ANÁLISIS DEL RETO

Tomas Diaz, 202220658, t.diazv@

Samuel Peña, 202028273, ss.pena

Manuel Pinzon, 202125748, mma.pinzonpi

Requerimiento <<n>>

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 se implementó y quien lo hizo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Pruebas Realizadas

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

Entrada	**Tiempo (s)**

Tablas de datos
Las tablas con la recopilación de datos de las pruebas.

Graficas
Las gráficas con la representación de las pruebas realizadas.

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

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

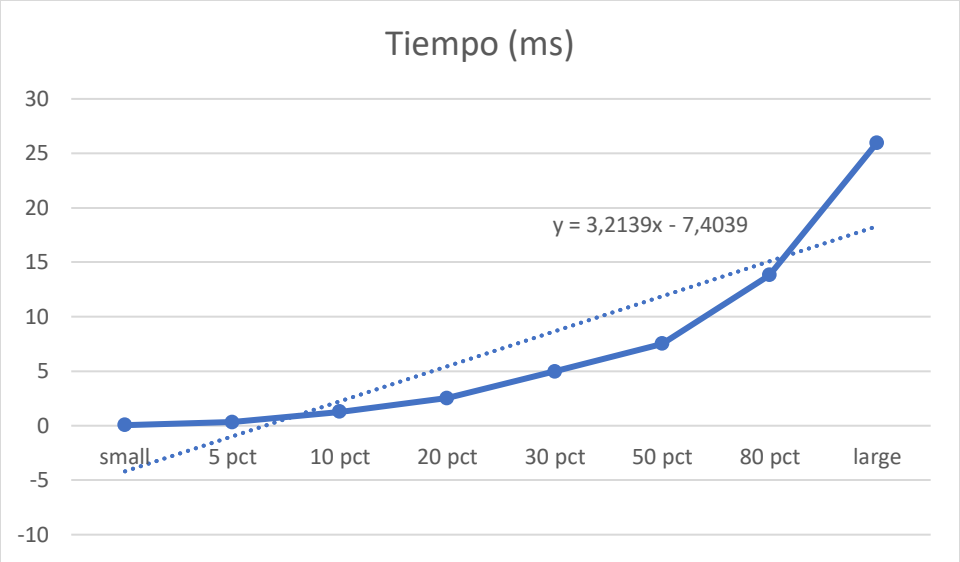
Tablas de datos

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.

Tablas y gráficas carga de datos

Carga PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	21820.23	82.02
0.5	21820.78	80.46
0.7	21818.19	79.13
0.9	21817.04	78.81

