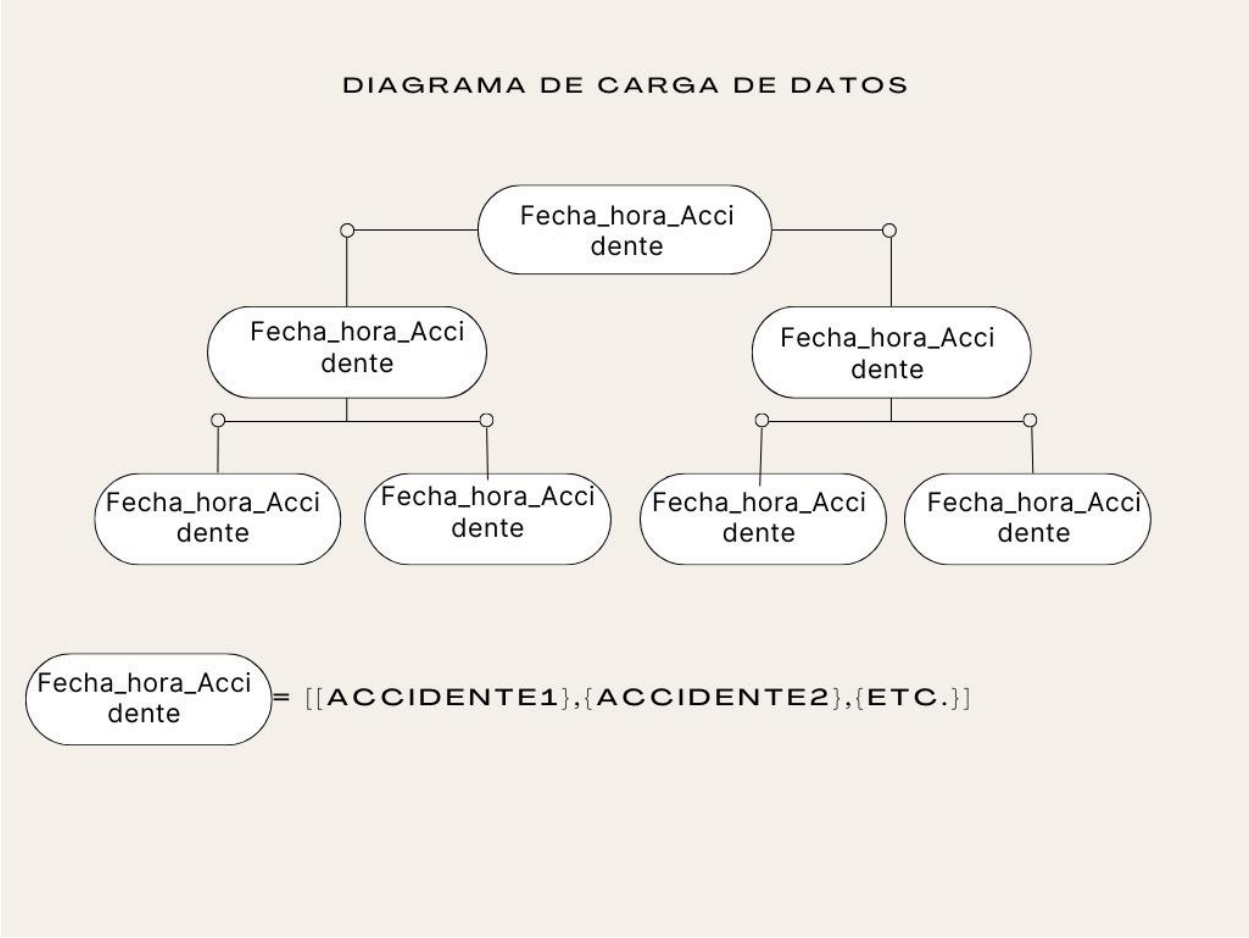
Estudiante 1 Andrés Cáceres Cod 202214863, a.caceresg@uniandes.edu.co

Estudiante 2 Juan Jose Diaz Cod 202220657, jj.diazo1@uniandes.edu.co

Estudiante 3 Ángel Farfán Cod 202222183, a.farfana@uniandes.edu.co

Requerimiento <<0>> Carga de datos

Plantilla para documentar y analizar cada uno de los requerimientos.



Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	archivo excel con datos
Salidas	árbol con todos los datos
Implementado (Sí/No)	Sí, Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Añadir a la lista y al árbol	O(1),O(Logn)
TOTAL	***O(n)***
Pasos	**Complejidad**

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)**
small	0.490
5pct	1.631
10pct	**2.24**
20pct	**5.89**
30pct	**12.10**
50pct	**14.35**
80pct	**22.30**
large	28.62

Tablas de datos

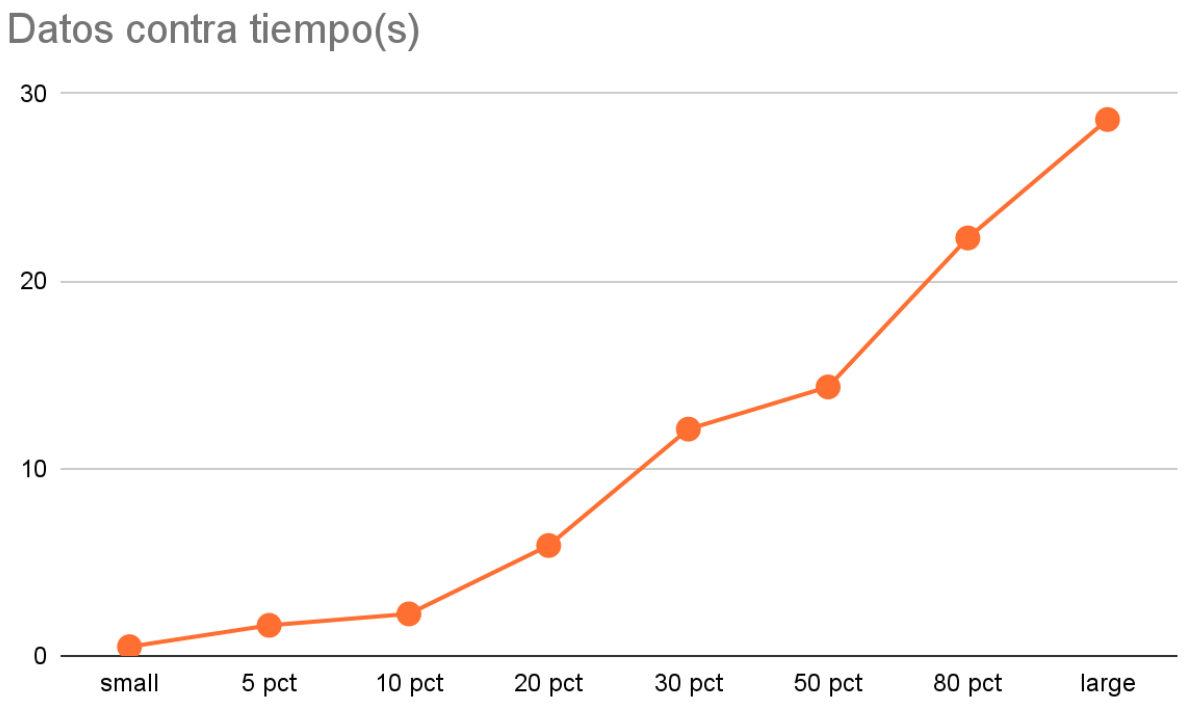
Las tablas con la recopilación de datos de las pruebas.

