ANÁLISIS DEL RETO 3

Gabriel Padilla, 202224552, g.padilla@uniandes.edu.co

Simón Saavedra, 202217058 s.saavedrap@uniandes.edu.co

Emilio Zea, 202221744, e.zea@uniandes.edu.co

Requerimiento <<1>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento. (`FECHA_HORA_ACC inicial y final`)
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Crear una lista donde se va a almacenar la respuesta	O(1)
Paso 2: Recorrido de los datos que se quiere saber si cumplen o no con el requerimiento	O(n)
Paso 3: Revisa si la fecha es parte del intervalo dado por el usuario	O(1)
Paso 4: Añadirlo a la lista si se cumple el paso 3	O(1)
TOTAL	***O(n)***

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).

Todas las pruebas fueron realizadas para el intervalo de fechas (2016/11/01 - 2016/11/08)

Memoria RAM	**8 GB**
Sistema Operativo	MacOs Ventura, m1

Entrada	Tiempo (ms)
small	1.12
5 pct	3.819
10 pct	7.206
20 pct	10.877
30 pct	16.57
50 pct	17.515
80 pct	30.304
large	35.020

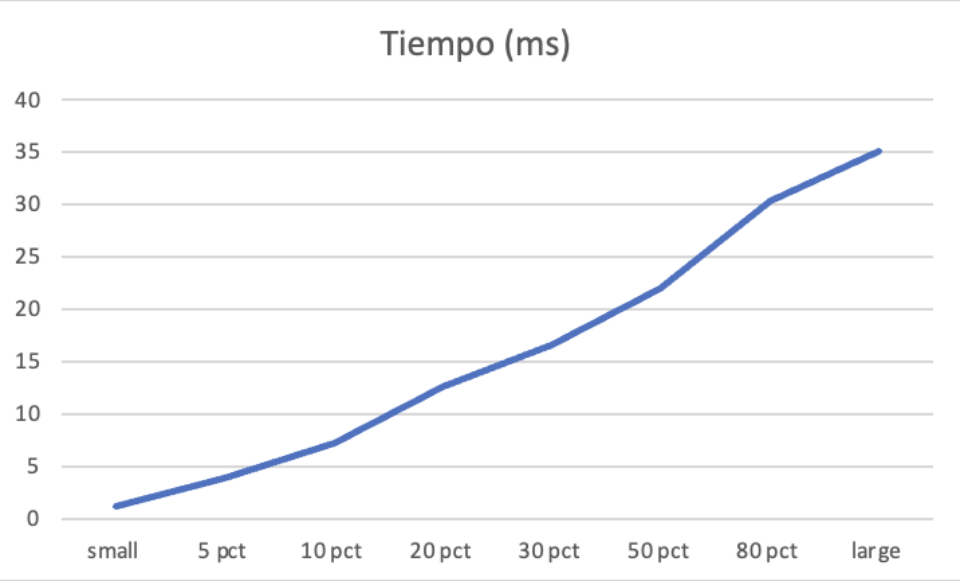
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
small	1.12
5 pct	3.819
10 pct	7.206
20 pct	10.877
30 pct	16.57
50 pct	17.515
80 pct	30.304
large	35.020

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.

Aunque es cierto que la obtención de un elemento en un ArrayList mediante su posición tiene una complejidad constante, en la implementación del requerimiento en cuestión se ha obtenido un orden lineal O(n). Esto se debe a que, antes de buscar el elemento en la lista, se realiza una verificación para determinar si dicho elemento se encuentra en la lista. En el peor de los casos, al buscar un elemento en una lista, es necesario recorrer toda la lista, lo que se traduce en una complejidad lineal.

Este comportamiento se puede observar claramente en la gráfica experimental, ya que los datos se ajustan a la línea de tendencia y se puede apreciar el comportamiento lineal esperado.