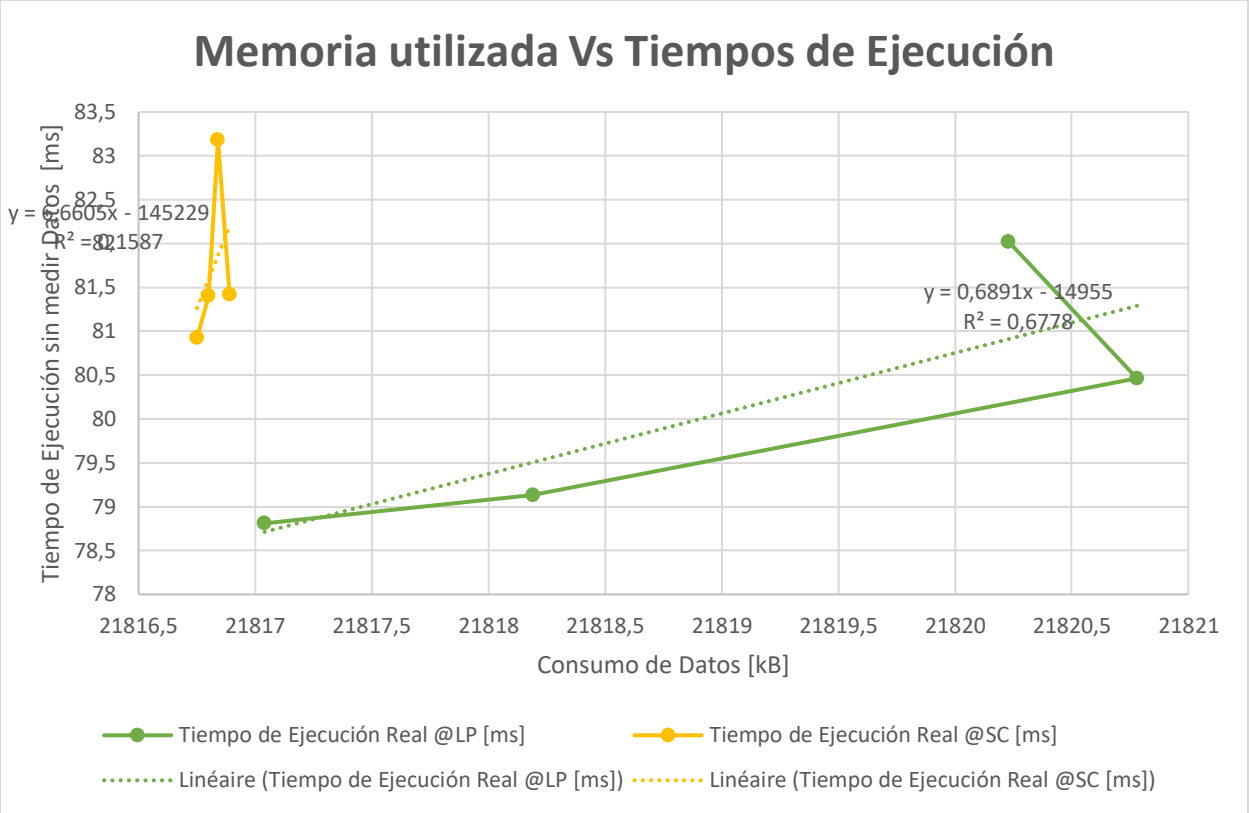
Carga CHAINING

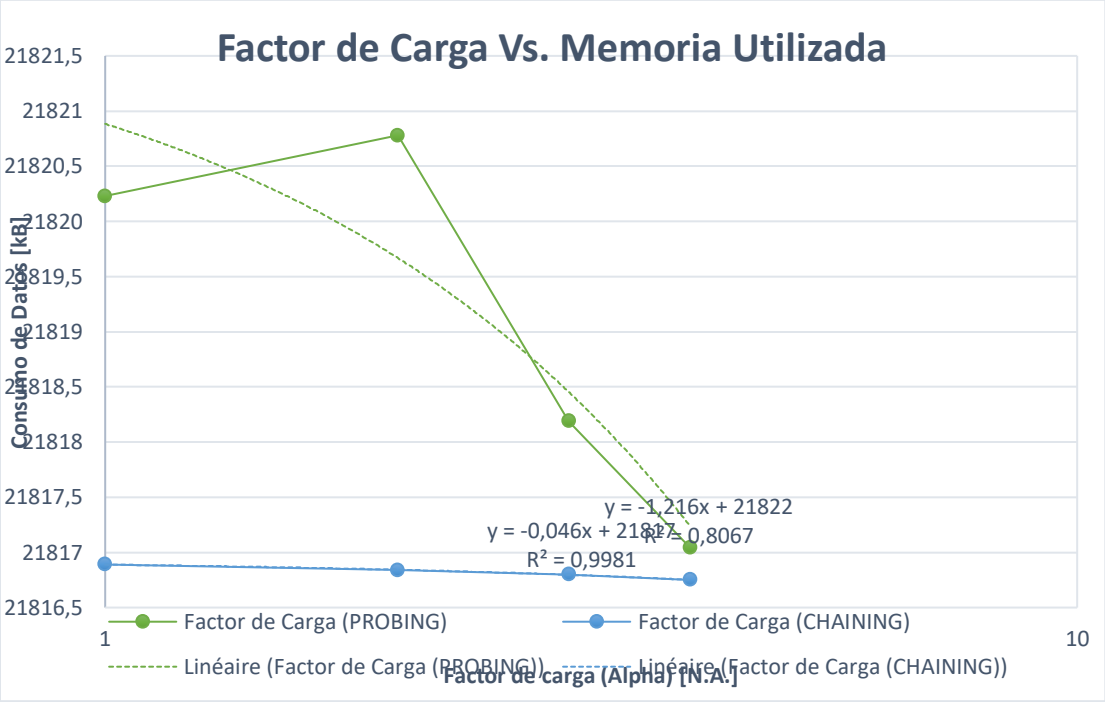
Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	21816.89	81.42
4.00	21816.84	83.18

6.00	21816.80	81.40
8.00	21816.75	80.92

Graficas

- Comparación de memoria y tiempo de ejecución para PROBING y CHAINING





Requerimiento 1

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

Descripción

```python
def req_1(data_structs , anio : int, codigo : int):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    # retornar ACTIVIDAD ECONOMICA con mayor SALDO A PAGAR dado un AÑO y SECTOR ECONOMICO
    dicco = organizar_sector(data_structs, "Código sector económico")
    anio_b = devolver_value(dicco,anio)
    sector_b = devolver_value(anio_b, codigo)
    resultado = encontrar_mayor_criterio(sector_b, "Total saldo por pagar")
    return resultado
```

Este requerimiento lo que busca es organizar los datos por años y luego por el sector. De esta forma podemos indagar en el año seleccionado sobre los datos que queremos analizar, principalmente sobre saldo por pagar.

Entrada	Data_structs: todos los datos cargados Anio: el año en el que se busca el mayor saldo total de impuestos Codigo : el sector en el que se desea buscar
Salidas	Devuelve el diccionario del mayor total saldo a favor del tiempo dado, con las partes que el requerimiento pide
Implementado (Sí/No)	Si. Por Manuel Pinzon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: organizar sector (dic según sectores)	O(n) pasa por todos los datos
Paso 2: dic (según el año) = devolver_value(devolver año buscado)	O(n) encuentra el valor en las llaves (menor el n en el que busca)
Paso 3: sector buscado (según sector y año dado) = devolver_value(del sector en el año especifico)	O(n) encuentra el sector especifico(menor n)
Paso 4: final = encontrar mayor (del sector buscado según el total saldo a favor)	O(n) busca dentro de esta subseccion de datos
Paso 5: res = filtrar dic con llaves (organiza el dic, para lo que quiera devolver)	O(n)las llaves de un valor(n muy pequeño)
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).

Memoria RAM	**32 GB**
Sistema Operativo	Windows 11

Procesador : Intel i

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

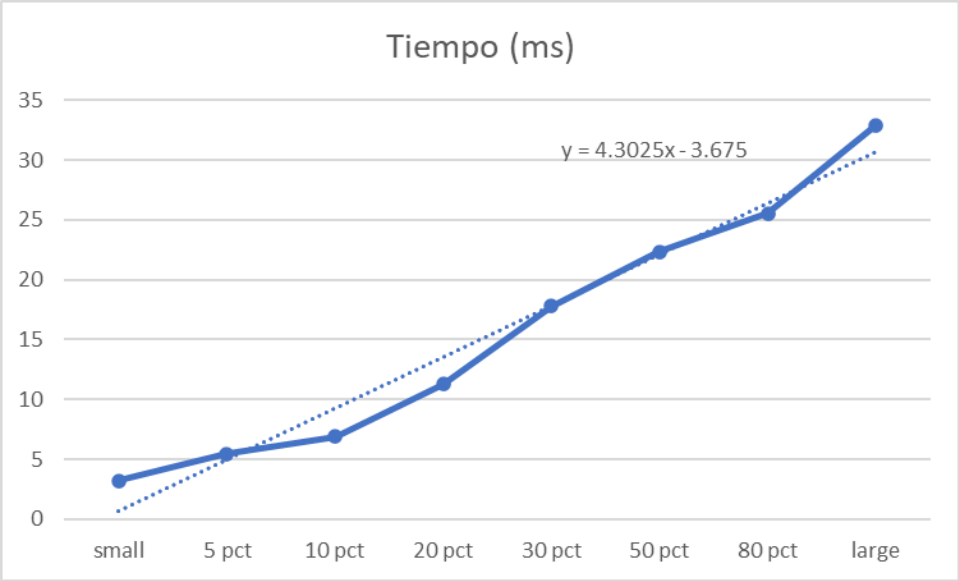
Se realizo con el año 2021 y en el sector 3