Entrada	**Salida**	**Tiempo (s)**
small	Dato1	0.490
5pct	Dato2	1.631
10pct	**Dato3**	**2.24**
20pct	**Dato4**	**5.89**
30pct	**Dato5**	**12.10**
50pct	**Dato6**	**14.35**

80pct	**Dato7**	**22.30**
large	Dato8	28.62

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.

Carga de datos

Descripción

```
def add_data(data_structs, data):
    """
    Función para agregar nuevos elementos a la lista
    """
    #TODO: Crear la función para agregar elementos a una lista
    date_acc=data['FECHA_HORA_ACC'][:-3]
    date_time = datetime.strptime(date_acc, '%Y/%m/%d %H:%M:%S')
    isPresent=om.get(data_structs['data'],date_time)

    if isPresent==None:
        lista=lt.newList(datastructure='ARRAY_LIST')
        lt.addLast(lista,data)
        om.put(data_structs['data'],date_time,lista)
    else:
        acc_content=isPresent['value']
        lt.addLast(acc_content,data)
        om.put(data_structs['data'],date_time,acc_content)

def add_data_acc(data_structs,data):
    """
    Función para agregar nuevos elementos a la lista
    """
    #TODO: Crear la función para agregar elementos a una lista
    key=data['CLASE_ACC']
    date_time = datetime.strptime(data['FECHA_HORA_ACC'][:-3], '%Y/%m/%d %H:%M:%S')

    if mp.contains(data_structs['data_acc'],key):
        isPresent=om.get(mp.get(data_structs['data_acc'],key)['value'],date_time)

        if isPresent==None:
            lista=lt.newList(datastructure='ARRAY_LIST')
            lt.addLast(lista,data)
            om.put((mp.get(data_structs['data_acc'],key)['value']),date_time,lista)
        else:
            acc_content=isPresent['value']
            lt.addLast(acc_content,data)
            om.put(mp.get(data_structs['data_acc'],key)['value'],date_time,acc_content)

    else:
        mini_map=om.newMap(omaptype='RBT')
        lista=lt.newList(datastructure='ARRAY_LIST')
        lt.addLast(lista,data)
        om.put(mini_map,date_time,lista)
        mp.put(data_structs['data_acc'],key,mini_map)
```

Este requerimiento se encarga de cargar los datos del excel dentro de un árbol de tipo rbt donde la llave de cada dato corresponde a su valor de FECHA_HORA_ACC y se guarda con el tipo datetime después de utilizar la función datetime.strptime().

Entrada	Estructuras de datos del modelo, ID.
Salidas	un árbol con todos los datos organizados según su ID.
Implementado (Sí/No)	Si. Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Buscar si el elemento existe (isPresent)	O(n)
Añadir a la lista y al árbol	O(1),O(Logn)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (s)
small	0.490
5 pct	1.631
10 pct	**2.24**
20 pct	**5.89**
30 pct	**12.10**
50 pct	**14.35**
80 pct	**22.30**
large	28.62

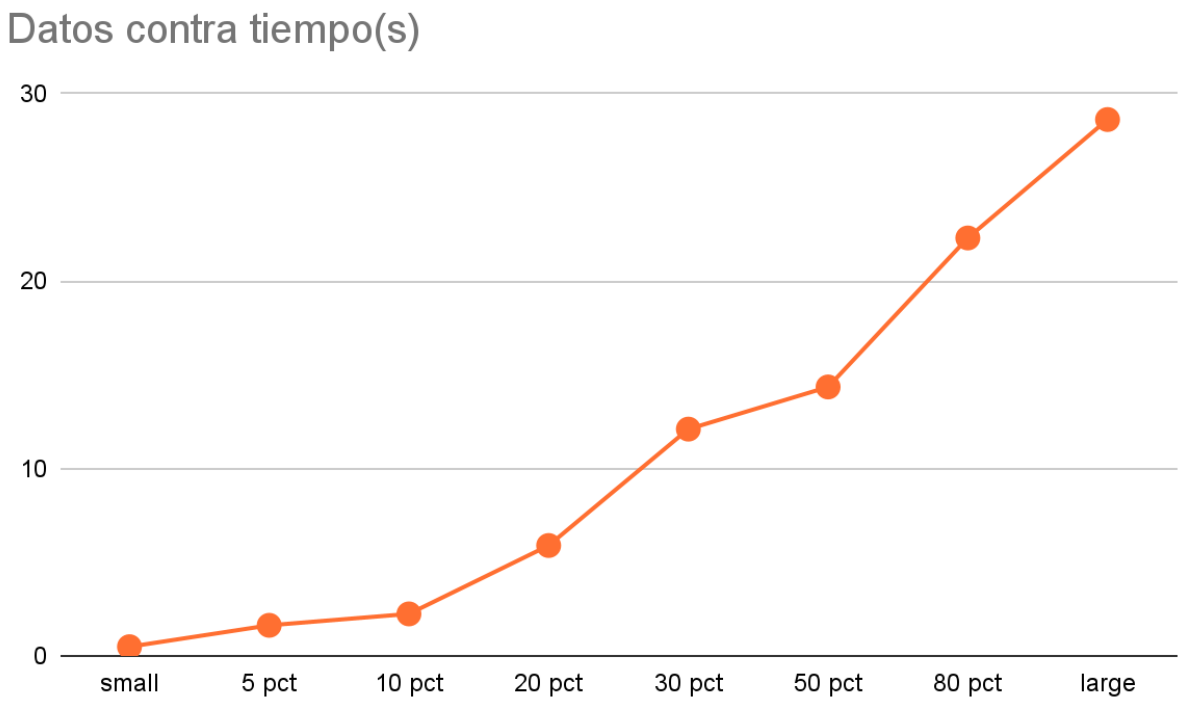
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (s)
small	Dato1	0.490
5pct	Dato2	1.631
10pct	**Dato3**	**2.24**
20pct	**Dato4**	**5.89**
30pct	**Dato5**	**12.10**
50pct	**Dato6**	**14.35**
80pct	**Dato7**	**22.30**
large	Dato8	28.62

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de cargar los datos del archivo csv dentro de un árbol de tipo rbt donde la llave de cada dato corresponde a su valor de FECHA_HORA_ACC y se guarda con el tipo datetime después de utilizar la función datetime.strptime().

Este requerimiento es un poco más demorado debido a que utiliza listas de tipo array dentro de cada nodo de un árbol, además de que la cantidad de datos de este reto es mucho más grande que la del reto anterior, y corresponde a 235000 datos que se organizan, guardan en listas e ingresan a nodos de un árbol.