Podemos ver en la gráfica que hay coherencia con el algoritmo y que a medida que crece la entrada de datos crece la complejidad temporal acorde a la entrada

Requerimiento <<2>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

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

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Convertir mes a mayúsculas por si el usuario no lo escribe bien	O(1)
Paso 2: Se realiza una búsqueda de datos con la función get() de DISClib que tiene como peor caso 0(n)	O(n)
Paso 3: For que itera por los datos, en el peor caso	O(n)
Paso 4: Se obtiene la hora de cada accidente t se compara para ver si cumple con los requerimentos	O(1)
TOTAL	***O(n)***

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).

Todas las pruebas fueron realizadas para los siguientes parámetros de entrada que se mantienen constantes por todos los tamaños de entrada:

Año: 2016

Mes: Mayo

Intervalo de horas: 18:00:00 – 23:59:00

Entrada	**Tiempo (ms)**
small	0.108
5 pct	0.232
10 pct	0.406
20 pct	0.790
30 pct	0.957
50 pct	1.813
80 pct	2.685
large	4.152

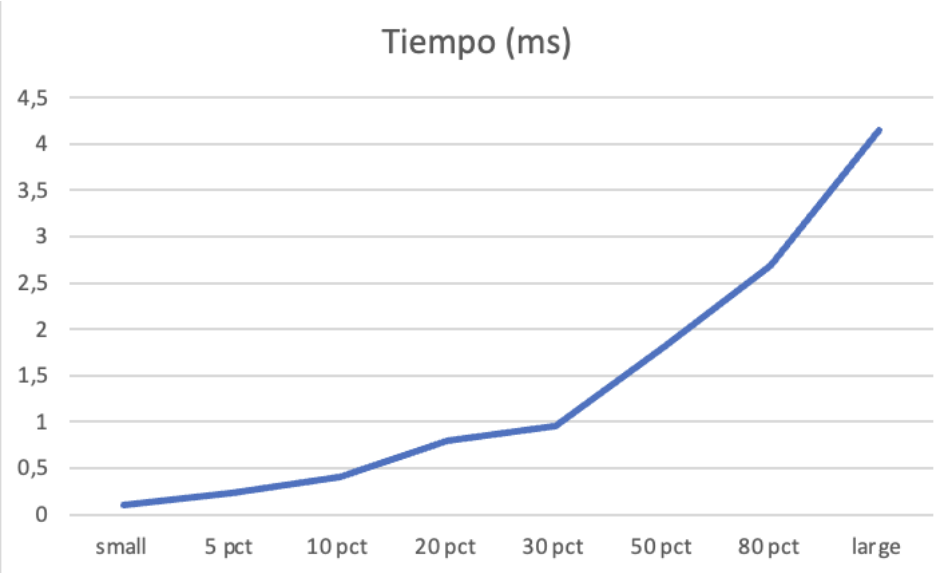
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (ms)**
small	0.108
5 pct	0.232
10 pct	0.406
20 pct	0.790
30 pct	0.957
50 pct	1.813
80 pct	2.685
large	4.152

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.

Aunque la complejidad de obtener un elemento en un ArrayList mediante su posición es constante, la implementación del requerimiento 2 presenta un orden lineal O(n). Esto se debe a que, antes de buscar el elemento en la lista, se realiza una verificación para determinar si el elemento está en ella. En el peor de los casos, buscar un elemento en una lista requiere recorrer toda la lista, lo que resulta en una complejidad lineal.

Este comportamiento se puede ver en la gráfica experimental, donde se ajusta a la línea de tendencia y se puede apreciar el comportamiento lineal esperado. El código comienza convirtiendo la entrada de mes a mayúsculas y accediendo a los datos del año y mes. A continuación, se obtienen los datos del año y se filtran por mes, lo que da como resultado una lista de elementos. Después, se crea un iterador de los datos y se recorre para agregar los elementos que cumplan con el criterio de hora a una nueva lista.

Podemos ver en la gráfica que hay coherencia con el algoritmo y que a medida que crece la entrada de datos crece la complejidad temporal acorde a la entrada. Podemos ver que al final de la gráfica los tiempos crecen de manera exponencial, pero esto se debe a que el intervalo de datos pasa de ser de 10% a 30% y debido a esto se la gráfica salte, sin embargo, podemos deducir que es lineal.