Entrada	**Memoria utilizada**
5 pct	12.94
20 pct	13.01
50 pct	12.98
large	13.01

Entrada	**Tiempo (ms)**
5 pct	3.22
20 pct	11.3
50 pct	22.36
large	32.92

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 complejidad espacial de este requerimiento consiste en o (n), porque todo el tiempo se hace una búsqueda de los mismos datos una vez, incluso a la vez que va subdividiéndose, se hace una búsqueda por menor cantidad de datos porque se va dividiendo, de esa manera el mayor recorrido se puede evidenciar en el primer caso se divide por subsectores en cada año. A diferencia del reto 1, en este utilizamos los mapas en hacer la organización de los diccionarios, facilitándonos la hora de ir en llave en llave con el keyset, o la velocidad de devolver una llave también nos lo facilito

Requerimiento 2

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

Descripción

```
def req_2(data_structs, anio, codigo):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    dic = organizar_sector(data_structs, "Código sector económico")
    #dic con sector y anios organizados
    anio_buscado = devolver_value(dic, int(anio))
    #dic del anio
    sector_buscado = devolver_value(anio_buscado, codigo)
    #devuelve acorde al anio y el sector, en este caso una lista
    final = encontrar_mayor_criterio(sector_buscado, "Total saldo a favor")
    #el mayor
    res = filtrar_dic_con_por_llaves(final,["Código actividad económica", "Nombre actividad económica", "Código sector económico","Nombre sector económico",
                                            "Código subsector económico", 'Nombre subsector económico', "Total ingresos netos", "Total costos y gastos",
                                            "Total saldo a pagar", "Total saldo a favor"])
    # organiza a lo que quiero que muestre

    return res
```

En este requerimiento lo que hicimos fue organizar los datos los años y después por el código del sector económico. Prosiguiendo a esto, buscamos el año buscado en la estructura mencionada, y después buscamos el código del cual se está buscando la información. Después de tener la zona específica donde se está buscando, buscamos el criterio mayor acorde a el total saldo a favor.

Entrada	Data_structs: todos los datos cargados Anio: el año en el que se busca el mayor saldo total de impuestos Codigo : el sector en el que se desea buscar
Salidas	Devuelve el diccionario del mayor total saldo a favor del tiempo dado, con las partes que el requerimiento pide
Implementado (Sí/No)	Si. Por Tomas DIaz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: organizar sector (dic según sectores)	O(n) pasa por todos los datos
Paso 2: dic (según el año) = devolver_value(devolver año buscado)	O(n) encuentra el valor en las llaves (menor el n en el que busca)
Paso 3: sector buscado (según sector y año dado) = devolver_value(del sector en el año especifico)	O(n) encuentra el sector especifico(menor n)
Paso 4: final = encontrar mayor (del sector buscado según el total saldo a favor)	O(n) busca dentro de esta subseccion de datos
Paso 5: res = filtrar dic con llaves (organiza el dic, para lo que quiera devolver)	O(n)las llaves de un valor(n muy pequeño)

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).

Memoria RAM	**8 GB**
Sistema Operativo	Windows 10

Procesador : AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx 2.60 GHz

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

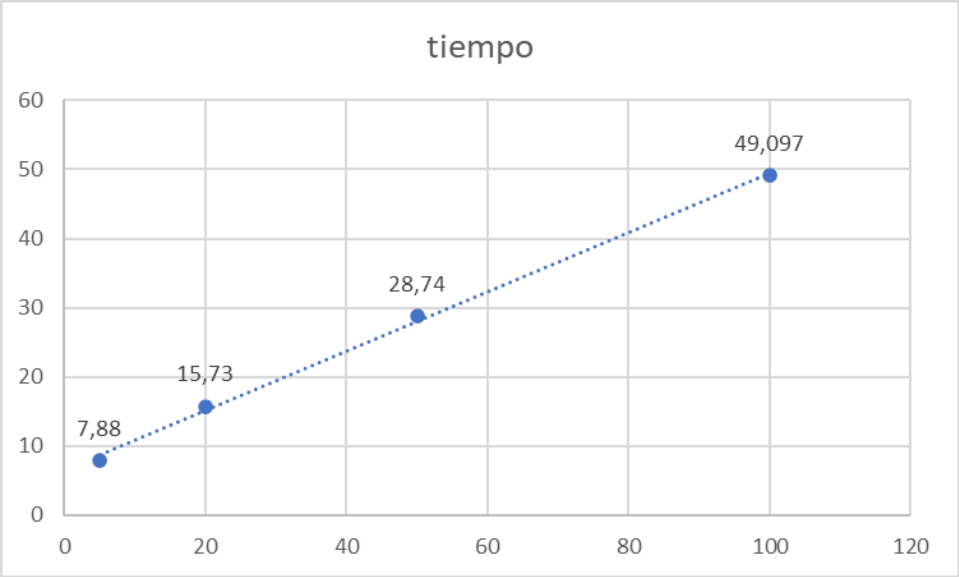
Se realizo con el año 2021 y en el sector 3

Entrada	**Memoria utilizada**
5 pct	13.68
20 pct	13.6875
50 pct	13.68
large	13.6875