Requerimiento 1

Descripción

```python
def req_1(data_structs,date_init,date_fin):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    data=om.values(data_structs['data'],datetime.strptime(str(date_init+' 00:00:00'), '%Y/%m/%d %H:%M:%S'),datetime.strptime(str(date_fin+' 23:59:59'), '%Y/%m/%d %H:%M:%S'))
    data_set_org=lt.newList(datastructure='ARRAY_LIST')
    for j in lt.iterator(data):
        if j['size']!=1:
            for k in lt.iterator(j):
                lt.addLast(data_set_org,k)
        else:
            lt.addLast(data_set_org,lt.firstElement(j))
    return data_set_org['elements']
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array.

Entrada	Estructuras de datos del modelo, fecha inicial, fecha final
Salidas	# de accidentes en rango de fechas, accidentes ordenados de más reciente a menos reciente
Implementado (Sí/No)	Si. Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
separar por fechas	O(n)
iterar la lista	O(n)
iterar la segunda lista	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	0.100
5 pct	0.213
10 pct	**1.12**
20 pct	**1.40**
30 pct	**2.90**
50 pct	**3.97**
80 pct	**4.07**
large	6.842

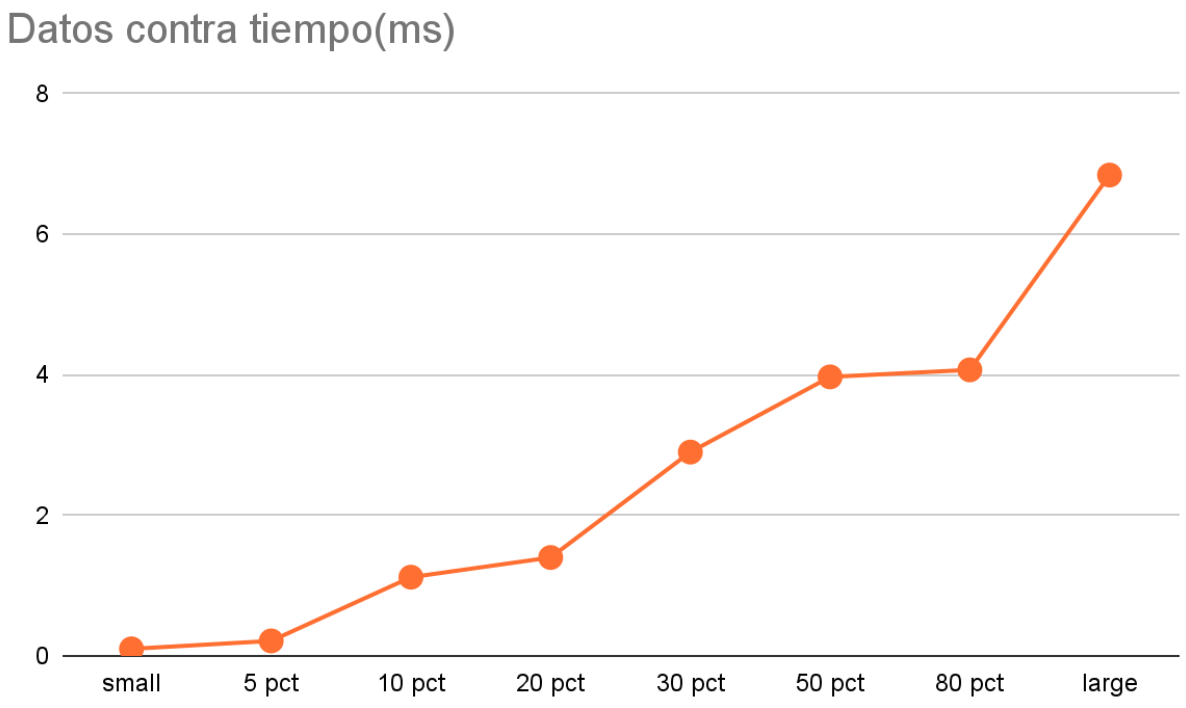
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	0.100
5pct	Dato2	0.213
10pct	**Dato3**	**1.12**
20pct	**Dato4**	**1.40**
30pct	**Dato5**	**2.90**
50pct	**Dato6**	**3.97**
80pct	**Dato7**	**4.07**
large	Dato8	6.842

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array.

Retorna los datos dentro de la lista ordenados según su fecha de ocurrencia junto con la información del código, la localidad, la dirección, gravedad, clase, latitud y longitud del accidente para que pueda ser identificado fácilmente. Además el requerimiento es bastante sencillo y eficiente en cuanto a código, por lo que es evidente que los tiempos de carga fueron extremadamente bajos incluso para los datos large.

Requerimiento 2

Descripción

```
def req_2(data_structs,init_time,fin_time,year,month):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2

    calendar.monthrange(int(year),int(month))[1]
    data=om.values(data_structs['data'],datetime.strptime(str(year+'/'+month+'/'+'01'+' '+ init_time+':00'),"%Y/%m/%d %H:%M:%S"),datetime.strptime(str(year+'/'+month+'/'+str(calendar.monthrange(int(year),int(month))[1])+' '+ init_time+':59'),"%Y/%m/%d %H:%M:%S") )
    lista_final=lt.newList(datastructure='ARRAY_LIST')
    init_time,fin_time=datetime.strptime(init_time+':00',"%H:%M:%S"),datetime.strptime(fin_time+':00',"%H:%M:%S")
    for j in lt.iterator(data):
        if j['size']!=1:
            for k in lt.iterator(j):
                acc_time=datetime.strptime(k['HORA_OCURRENCIA_ACC'],"%H:%M:%S")
                if init_time<=acc_time<=fin_time:
                    k['HORA_OCURRENCIA_ACC_CH']=acc_time
                    lt.addLast(lista_final,k)
        else:
            acc_time=datetime.strptime(j['elements'][0]['HORA_OCURRENCIA_ACC'],"%H:%M:%S")
            if init_time<=acc_time<=fin_time:
                j['elements'][0]['HORA_OCURRENCIA_ACC_CH']=acc_time
                lt.addLast(lista_final,j['elements'][0])
    return quk.sort(lista_final,cmp_req2)['elements']
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de año, mes, hora inicial y hora final.

Entrada	Hora y minutos iniciales del intervalo de tiempo. Hora y minutos finales del intervalo de tiempo. Año de consulta. Mes de consulta
Salidas	El número total de accidentes ocurridos en el intervalo de tiempo, año y mes dados. Todos los accidentes ocurridos, ordenados de menos reciente a más reciente por su hora y minutos de ocurrencia.
Implementado (Sí/No)	Si. Andrés Cáceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
separar por fechas	O(n)
iterar la lista	O(n)
iterar otra vez	O(n)
quicksort	O(nlogn)
TOTAL	***O(nlogn)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	AMD Ryzen 7 5000HS with Radeon Graphics
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	0.120
5 pct	0.221
10 pct	**1.127**
20 pct	**1.413**
30 pct	**2.910**
50 pct	**3.973**
80 pct	**4.076**
large	6.863

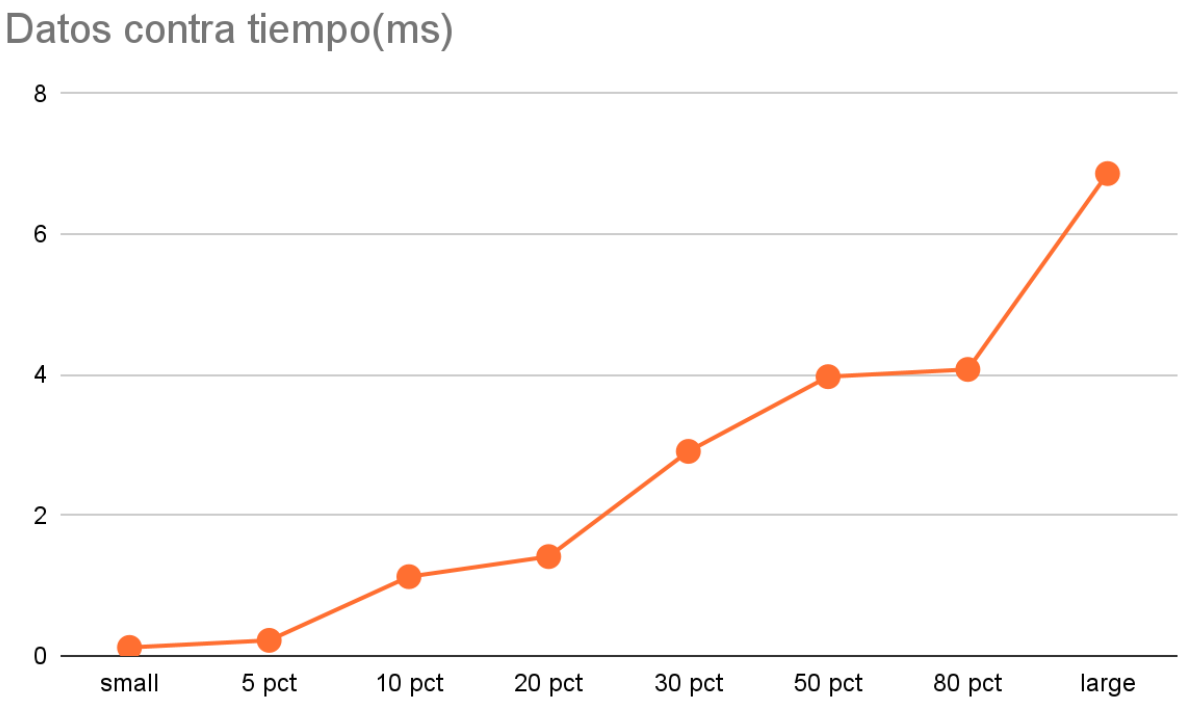
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	0.120
5pct	Dato2	0.221
10pct	**Dato3**	**1.127**
20pct	**Dato4**	**1.413**
30pct	**Dato5**	**2.910**
50pct	**Dato6**	**3.973**
80pct	**Dato7**	**4.076**
large	Dato8	6.863