Requerimiento <<3>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

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

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: usa un mp.get() para obtener los datos por clase de accidente	O (1)
Paso 2: Crea un nuevo mapa ordenado om.newMap()	O (1)
Paso 3: Itera por los datos y determina si la vía dada por el usuario está en la dirección del accidente que se está iterando	O(n)
Paso 4: Con maxkey solicita la llave más grande	O (1)
Paso 5: Usa om.get() para tomar los datos de los 3 más recientes que son los que se quieren solicitar	O (log n)
TOTAL	***O(n)***

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).

Los siguentes datos se mantienen constante en todas las pruebas:

Tipo de accidente: CHOQUE

A lo largo de la vía: AV AVENIDA DE LAS AMERICAS

Entrada	**Tiempo (ms)**
small	1.181
5 pct	7.329
10 pct	13.938
20 pct	21.795
30 pct	28.705
50 pct	44.164
80 pct	71.316
large	100.640

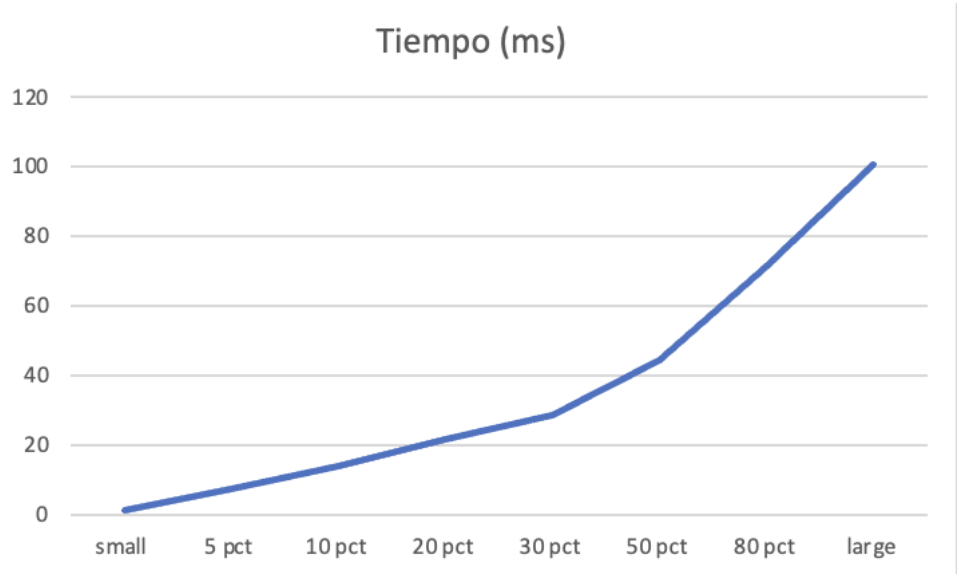
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
small	1.181
5 pct	7.329
10 pct	13.938
20 pct	21.795
30 pct	28.705
50 pct	44.164
80 pct	71.316
large	100.640

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.

Podemos ver que el algoritmo es bastante eficiente, todos sus pasos son de tiempo constante O(1) o logarítmico O(log n) para las operaciones del mapa ordenado, sin embargo, se deben recorrer todos los datos por lo que la complejidad de este recorrido en su peor caso es de O(n). El hecho de que los datos ya estén cargados en un mapa de clase de accidentes nos ayuda mucho a realizar este requerimiento ya que no se debe separar los datos por la clase de accidente ya que esto ya fue implementado en la carga.

Podemos ver en la gráfica que hay coherencia con el algoritmo y que a medida que crece la entrada de datos crece la complejidad temporal acorde a la entrada. Podemos ver que al final de la gráfica los tiempos crecen de manera exponencial, pero esto se debe a que el intervalo de datos pasa de ser de 10% a 30% y debido a esto se la gráfica salte, sin embargo, podemos deducir que es lineal.