Entrada	**Tiempo (ms)**
5 pct	7.88
20 pct	15.73
50 pct	28.74
large	49.097

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 complejidad espacial de este requerimiento consiste en o (n), porque todo el tiempo se hace una búsqueda de los mismos datos una vez, incluso a la vez que va subdividiéndose, se hace una búsqueda por menor cantidad de datos porque se va dividiendo, de esa manera el mayor recorrido se puede evidenciar en el primer caso se divide por subsectores en cada año. A diferencia del reto 1, en este utilizamos los mapas en hacer la organización de los diccionarios, facilitándonos la hora de ir en llave en llave con el keyset, o la velocidad de devolver una llave también nos lo facilito

Requerimiento 3

Descripción

```python
def req_3(data_structs,anio):
    """
    Función que soluciona el requerimiento 3
    """
    anio_llave = int(anio)


    mapa_anios = data_structs['Años']



    array_anio = devolver_value(mapa_anios,anio_llave)['Lista']

    lista_subsects_anio = crear_lista_subsectores_por_anio(array_anio)

        ##encuentra menor subsect por retenciones
    menor = encontrar_menor(lista_subsects_anio, 'Total retenciones')
    agregar_lista_de_6_a_subsector(menor,array_anio)

    return menor
```

Este requerimiento se encarga de encontrar el subsector económico que tuvo el menor total de retenciones (Total retenciones) para un año especifico. Lo primero que hace es obtener el array que contiene todas las actividades del año dado. A partir de este array, crea una lista de subsectores totalizados mediante una iteración para luego encontrar el menor subsector por total de retenciones mediante otra iteración. Por último, añade una lista con los primeros y últimos 3 impuestos del subsector a este, mediante un llamamiento de elementos sobre una lista ordenada. Así, la función retorna el subsector menor con su respectiva lista de 3 primeros y últimos.

Entrada	Estructuras de datos del modelo, año.
Salidas	El mayor subsector (incluye la lista de 6 actividades)
Implementado (Sí/No)	Si. Implementado por Samuel Santiago Peña Ricaurte

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Encontrar array del año.	O(1)
Crea lista totalizada de subsectores (iteración)	O(n)
Encontrar menor subsector por retenciones (iteración)	O(n)
Crear y añadir lista de 6 actividades a subsector (iteración y ordenar lista acotada)	O(nlog(n))
Retorno	O(1)
Total	***O(nlog(n)***

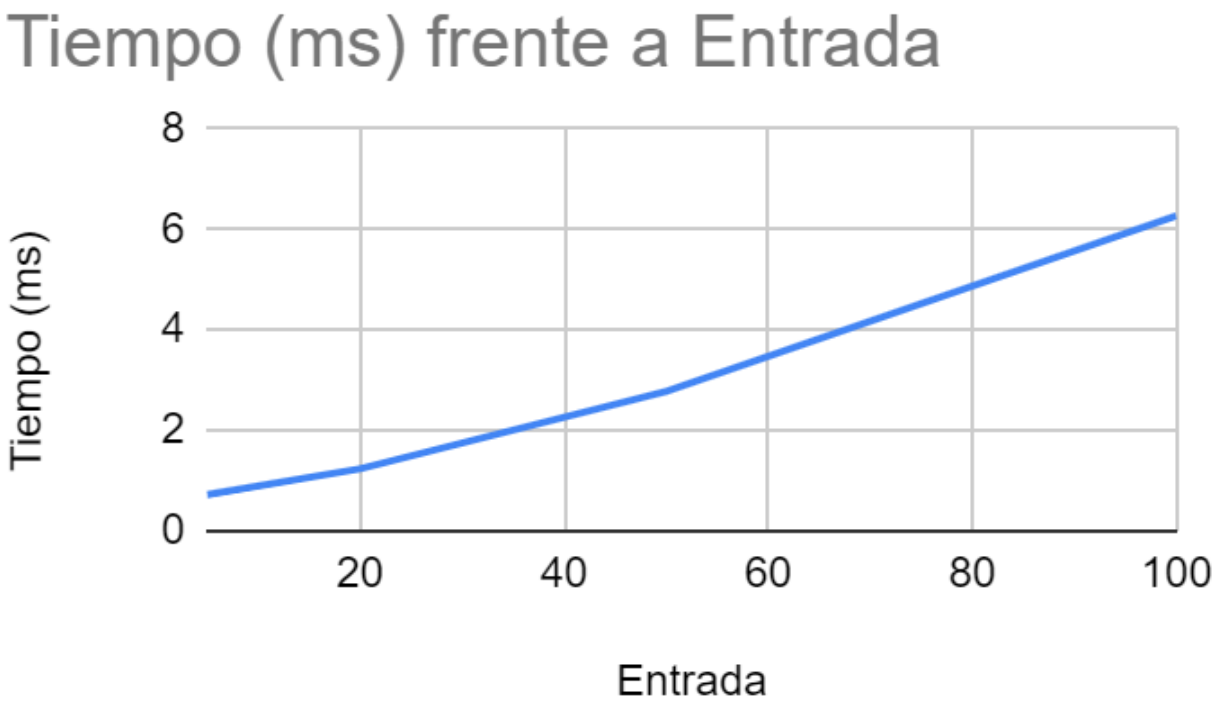
Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el año 2013, el tipo chaining y el factor 4.

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

Entrada	**Tiempo (ms)**
5 pct	0,7096000909805298
20 pct	1,2290000915527344
50 pct	2,7728999853134155
large	6,269199848175049

Graficas



Análisis

A pesar de que en términos de notación O el requerimiento tiene complejidad O(nlog(n)), pues la función agregar_lista_6_subsector toma un array y lo ordena y la complejidad de eso es O(nlog(n)) en Quick sort, como muestra la gráfica parece ser que la complejidad real tiende a ser lineal, es decir, de orden n. Esto tiene sentido si se considera que el quick sort se aplica sobre una lista de actividades de un subsector, muy inferior en número a la lista de actividades por año sobre la que se hacen las iteraciones de orden n. Así, la complejidad de estas iteraciones pesa más en la realidad en la complejidad.