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de año, mes, hora inicial y hora final.

Retorna los datos de los accidentes dentro de una lista de tipo array ordenados según el año que entra por parámetro, el mes que entra por parámetro, una hora inicial dada y una hora final dada. Estos parámetros permiten seleccionar cuáles son los datos que hay que retornar y para esto se realiza una comparación de su hora de ocurrencia con la de los parámetros, si el accidente está dentro de estos rangos entonces se ingresa a la lista para luego ordenarla de más tarde a más temprano. Además también es de notar la velocidad y eficiencia del requerimiento.

Requerimiento 3

Descripción

```
def req_3(data_structs,acc,name_city):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    lista=None
    lista_final=lt.newList(datastructure='ARRAY_LIST')
    for i in lt.iterator(data_structs['data_acc']['table']):
        if i['key']==acc:
            lista=om.values(i['value'],datetime.strptime(str('2015/01/01 00:00:00'), '%Y/%m/%d %H:%M:%S'),datetime.strptime(str('2023/12/31 23:59:59'), '%Y/%m/%d %H:%M:%S'))
    for j in lt.iterator(lista):
        if j['size']!=1:
            for k in lt.iterator(j):
                if name_city in k['DIRECCION']:
                    lt.addLast(lista_final,k)
        else:
            if name_city in j['elements'][0]['DIRECCION']:
                lt.addLast(lista_final,j['elements'][0])
    return lt.subList(lista_final,lista_final['size']-2,3)['elements']
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de clase de accidente y nombre de la vía de la ciudad.

Entrada	clase de accidente y nombre de la vía de la ciudad.
Salidas	El número total de accidentes ocurridos en la vía en cuestión que correspondan a la clase. Los 3 accidentes más recientes ocurridos en la vía en cuestión que correspondan a la clase.
Implementado (Sí/No)	Si. Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
iterar la lista	O(n)
separar por fechas	O(n)
iterar otra vez	O(n)
tercera iteración	o(n)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	2.21
5 pct	9.4
10 pct	**12.9**
20 pct	**18.132**
30 pct	**19.47**
50 pct	**29.10**
80 pct	**42.01**
large	47.23

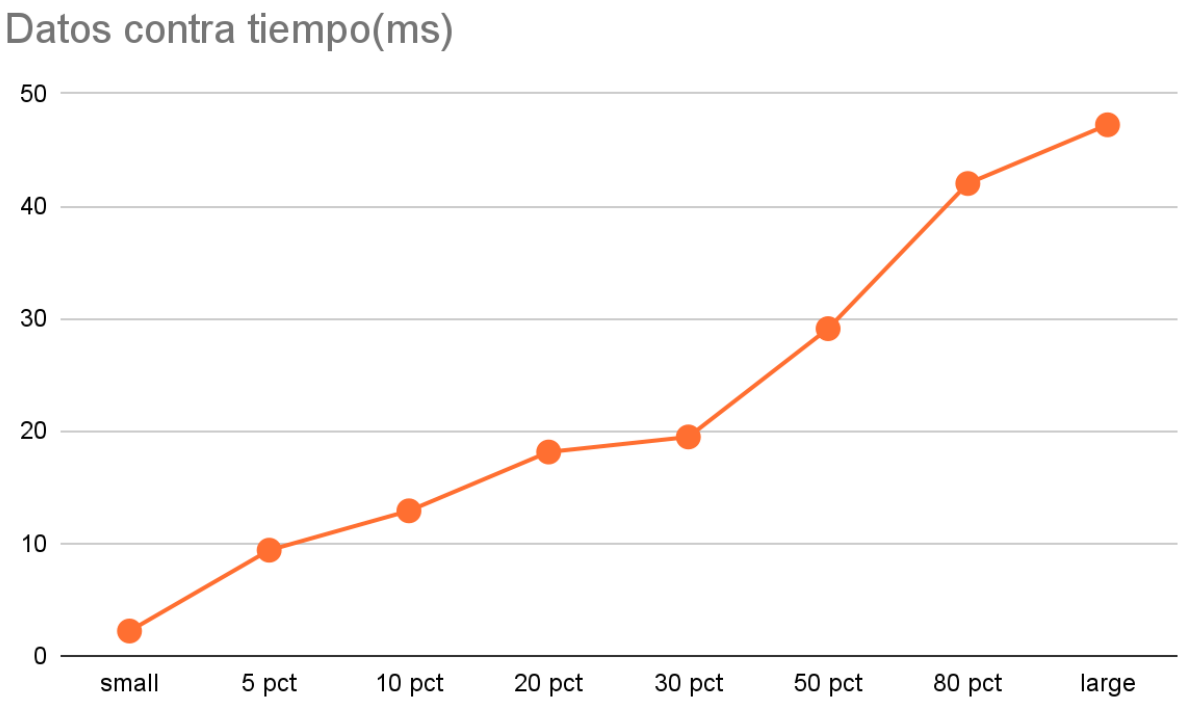
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	2.21
5pct	Dato2	9.4
10pct	**Dato3**	**12.9**
20pct	**Dato4**	**18.132**
30pct	**Dato5**	**19.47**
50pct	**Dato6**	**29.10**
80pct	**Dato7**	**42.01**
large	Dato8	47.23

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de clase de accidente y nombre de la vía de la ciudad.

Este requerimiento retorna los 3 accidentes más recientes de una clase ocurridos en una vía en cuestión. La clase y la vía entran por parámetro, con esta información se organiza una lista con una sublista,en la que se ingresan los datos que cumplan con las comparaciones de que sus valores de clase y vía correspondan a los mismos que entran por parámetro. Aunque este requerimiento pueda ser un poco más demorado que los demás sigue manteniéndose bastante rápido.

Requerimiento 4

Descripción

```
def req_4(data_structs,date_init,date_fin,grav_acc):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4

    data=om.values(data_structs['data'],datetime.strptime(str(date_init+' 00:00:00'), '%Y/%m/%d %H:%M:%S'),datetime.strptime(str(date_fin+' 23:59:59'), '%Y/%m/%d %H:%M:%S'))
    lista_final=lt.newList(datastructure='ARRAY_LIST')

    for j in lt.iterator(data):
        if j['size']!=1:
            for k in lt.iterator(j):
                if grav_acc==k['GRAVEDAD']:
                    lt.addLast(lista_final,k)
        else:
            if grav_acc==j['elements'][0]['GRAVEDAD']:
                lt.addLast(lista_final,j['elements'][0])
    return lt.subList(lista_final,lista_final['size']-4,5)['elements']
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de fecha inicial del intervalo,fecha final del intervalo, gravedad del accidente.