Requerimiento <<4>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, fecha inicial, fecha final y gravedad a consultar
Salidas	Diccionario de listas de los valores de los 5 accidentes mas antiguos en la franja consultada y el total de accidentes en esta franja
Implementado (Sí/No)	Si, Simon Saavedra

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Categorizar los datos importados en una lista que discrimine por la franja de fechas en los parametros.	O(n)
Categorizar la lista anterior en una que discrimine por la gravedad del accidente	O(n)
Encontrar el tamaño de la lista	O(1)
Organizar la lista por su fecha y hora de menor a mayor	***O(nlogn)***
Crear una sublista de los primeros 5, formatearla y retornarla	***O(1)***
TOTAL	***O(nlogn)***

Pruebas Realizadas

Utilizando librerías para encontrar el delta en tiempo entre el inicio y el final de la ejecución del programa, se recolectaron estos datos para comparar su crecimiento con el crecimiento temporal esperado según el análisis del algoritmo.

Para las pruebas, se utilizó 2016/10/01 como fecha inicial y 2018/10/01 como fecha final, ademas de una gravedad de “Con muertos”

Estas pruebas se realizaron en:

Procesadores	**Intel i7-9700K 3.9 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

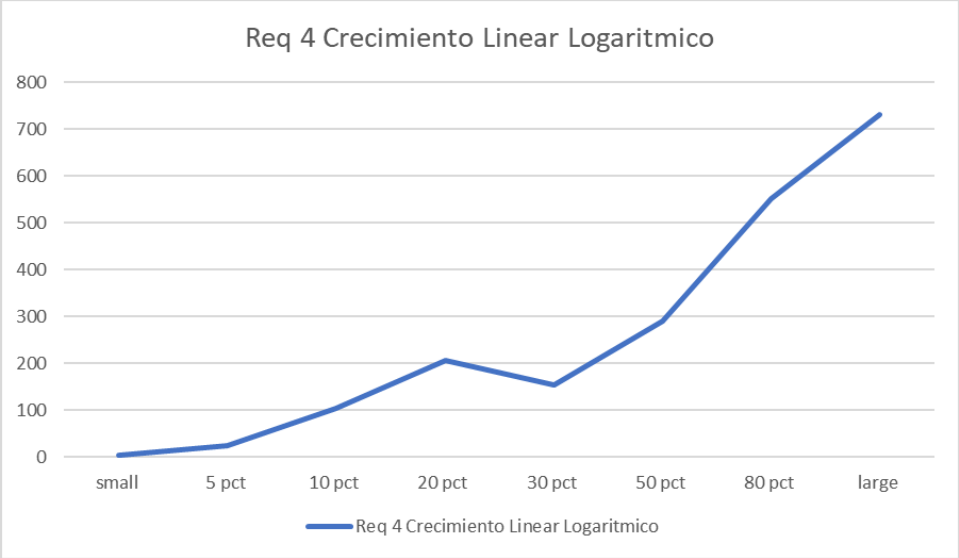
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
small	4.5759
5 pct	22.9355
10 pct	104.49
20 pct	206.867
30 pct	153.769
50 pct	290.822
80 pct	550.866
large	730.344

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Es posible detallar en la gráfica que el crecimiento de la funcion es un poco más agresivo que el de una funcion linear. Con más datos, este comportamiento linear logaritmico se acentuaría.

Requerimiento <<5>>

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El año de los accidentes, el mes de los accidentes y la localidad donde ocurrieron
Salidas	Una lista con los 10 accidentes menos recientes ocurridos en un mes y año para una localidad de la ciudad.
Implementado (Sí/No)	Emilio Zea(si)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: usa un om.get() para obtener los datos por año	O (logn)
Paso 2: usa un mp.get() para obtener los datos por mes	O (1)
Paso 3: Itera por los datos y si están en la localidad que eligió el usuario los guarda en un mapa	O(m) m=datos del mes
Paso 4: Dependiendo de las llaves del mapa que son las fechas, se van guardando los elementos en una lista.	O (m)
Paso 5: Se organiza la lista de la fecha mas lejana a la mas cercana a la actualidad con mergesort	O(nlogn)
Paso 6: Se hace una sublista con los primeros 10 valores de la lista organizada, en caso de que hayan mas de 10.	O(10)
Complejidad final:	O(logn)

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).