Chaining 4 2021

Entrada	**Tiempo (ms)**
5 pct	1.065000057220459
20 pct	2.159999966621399
50 pct	4.121500015258789
large	5.623900055885315

Requerimiento 4

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

Descripción

```
def req_4(data_structs, anio):
    """
    Función que soluciona el requerimiento 4
    """

    # TODO: Realizar el requerimiento 4
    #SUBSECTOR ECONOMICO con mayores COSTOS Y GASTOS DE NOMINA dado un AÑO
    dico = organizar_sector(data_structs, "Código subsector económico")
    dico_anio = devolver_value(dico, anio)
    llaves = mp.keySet(dico_anio)
    numero = 0
    for numero in range(lt.size(llaves)):
        llave = llaves[numero]
        if llave == "Código subsector económico":
            sub_b = devolver_value(dico_anio, llave)
        numero += 1
    mayor = encontrar_mayor_criterio(sub_b , "Total costos y gastos nómina")
    sector_mayor = lt.getElement(llaves,mayor[1])
    return sector_mayor
```

Este requerimiento lo que busca es organizar los datos por años y luego por el sector. De esta forma podemos indagar en el año seleccionado sobre los datos que queremos analizar, principalmente sobre los subsectores y sus costos y gastos de nomina

Entrada	Data_structs: todos los datos cargados Anio: el año en el que se busca el mayor total cosotos y gastos de nomina
Salidas	Devuelve el diccionario del mayor total saldo a favor del tiempo dado, con las partes que el requerimiento pide

Implementado (Sí/No)	Si. Por Manuel Pinzon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Pasos	**Complejidad**
Organizar los datos por codigo	O(n) pasa por todos los datos
Buscar el año que queremos en los datos	O(n) encuentra el valor en las llaves (menor el n en el que busca)
Buscar el sector que queremos estudiar	O(n) encuentra el sector especifico(menor n)
Organizar los datos por el criterio seleccionado	O(n) busca dentro de esta subseccion de datos
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).

Memoria RAM	**32 GB**
Sistema Operativo	Windows 11

Procesador : Intel i9 !2th

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

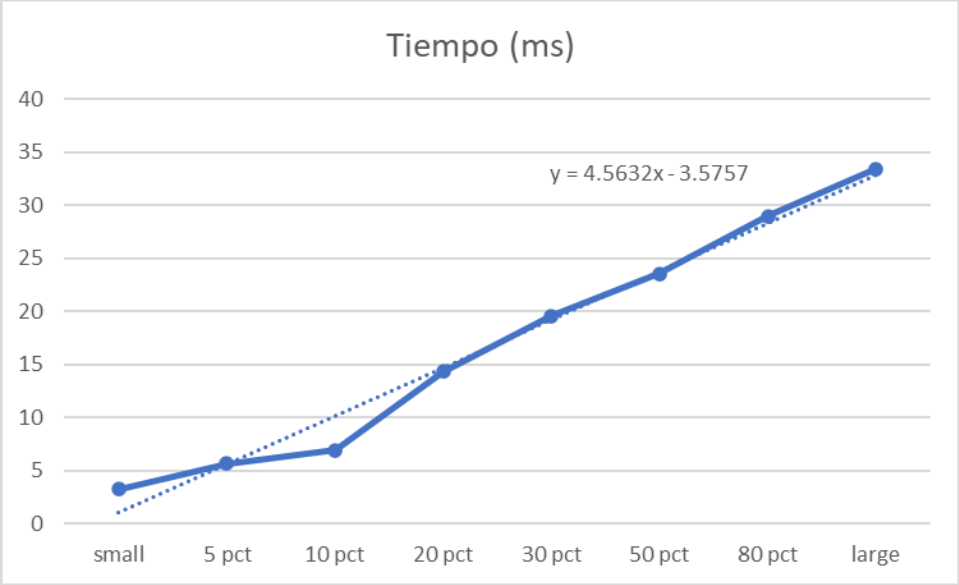
Se realizo con el año 2021 y en el sector 3

Entrada	**Memoria utilizada**
5 pct	16.66
20 pct	25.84
50 pct	49.63
large	73.52

Entrada	**Tiempo (ms)**
5 pct	3.26
20 pct	14.3
50 pct	23.56
large	33.46

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Podemos evidenciar que a diferencia del reto anterior se facilito mucho mas la carga y como se buscaban los datos, sin embargo a veces la complejidad de los datos hacia con se necesitara mucho mas codigo

Comparación Reto 1:

La complejidad en O es similar en ambos retos, por lo de organizar la lista. Sin embargo, los tiempos de ejecución son mayores en el reto 1, pues en este reto lo que se pedía en la pregunta no coincidía con lo que se esperaba, y se esperaba devolver una lista de subsectores, no un subsector único.

Requerimiento 5

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

Descripción

Entrada	Data_structs: todos los datos cargados Anio: el año en el que se busca el mayor saldo total de impuestos