Entrada	Fecha inicial del intervalo. Fecha final del intervalo. Gravedad del accidente.
Salidas	El número de accidentes para el intervalo de fechas especificado. Los 5 registros de accidentes de la gravedad dada más antiguos
Implementado (Sí/No)	Si. Ángel Farfán/Juan José Díaz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
iterar la lista	O(n)
separar por fechas	O(n)
iterar otra vez	O(n)
tercera iteración	o(n)
quicksort	O(nlogn)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	0.10
5 pct	0.215
10 pct	**1.114**
20 pct	**1.418**
30 pct	**2.921**
50 pct	**3.969**
80 pct	**4.061**
large	6.859

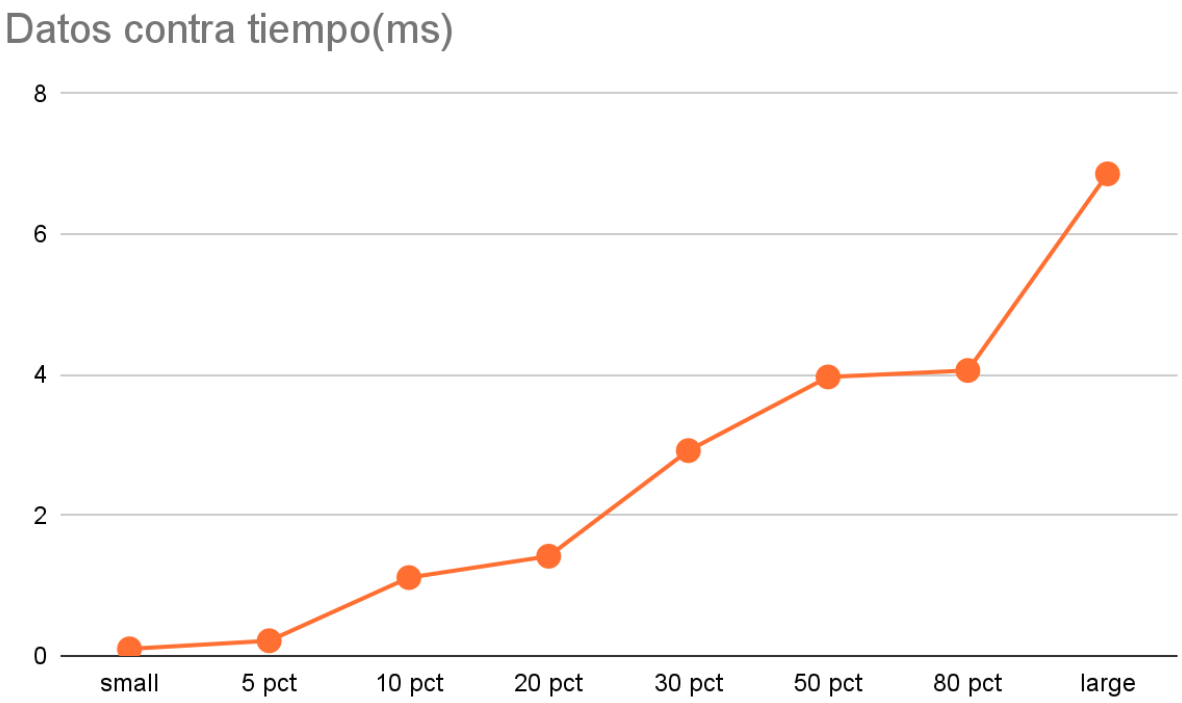
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	0.10
5pct	Dato2	0.215
10pct	**Dato3**	**1.114**
20pct	**Dato4**	**1.418**
30pct	**Dato5**	**2.921**
50pct	**Dato6**	**3.969**
80pct	**Dato7**	**4.061**
large	Dato8	6.859

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de fecha inicial del intervalo,fecha final del intervalo, gravedad del accidente.

Este requerimiento retorna los 5 accidentes más antiguos de una gravedad dada por parámetro. Para cada parámetro se retorna sus datos de tiempo como fecha,hora y día, además sus datos de ubicación como latitud,longitud,dirección y localidad, finalmente el código y la clase de accidente. La lista retornada se encuentra ordenada en orden descendente en cuanto a tiempo, es decir primero los accidentes más recientes finalmente los más antiguos dentro del top 5 requerido.

Requerimiento 5

Descripción

```
def req_5(data_structs,zone,month,year):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    data=om.values(data_structs['data'],datetime.strptime(str(year+'/'+month+'/'+'01'+' '+'00:00:00'),"%Y/%m/%d %H:%M:%S"),datetime.strptime(str(year+'/'+month+'/'+str(calendar.monthrange(int(year),int(month))[1])+' '+' '+'23:59:59'),"%Y/%m/%d %H:%M:%S") )
    lista_final=lt.newList(datastructure='ARRAY_LIST')

    for j in lt.iterator(data):
        if j['size']!=1:
            for k in lt.iterator(j):
                if zone==k['LOCALIDAD']:
                    lt.addLast(lista_final,k)
        else:
            if zone==j['elements'][0]['LOCALIDAD']:
                lt.addLast(lista_final,j['elements'][0])
    return lt.subList(lista_final,lista_final['size']-9,10)['elements']
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de localidad de Bogotá, un mes, un año.

Entrada	Localidad de Bogotá en la que ocurrieron accidentes. Un mes. Año entre 2015 y 2022.
Salidas	El número total de accidentes ocurridos en la localidad durante la fecha indicada. Los 10 registros de accidentes más recientes en la localidad
Implementado (Sí/No)	Si. Andrés Cáceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
separar por fechas	O(n)
iterar la lista	O(n)
iterar otra vez	O(n)
tercera iteración	o(n)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	0.157
5 pct	0.213
10 pct	**1.124**
20 pct	**1.442**
30 pct	**2.965**
50 pct	**3.995**
80 pct	**4.018**
large	6.881

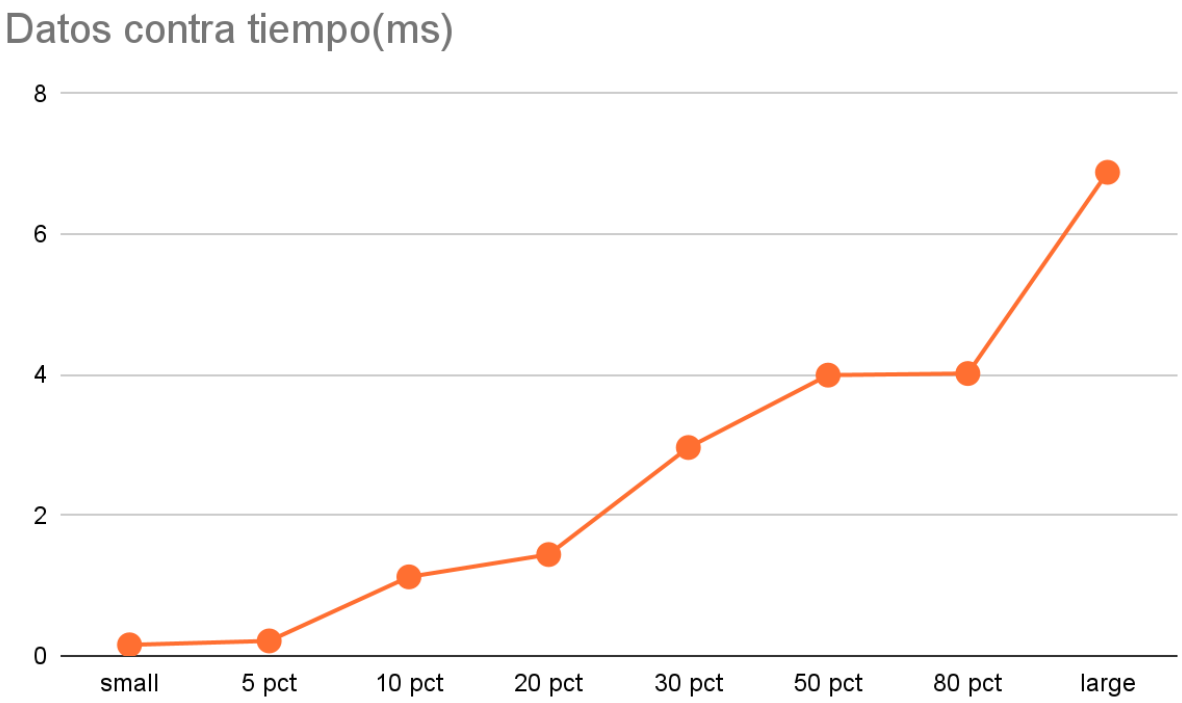
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	0.157
5pct	Dato2	0.213
10pct	**Dato3**	**1.124**
20pct	**Dato4**	**1.442**
30pct	**Dato5**	**2.965**
50pct	**Dato6**	**3.995**
80pct	**Dato7**	**4.018**
large	Dato8	6.881

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de localidad de Bogotá, un mes, un año.

A manera de retorno podemos ver que al igual que en el req 4 se utilizó una lista con una sublista y sigue el mismo esqueleto que el requerimiento anterior, pero en este caso se comparan los datos que entran por parámetro en sus valores de localidad, mes y año. Finalmente después de haber pasado todos los datos por esta comparación se añaden a las listas.

Requerimiento 6

Descripción