Datos constantes:

Año=2016

Mes=Noviembre

Localidad= Fontibon

Entrada	**Tiempo (ms)**
small	0,771

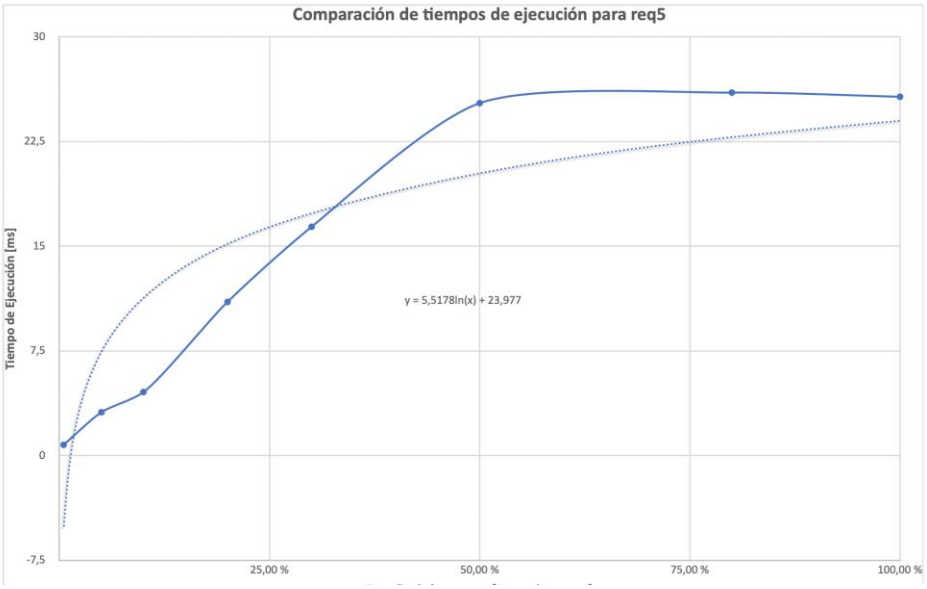
5 pct	3,106
10 pct	4,552
20 pct	11,005
30 pct	16,390
50 pct	25,243
80 pct	26,00
large	25.699

Tablas de datos

Entrada	**Tiempo (ms)**
small	0,771
5 pct	3,106
10 pct	4,552
20 pct	11,005
30 pct	16,390
50 pct	25,243
80 pct	26,00
large	25.699

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Como podemos ver, la gráfica tiene una forma bastante similar a la de O(logn), sin embargo, como la linea de referencial comienza desde –6 no se nota tanta correlación. Si subiéramos el origen, la correlación los datos y la línea de referencia sería mucho más similar. Por estas razones puedo decir que mi análisis de que la complejidad de la función es O(logn) es correcta y, por lo tanto, la función es eficiente.

Requerimiento <<6>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El Año, el mes, la Latitud y la Longitud del centro del area, el radio del área en km y el Número de accidentes que quiere que se muestren.
Salidas	Los N accidentes ocurridos, organizados por su cercanía al centro de la zona, esto se saca utilizando la formula de Haversine.
Implementado (Sí/No)	Emilio Zea (Si)

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: usa un om.get() para obtener los datos por año	O (logn)
Paso 2: usa un mp.get() para obtener los datos por mes	O (1)
Paso 3: Itera por los datos, se revisa su lejanía al punto centro que dio el usuario con la fórmula de Haversine y si están dentro del radio, se meten a un mapa	O(m) m=datos del mes
Paso 4: Dependiendo de las llaves del mapa que son las fechas, se van guardando los elementos en una lista.	O (m)
Paso 5: Se organiza la lista de la fecha más lejana a la más cercana a la actualidad con mergesort	O(nlogn)
Paso 6: Se hace una sublista con los primeros N valores de la lista organizada, en caso de que hayan más de N.	O(10)
Complejidad final:	O(n)

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).