Salidas	Devuelve una tupla de tres elementos (un dic con el sector que más aporto y la suma de lo que se pide, una lista de actividades del sector, el sector que más aporto)
Implementado (Sí/No)	Si. Por Tomas DIaz

```
def req_5(data_structs, anio):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    dic = organizar_sector(data_structs, "Código subsector económico" )
    #dic subdividido


    dic_del_anio = devolver_value(dic,int(anio))
    #dic del subsector en el año
    llaves = mp.keySet(dic_del_anio)

    i = 1
    sumas = lt.newList(cmpfunction="ARRAY_LIST")
    #estara la suma del descuento
    sectores = lt.newList()
    #estara cual sector (mismo orden al de arriba)

    while i<=lt.size(llaves):
        numero = 0
        pos = lt.getElement(llaves, i )
        # pos es igual a el numero del subsector

        lt.addLast(sectores, i)
        #posicion en la lista de llaves

        valor = devolver_value(dic_del_anio,pos)
        #lista de dicionarios
        numero = suma_variable(valor,"Descuentos tributarios" )
        #suma descuentos totales por subsector

        lt.addLast(sumas, numero)
        i+=1
```

```
    posicion = encontrar_mayor_list(sumas)
    #en que posicion de las llaves(subsector) esta el de mayores descuentos

    sector_mayor = lt.getElement(llaves,posicion[1])
    #nombre del subsector

    para_organizar = devolver_value(dic_del_anio, sector_mayor)
    #lista de los subsectores
    merg.sort(para_organizar,sort_criteria_descuentos_tributarios)
    #lista subsectores organizados de menor a mayor por descuento

    sacar_basicos = lt.getElement(para_organizar,1)
    #encontrar las cosas ue comparten todos



    respuesta = {"Código sector económico": sacar_basicos["Código sector económico"],
                 "Nombre sector económico":sacar_basicos["Nombre sector económico"] ,
                  "Código subsector económico": sacar_basicos["Código subsector económico"],
                   "Código subsector económico":sacar_basicos["Código subsector económico"],
                   "Descuentos tributarios": posicion[0]
                   }

    #diccionario con todo lo necesario
    codigos= [ "Total ingresos netos", "Total costos y gastos", "Total saldo a pagar", "Total saldo a favor" ]

    for cod in codigos:
        #sumar cada variable
        la_suma=suma_variable(para_organizar,cod)
        respuesta[cod] = la_suma

    heads = ["Código actividad económica", "Nombre actividad económica", "Código sector económico","Nombre sector económico",
             "Código subsector económico", 'Nombre subsector económico',"Descuentos tributarios", "Total ingresos netos", "Total costos y gastos",
             "Total saldo a pagar", "Total saldo a favor"]
    final = filtrar_lista_dics_por(para_organizar, heads)
    #asignarle diccionario a cada uno por lo que quiero que muestre
    return respuesta, final, sector_mayor
```

En este requerimiento lo que hice fue organizar los datos los años y después por el código del subsector económico, de aquí proseguí a sacar una lista de los subsectores, la cual usé para posteriormente sumar todos los descuentos tributarios por subsector y así poder cual fue el sector que mayor descuento tributario tubo. Después de encontrar dicho subsector, lo organice de mayor a menor. Por último, en ese mismo subsector sumes todas las variables las que necesitaban sumar y organizándolo a lo que quiero mostrar, devolviendo el subsector que más aporto y sus dichos valores.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: organizar sector (dic según sectores)	O(n) pasa por todos los datos
Paso 2: dic (según el año) = devolver_value(devolver año buscado)	O(n) encuentra el valor en las llaves (menor el n en el que busca)
Paso 3: llaves = keyset(los subsectores del año especifico)	O(n) encuentra las llaves del año en el que busca
Paso 4: sumas = new list (estarán las sumas de cada subsector)	O(1)

Paso 5: sectores = new list (estará cada sector en el mismo orden que en el de arriba)	O(1)
Paso 6 : while = se suman todos los descuentos en los subsectores, y se agregan en una lista el valor de la suma y el otro el subsector	O(n) va por los datos del año
Paso 7 : posición = encontrar mayor (busca cual es mayor de los descuentos y devuelve su posición y valor)	O(n) es un n pequeño porque es la cantidad de subsectores en el año
Paso 8 : sector_mayor = dentro de la lista de sectores busca la posición dada y este es el subsector mayor	O(n) es un n pequeño porque es la cantidad de subsectores en el año
Paso 9 : para_organizar = devuelve una lista de los id dentro del subsector con mayores descuentos	O(n) es la cantidad de id dentro del subsector mayor en el año
Paso 10: merg(organiza la lista según los descuentos)	O(nlogn) es la cantidad de id dentro del subsector mayor en el año
Paso 11: sacar_basicos (para devolver lo requerido, que comparten el mismo subsector)	O (1)
Paso 12: for = suma todas las variables en el subsector que estamos buscando	O(n) es la cantidad de id dentro del subsector mayor en el año
Paso 13: res = filtrar dic con llaves (organiza el dic, para lo que quiera devolver)	O(n)las llaves de un valor (n muy pequeño)
TOTAL	***O(nlogn)***

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).

Memoria RAM	**8 GB**
Sistema Operativo	Windows 10

Procesador : AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx 2.60 GHz

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

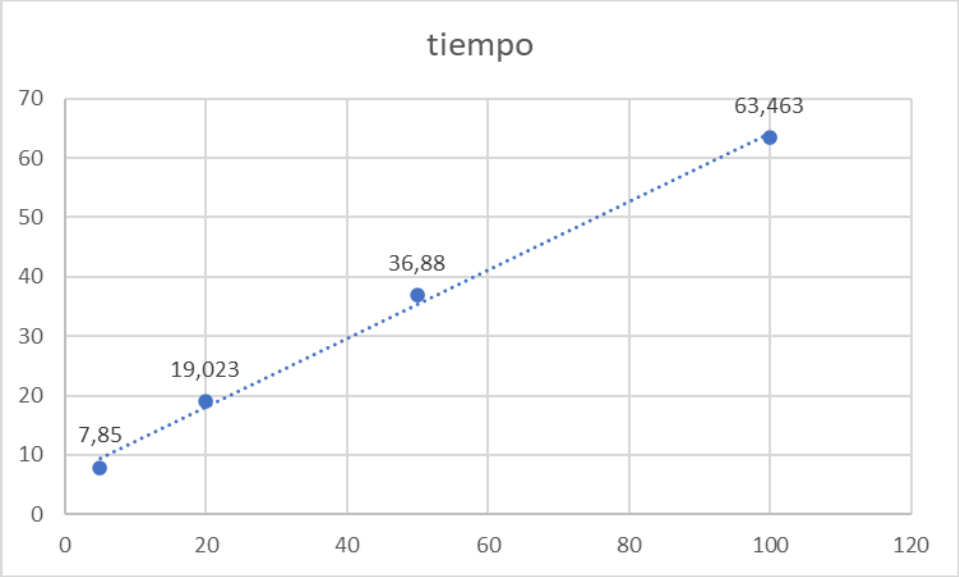
Se realizo con el año 2021

Entrada	**Memoria utilizada**
5 pct	18.03
20 pct	19.84375
50 pct	51.7656
large	83.921875