```
def req_6(data_structs,month,year,latitude,lenght,ratio,num_accidents):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    data=om.values(data_structs['data'],datetime.strptime(str(year+'/'+month+'/'+'01'+' '+'00:00:00'),"%Y/%m/%d %H:%M:%S"),datetime.strptime(str(year+'/'+month+'/'+str(calendar.monthrange(int(year),int(month))[1])+' '+' '+'23:59:59'),"%Y/%m/%d %H:%M:%S") )

    lista_final=lt.newList(datastructure='ARRAY_LIST')
    for j in lt.iterator(data):
        if j['size']!=1:
            for k in lt.iterator(j):
                k['DISTANCIA_PARAMETRO_PUNTO']=haversine_equation(lenght,latitude,k['LONGITUD'],k['LATITUD'])

                if float(k['DISTANCIA_PARAMETRO_PUNTO'])<=float(ratio):
                    lt.addLast(lista_final,k)
        else:
            j['elements'][0]['DISTANCIA_PARAMETRO_PUNTO']=haversine_equation(lenght,latitude,j['elements'][0]['LONGITUD'],j['elements'][0]['LATITUD'])

            if float(j['elements'][0]['DISTANCIA_PARAMETRO_PUNTO'])<=float(ratio):
                lt.addLast(lista_final,j['elements'][0])

    return quk.sort(lista_final,cmp_req6)['elements'][:int(num_accidents)]
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de mes, año, coordenadas del centro del área, radio en km, número de accidentes.

Entrada	Mes. Año entre 2015 y 2022 . Coordenadas del centro del área (Latitud y Longitud) . Radio del área en km . Número de accidentes
Salidas	Los N accidentes ocurridos, organizados por su cercanía al centro de la zona. Los registros de todos los N accidentes ocurridos en la zona.
Implementado (Sí/No)	Si. Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
separar por fechas	O(n)
haversine equation	O(n)
filtrar accidentes según radio	O(n)
quicksort según distancia	o(nlogn)
TOTAL	***O(nlogn)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	2.13
5 pct	9.14
10 pct	**12.924**
20 pct	**18.181**
30 pct	**19.483**
50 pct	**29.167**
80 pct	**42.101**
large	47.240

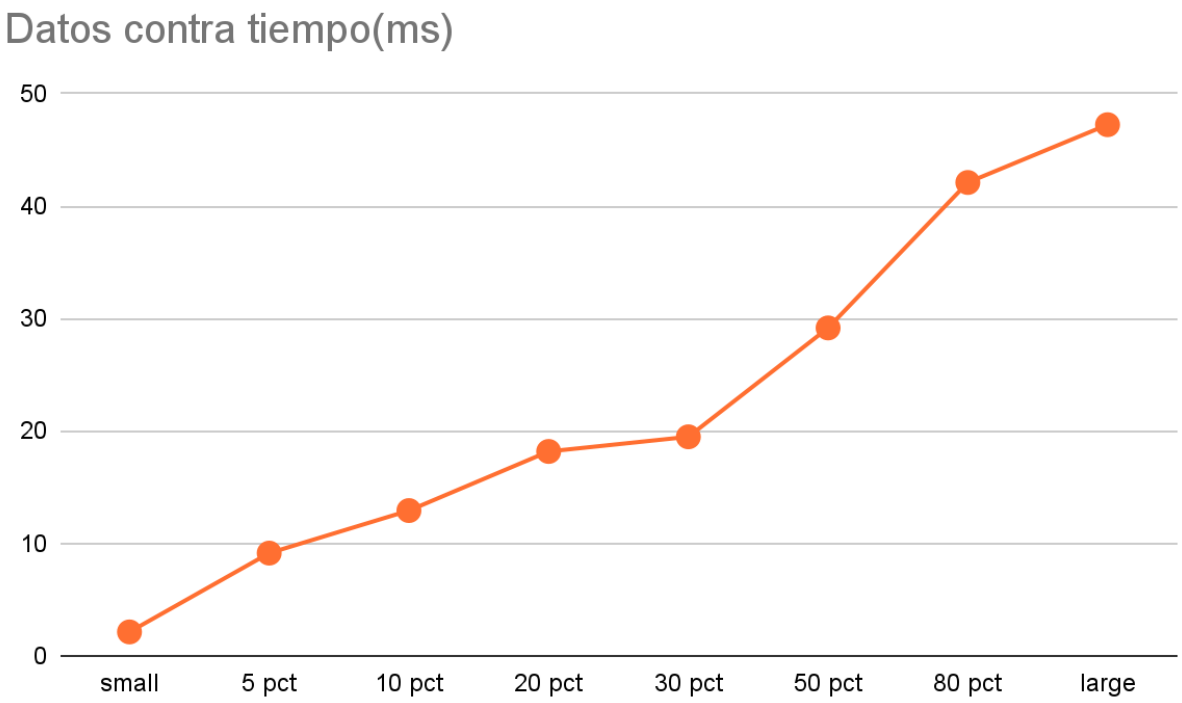
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	2.13
5pct	Dato2	9.14
10pct	**Dato3**	**12.924**
20pct	**Dato4**	**18.181**
30pct	**Dato5**	**19.483**
50pct	**Dato6**	**29.167**
80pct	**Dato7**	**42.101**
large	Dato8	47.240

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de mes, año, coordenadas del centro del área, radio en km, número de accidentes.