Datos constantes:

Año=2022

Mes= Enero

Latitud= 4.674

Longitud=-74.068

Radio=5

N=3

Entrada	**Tiempo (ms)**
small	0,7280
5 pct	5,372
10 pct	10,66
20 pct	23,92
30 pct	47,792
50 pct	68,452
80 pct	139,192
large	178,8092

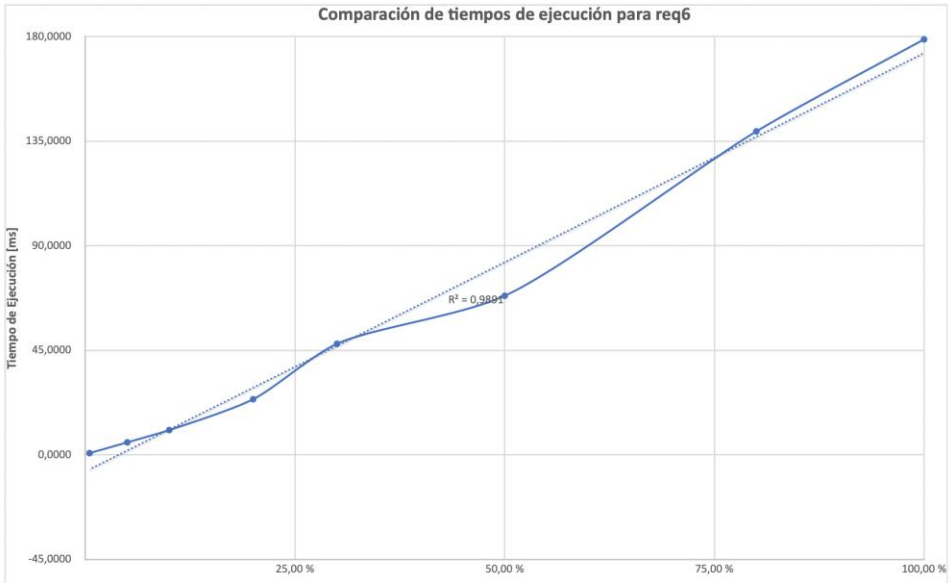
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (ms)**
small	0,7280
5 pct	5,372
10 pct	10,66
20 pct	23,92
30 pct	47,792
50 pct	68,452
80 pct	139,192
large	178,8092

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.

Como se puede ver en el gráfico, la complejidad temporal de la función es O(n) pues está muy cerca de una gráfica lineal. Gracias a que tiene esta complejidad podemos decir que la funcion es bastante eficiente temporalmente hablando.

Requerimiento <<7>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, mes y año a consultar
Salidas	Gráfica de barras y accidente más temprano y más tarde de cada día del mes
Implementado (Sí/No)	Si, Simón Saavedra Pérez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Importar los datos y filtrarlos por año y mes	O(n)

Para cada día en el mes, encontrar el accidente más temprano y más tarde	O(n)
Categorizar los accidentes de cada día en una franja horaria para graficarlos	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

Utilizando librerías para encontrar el delta en tiempo entre el inicio y el final de la ejecución del programa, se recolectaron estos datos para comparar su crecimiento con el crecimiento temporal esperado según el análisis del algoritmo.

Para las pruebas, se utilizó diciembre de 2019 como parámetro de entrada.

Estas pruebas se realizaron en:

Procesadores	**Intel i7-9700K 3.9 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