Entrada	Tiempo (ms)
5 pct	7.85
20 pct	19.023
50 pct	36.88
large	63.463

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 complejidad espacial de este requerimiento es de orden O(nlogn), ya que en todo momento se busca los mismos datos una sola vez, incluso cuando se van subdividiendo. A medida que se divide, la búsqueda se realiza en una cantidad menor de datos, lo que resulta en un menor recorrido. En el primer caso, la división por subsectores ocurre cada año, lo que representa el mayor recorrido. Lo que aumenta la complejidad de este caso es cuando se utiliza merge, porque este tiene una complejidad espacial de nlogn, a la hora de ordenar los datos de menor a mayor por los totales de descuentos, después prosigue a hacer la suma en estos sectores, haciendo un o(n). En comparación al reto uno, en este se facilitó la búsqueda de información dentro de los diccionarios, y también fue mucho más sencillo a la hora de organizarlo, pero se seguía trabajando con las listas, porque pasar de un id a otro es mucho más fácil, o el hecho de dar una posición exacta hace que sea mucho más eficaz el procedimiento.

Requerimiento 6

Descripción

```
def req_6(data_structs, anio):
    """
    Función que soluciona el requerimiento 6
    """
    anio = int(anio)
    map_anios = data_structs['Años']
    array_del_anio = devolver_value(map_anios,anio)['Lista']

    #### Crear map de actividades por subsector (llave subsector, valor array de actividades)


    ### crea lista totalizada de subsectores

    lista_subsectores = crear_lista_subsectores_por_anio(array_del_anio)


    ### Crea lista de sectores más general

    lista_sectores = crear_lista_sectores_totalizados_por_anio(lista_subsectores)


    #### Encontrar mayor sector

    mayor_sector = encontrar_mayor_con_condicion(lista_sectores,'Total ingresos netos')
    codigo_sector_mayor = mayor_sector['Código sector económico']

        ###Proceso con mayor

    mayor_subsector_para_sector_dado = encontrar_mayor_con_condicion(lista_subsectores,'Total ingresos netos','Código sector económico',codigo_sector_mayor)
    codigo_mayor_subsector = mayor_subsector_para_sector_dado['Código subsector económico']
    mayor_actividad_mayor_subsector = encontrar_mayor_con_condicion(array_del_anio,'Total ingresos netos','Código subsector económico',codigo_mayor_subsector)
    menor_actividad_mayor_subsector = encontrar_menor_con_condicion(array_del_anio,'Total ingresos netos','Código subsector económico',codigo_mayor_subsector)

        ## añadir mayor y menor actividad a mayor subsector

    mayor_subsector_para_sector_dado['Actividad que más contribuyó']= mayor_actividad_mayor_subsector
    mayor_subsector_para_sector_dado['Actividad que menos contribuyó']=menor_actividad_mayor_subsector

        ### añadir mayor subsector a sector dado

    mayor_sector['Subsector que más contribuyó'] = mayor_subsector_para_sector_dado

    ##### Proceso con menor
```

```
menor_subsector_para_sector_dado = encontrar_menor_con_condicion(lista_subsectores, 'Total ingresos netos','Código sector económico', codigo_sector_mayor)
codigo_menor_subsector = menor_subsector_para_sector_dado['Código subsector económico']
mayor_actividad_menor_subsector = encontrar_mayor_con_condicion(array_del_anio,'Total ingresos netos','Código subsector económico',codigo_menor_subsector)
menor_actividad_menor_subsector = encontrar_menor_con_condicion(array_del_anio,'Total ingresos netos','Código subsector económico',codigo_menor_subsector)

    ## Añadir mayor y menor actividad a menor subsector

menor_subsector_para_sector_dado['Actividad que más contribuyó']=mayor_actividad_menor_subsector
menor_subsector_para_sector_dado['Actividad que menos contribuyó']= menor_actividad_menor_subsector

    ### Añadir menor subsector a sector dado
mayor_sector['subsector que menos aportó'] = menor_subsector_para_sector_dado
return mayor_sector
```

Este requerimiento se encarga de encontrar el sector económico que tuvo el mayor total de ingresos netos para un año especifico. Lo primero que hace es obtener el array que contiene todas las actividades del año dado. A partir de este array, crea una lista de subsectores totalizados mediante una iteración y a partir de la dicha lista crea una totalizada de sectores mediante otra iteración para luego encontrar el mayor subsector por total de ingresos mediante otra iteración. Posteriormente, encuentra (mediante iteración condicionada de la lista de subsectores del año) el que más y el que menos aportó y añade a los dichos subsectores las actividades que más y menos les aportaron (mediante iteración condicionada

sobre la lista de actividades del año). Por último, añade los dichos subsectores al sector y retorna la información solicitada del sector en forma de diccionario.