Es un poco más complejo que los demás requerimientos anteriores, sin embargo tiene muy buenos tiempos y pocas líneas de código, lo que demuestra una alta eficiencia. Este requerimiento en particular consiste de unos parámetros de ubicación y unas fechas con una cantidad de accidentes que se deseen ver, con esta información se utiliza la ecuación de haversine para poder encontrar correctamente la cantidad de accidentes que se desean ver en el espacio especificado y en el tiempo dado,

Requerimiento 7

Descripción

```python
def req_7(data_structs,year,month):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    init_time="00:00"
    fin_time="23:59"
    lista=req_2(data_structs,init_time,fin_time,year,month)
    list_month=lt.newList(datastructure="ARRAY_LIST")
    for i in range(1,calendar.monthrange(int(year),int(month))[1]+1):
        date_cmp=datetime(int(year),int(month),i,0,0)
        list_per_day=lt.newList(datastructure="ARRAY_LIST")
        max_min=lt.newList(datastructure="ARRAY_LIST")
        max_min['key']=i
        for j in lista:
            date_var=datetime.strptime(j['FECHA_OCURRENCIA_ACC'],'%Y/%m/%d')
            if date_cmp == date_var:
                lt.addLast(list_per_day,j)
        if list_per_day["size"]>1:
            lt.addLast(max_min,lt.firstElement(list_per_day))
            lt.addLast(max_min,lt.lastElement(list_per_day))
        elif list_per_day["size"]==1:
            lt.addLast(max_min,lt.firstElement(list_per_day))
        if max_min["size"]!=0:
            lt.addLast(list_month,max_min)
    mapa=mp.newMap(numelements=23,loadfactor=1,maptype='PROBING')
    lista_horas=lt.newList(datastructure='ARRAY_LIST')
    l=[lt.addLast(lista_horas,f'0{i}:00:00') for i in range(0,10)]
    k=[lt.addLast(lista_horas,f'{i}:00:00') for i in range(10,24)]
    for k in lista:
        l=datetime.strptime(k['HORA_OCURRENCIA_ACC'],"%H:%M:%S").time()
        for t in range(0,24):

            if t<=9:
                l
                if not mp.contains(mapa,f'0{t}:00:00') :
                    value=lt.newList(datastructure='ARRAY_LIST')
                    mp.put(mapa,f'0{t}:00:00',value)
                if datetime.strptime(f'0{t}:00:00',"%H:%M:%S").time()<=l<=datetime.strptime(f'0{t}:59:59',"%H:%M:%S").time():
                    lt.addLast(mp.get(mapa,f'0{t}:00:00')['value'],k)
            else:
                l
                if not mp.contains(mapa,f'{t}:00:00'):
                    value=lt.newList(datastructure='ARRAY_LIST')
                    mp.put(mapa,f'{t}:00:00',value)
                if datetime.strptime(f'{t}:00:00',"%H:%M:%S").time()<=l<=datetime.strptime(f'{t}:59:59',"%H:%M:%S").time():
                    lt.addLast(mp.get(mapa,f'{t}:00:00')['value'],k)
    datos_tuplas=lt.newList(datastructure='ARRAY_LIST')
    size=0
    for j in lt.iterator(mapa['table']):
        if j['key']!=None:
            size+=j['value']['size']
            lt.addLast(datos_tuplas,(j['key'],j['value']['size']))

    df=pd.DataFrame(datos_tuplas['elements'],columns=['key', 'value'])

    df = df.set_index("key").loc[lista_horas['elements']].reset_index()
    plt.bar(df["key"], df["value"])
    plt.xticks(rotation=90)
    plt.xlabel("Hora del día")
    plt.ylabel("Número de accidentes")
    plt.title(f"Frecuencia de {size} accidentes por hora del día, para el mes: {month} año: {year}")
    plt.show()
    return list_month
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de mes y año.

Entrada	Mes. Año entre 2015 y 2022
Salidas	Para cada día del mes en el año dados, dar la información del accidente ocurrido más temprano y más tardío

Implementado (Sí/No)	Si. Andrés Cáceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
llamar al req2	O(n)
asignaciones y primer for	O(1)
for j in lista	O(n)
todos los if	O(1)
for k in lista	O(n)
for j in lt.iterator(mapa[“table”])	O(m)
df=pd.dataframe	O(n)
df=df.set_index	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 5000HS with Radeon Graphics**
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	3.10
5 pct	10.41
10 pct	**13.219**
20 pct	**19.913**
30 pct	**20.675**
50 pct	**29.985**
80 pct	**42.396**
large	48.318

Tablas de datos

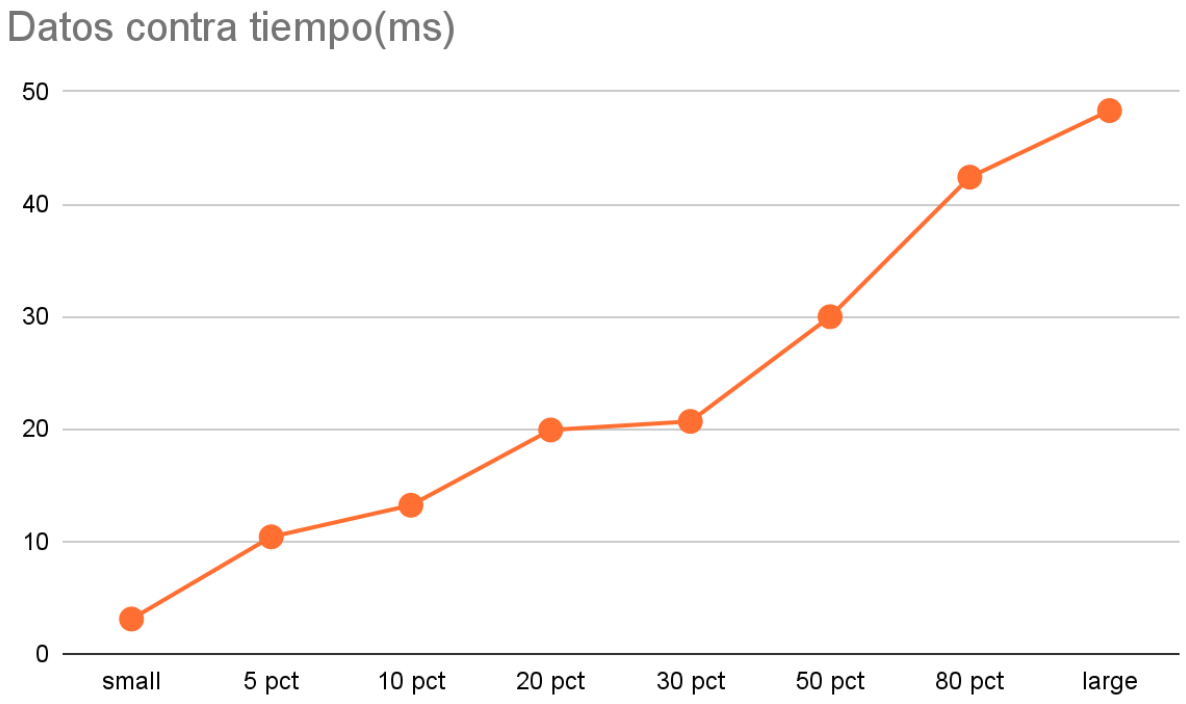
Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	3.10
5pct	Dato2	10.41

10pct	Dato3	13.219
20pct	Dato4	19.913
30pct	Dato5	20.675
50pct	Dato6	29.985
80pct	Dato7	42.396
large	Dato8	48.318

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de mes y año.

Aunque también es un requerimiento distinto a los demás debido a la respuesta esperada y a su vez es también más complejo tiene buenos tiempos y muchas líneas de código pero es por lo que se muestra un histograma con la frecuencia de accidentes en un rango de horas y se hace un tabulate con el accidente más temprano y más tarde para cada día del mes. Para acortar un poco la cantidad de código se utilizó un llamado de la función 2 a la que se le pasó por parámetro un mes y año variable y una hora inicial en 00:00 y final en 23:59, esto con el fin de asegurar que se cubrirán todos los accidentes del mes.

Requerimiento 8

Descripción