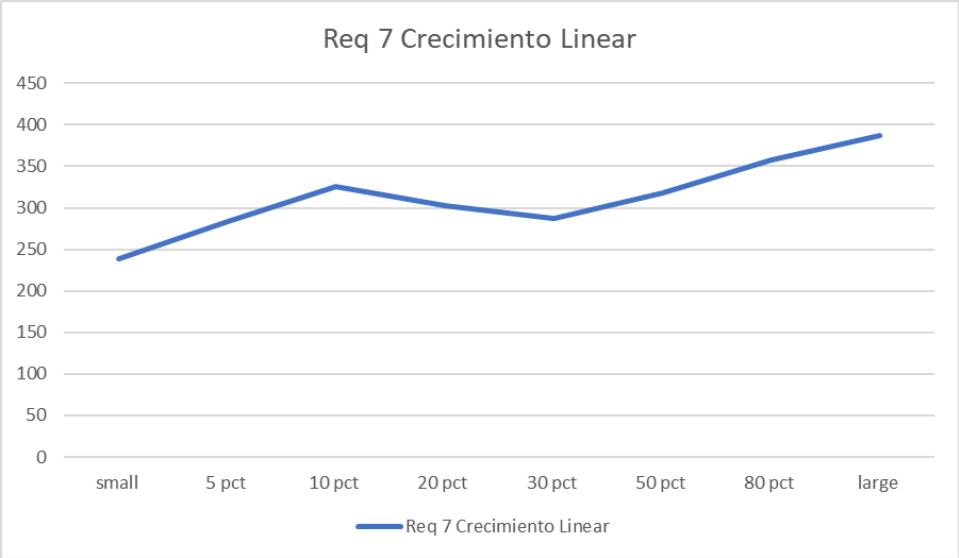
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (ms)**
small	238.1675
5 pct	283.2649
10 pct	325.7616
20 pct	301.9550
30 pct	287.7896
50 pct	317.7442
80 pct	357.5993
large	386.4803

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Gracias al uso de estructuras de datos apropiadas, el tiempo que la función demora en ejecutarse es linealmente proporcional al tamaño de los datos. Esto se evidencia en la gráfica: el tiempo que toma crece en una línea recta.

Requerimiento Ejemplo

Descripción

```python
def get_data(data_structs, id):
    """
    Retorna un dato a partir de su ID
    """
    pos_data = lt.isPresent(data_structs["data"], id)
    if pos_data > 0:
        data = lt.getElement(data_structs["data"], pos_data)
        return data
    return None
```

Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

Entrada	Estructuras de datos del modelo, ID.
Salidas	El elemento con el ID dado, si no existe se retorna None
Implementado (Sí/No)	Si. Implementado por Juan Andrés Ariza

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Obtener el elemento (getElement)	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

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

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**
small	0.05
5 pct	0.33
10 pct	1.28
20 pct	2.54
30 pct	4.98
50 pct	7.51
80 pct	13.81
large	25.97

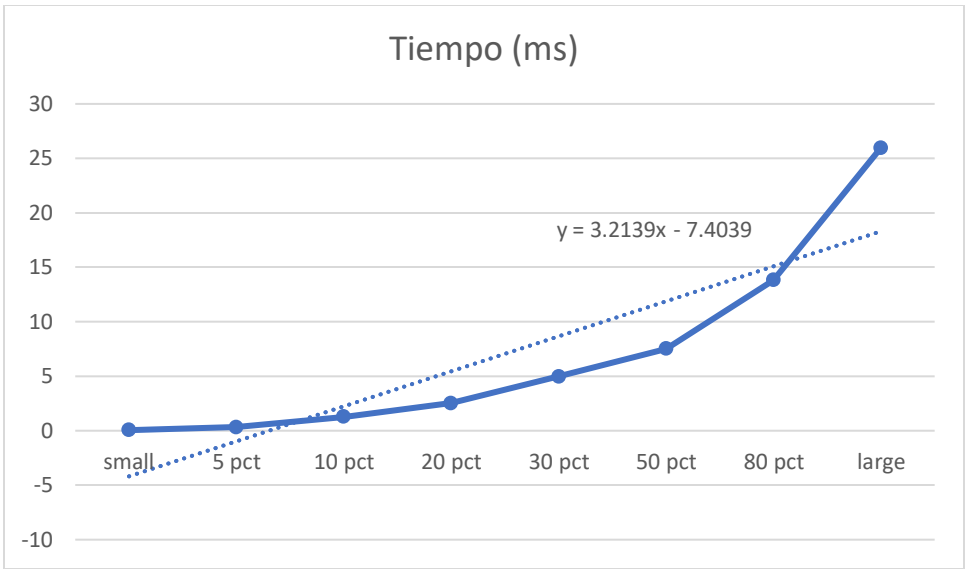
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	0.05
5 pct	Dato2	0.33
10 pct	Dato3	1.28
20 pct	Dato4	2.54
30 pct	Dato5	4.98
50 pct	Dato6	7.51
80 pct	Dato7	13.81
large	Dato8	25.97

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.