```
def req_8(data_structs,date_init,date_fin,type_acc):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8

    m = folium.Map(location=[4.6097, -74.0817], zoom_start=12)
    lista=None
    lista_final=lt.newList(datastructure='ARRAY_LIST')

    for i in lt.iterator(data_structs['data_acc']['table']):
        if i['key']==type_acc:
            lista=om.values(i['value'],datetime.strptime(str(date_init+' 00:00:00'), '%Y/%m/%d %H:%M:%S'),datetime.strptime(str(date_fin+' 23:59:59'), '%Y/%m/%d %H:%M:%S'))
    for j in lt.iterator(lista):
        if j['size']!=1:
            for k in lt.iterator(j):
                if k['GRAVEDAD']=='CON HERIDOS':
                    folium.Marker(location=[float(k['LATITUD']),float(k['LONGITUD'])],popup='CON HERIDOS '+type_acc,icon=folium.Icon(color='green')).add_to(m)
                elif k['GRAVEDAD']=='CON MUERTOS':
                    folium.Marker(location=[float(k['LATITUD']),float(k['LONGITUD'])],popup='CON MUERTOS '+type_acc,icon=folium.Icon(color='red')).add_to(m)
                else:
                    folium.Marker(location=[float(k['LATITUD']),float(k['LONGITUD'])],popup='SOLO DANOS '+type_acc,icon=folium.Icon(color='black')).add_to(m)
        else:
                if j['elements'][0]['GRAVEDAD']=='CON HERIDOS':
                    folium.Marker(location=[float(j['elements'][0]['LATITUD']),float(j['elements'][0]['LONGITUD'])],popup='CON HERIDOS '+type_acc,icon=folium.Icon(color='green')).add_to(m)
                elif j['elements'][0]['GRAVEDAD']=='CON MUERTOS':
                    folium.Marker(location=[float(j['elements'][0]['LATITUD']),float(j['elements'][0]['LONGITUD'])],popup='CON MUERTOS '+type_acc,icon=folium.Icon(color='red')).add_to(m)
                else:
                    folium.Marker(location=[float(j['elements'][0]['LATITUD']),float(j['elements'][0]['LONGITUD'])],popup='SOLO DANOS '+type_acc,icon=folium.Icon(color='black')).add_to(m)
    m.save('mapa.html')
    webbrowser.get('safari').open('file://' + os.path.abspath('mapa.html'))
```

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de fecha inicial del intervalo, fecha final del intervalo y clase de accidente.

Entrada	Fecha inicial del intervalo. Fecha final del intervalo. Clase de accidente
Salidas	El número total de accidentes ocurridos en el rango de fecha. Un mapa interactivo de clústeres que muestre todos los accidentes según su tipo en el mapa de Bogotá, en donde cada marcador tenga asignado un color de acuerdo con su gravedad.
Implementado (Sí/No)	Si. Ángel Farfán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
primer for	O(n)
segundo for	O(n)
operaciones dentro del segundo for	O(1)
abrir el mapa en el navegador	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	AMD Ryzen 7 5000HS with Radeon Graphics
Memoria RAM	16GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
small	2.40
5 pct	9.103
10 pct	**11.258**
20 pct	**17.824**
30 pct	**18.342**
50 pct	**27.513**
80 pct	**40.194**
large	46.932

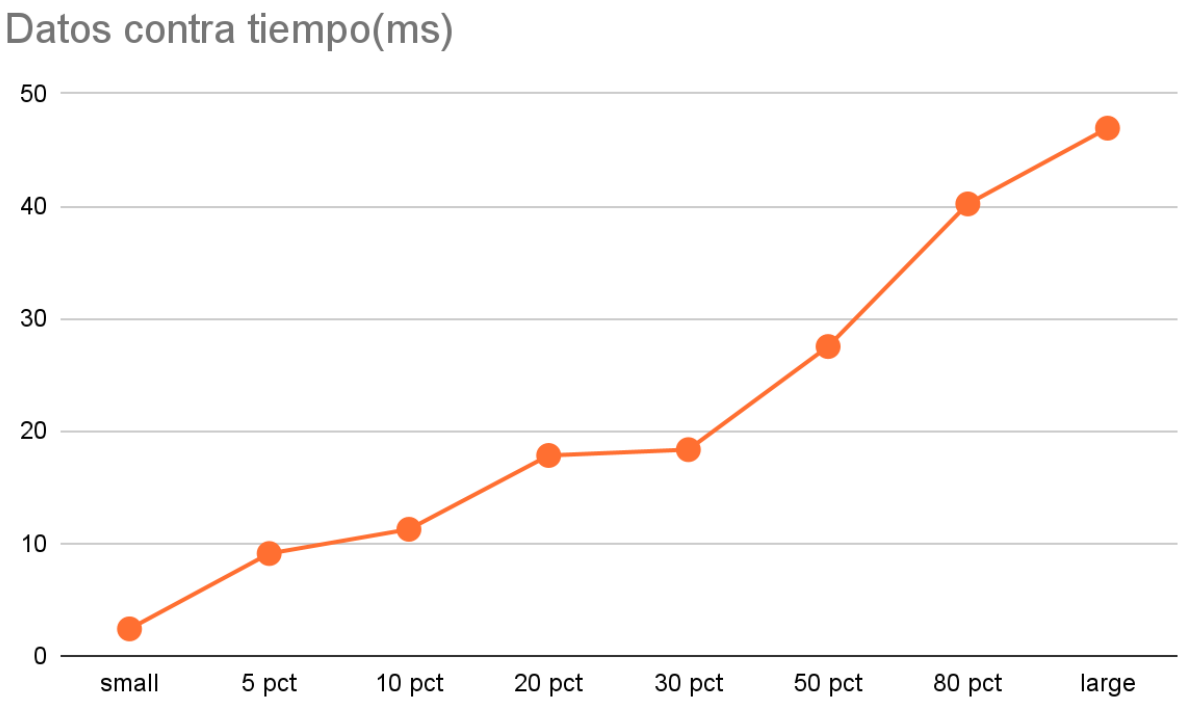
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Salida	Tiempo (ms)
small	Dato1	2.40
5pct	Dato2	9.103
10pct	**Dato3**	**11.258**
20pct	**Dato4**	**17.824**
30pct	**Dato5**	**18.342**
50pct	**Dato6**	**27.513**
80pct	**Dato7**	**40.194**
large	Dato8	46.932

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento se encarga de analizar, seccionar y ordenar los datos del datastructs dentro de una lista de tipo array según los parámetros de fecha inicial del intervalo, fecha final del intervalo, clase del accidente.

Este requerimiento corresponde al bono y es especial debido a su respuesta esperada. Principalmente se crean listas y un mapa usando la librería folium, se separan los datos mediante comparaciones de fecha inicial y fecha final, se realizan otras comparaciones con el fin de separar los accidentes según su clase , finalmente todo esto es mostrado mediante un mapa interactivo con puntos de colores distintos según la clase del accidente ubicados según los datos de espacio de cada accidente y a los puntos se le agregaron unos pop-up con toda su información.