Entrada	Estructuras de datos del modelo, año.
Salidas	El mayor sector por ingresos (los subsectores que mas y menos aportaron y estos a su vez contienen las actividades que más y menos les aportaron)
Implementado (Sí/No)	Si. Implementado por Samuel Santiago Peña Ricaurte

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Encontrar array de actividades del año.	O(1)
Crea lista totalizada de subsectores (iteración)	O(n)
Crea lista totalizada de sectores (iteración)	O(n)
Encuentra mayor sector en lista sectores por criterio de ingresos (iteración)	O(n)
Encuentra mayor y menor subsector del sector en lista de subsectores del año (iteración)	O(n)
Encuentra mayor y menor actividad por ingresos para ambos subsectores en array del año (iteración)	O(n)
Añade las dichas actividades a los dichos subsectores y los dichos subsectores al sector y retorna el sector	O(1)
Total	***O(n)***

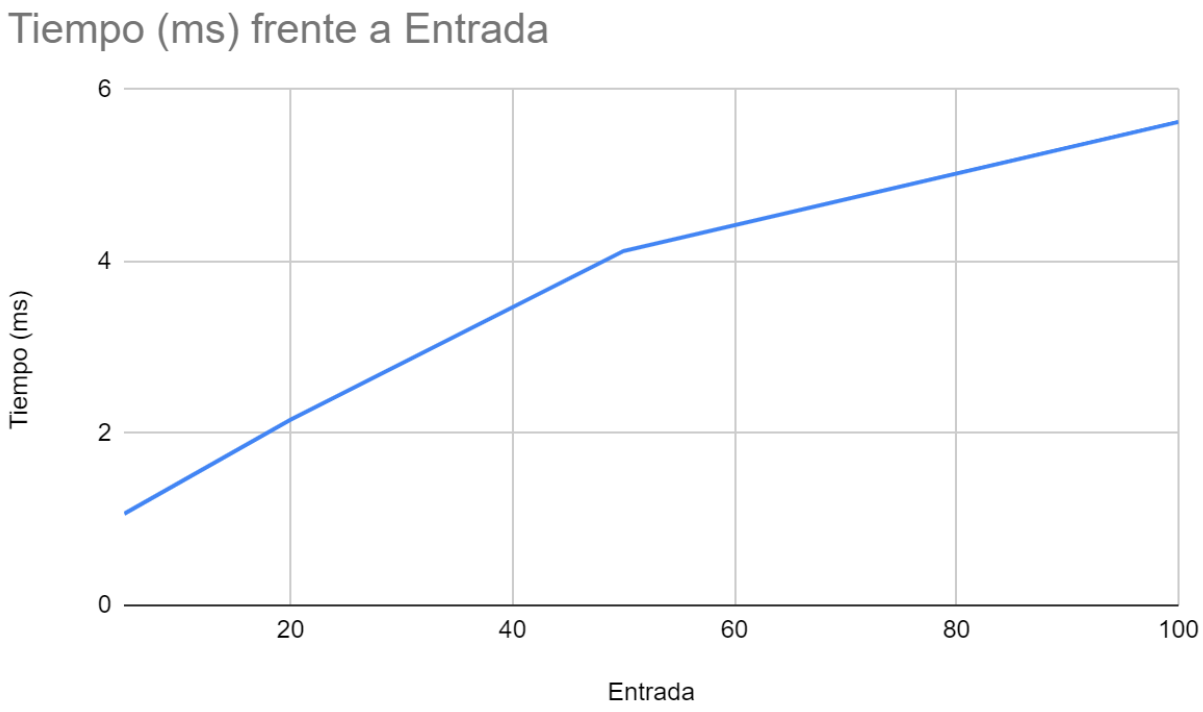
Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el año 2021, el tipo chaining y el factor 4.

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

Entrada	**Tiempo (ms)**
5 pct	1.065000057220459
20 pct	2.159999966621399
50 pct	4.121500015258789
large	5.623900055885315

Graficas



Análisis

A pesar de que en términos de notación O el requerimiento tiene complejidad O(n), pues todo se basa en iteraciones (no hay necesidad de ordenar), la grafica discrepa un poco de esto, ya que presenta cierta concavidad. Esto puede deberse tanto a variaciones aleatorias del procesador como a que el n que se toma para las pruebas es el total de datos, y este no cambia proporcionalmente con el n del año dado.

Comparación Reto 1:

La complejidad en O es de n cuadrada en reto 1 (mayor) pues se pedía devolver lista de sectores, y debía hacerse la iteración n veces. Los tiempos de ejecución son mayores también en el reto 1, pues en este reto lo que se pedía en la pregunta no coincidía con lo que se esperaba, y se esperaba devolver una lista de sectores, no un sector único.

Requerimiento 7

Descripción

Entrada	Data_structs: todos los datos cargados Anio: el año en el que se busca el mayor saldo total de impuestos Codigo: codigo del subsector económico que se desea buscar Num_actividades: número de actividades que identificar
Salidas	Devuelve una lista con la cantidad de actividades que se desea buscar
Implementado (Sí/No)	Si. Por Tomas DIaz

```
def req_7(data_structs, anio, codigo, num_actividades):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    dic = organizar_sector(data_structs, "Código subsector económico" )
    #dic subdividido


    dic_del_anio = devolver_value(dic,int(anio))
    #dic segun el año pedido
    esta = mp.contains(dic_del_anio,codigo)
    if esta:
        lista_sub_sector = devolver_value(dic_del_anio, str(codigo))
     #lista segun el codigo pedido

        merg.sort(lista_sub_sector, sort_criteria_total_costos_gastos)
     #organizado

        tamanio = lt.size(lista_sub_sector)
        respuesta = lt.newList(cmpfunction="ARRAY_LIST")

        if int(num_actividades) < tamanio:
            i =1
            while i <= int(num_actividades):
                valor = lt.getElement(lista_sub_sector,i)
                lt.addLast(respuesta,valor)
                i+=1
        else:
            respuesta = lista_sub_sector

        heads = ["Código actividad económica", "Nombre actividad económica", "Código sector económico","Nombre sector económico",
                 "Código subsector económico", 'Nombre subsector económico', "Total ingresos netos", "Total costos y gastos",
                 "Total saldo a pagar", "Total saldo a favor"]
        final = filtrar_lista_dics_por(respuesta,heads)
    else:
        final = "No hay ese subsector en el año indicado "
    return final
```

En este requerimiento lo que hice fue organizar los datos de los años y después por el código del subsector económico, después me aseguré si el código que se está buscando y si es así procede a organizar ese subsector de menor a mayor según los datos del total costos de cada id. Ahí evaluó si el número de actividades solicitadas es mayor al que presenta este subsector, si es así devuelvo las listas del subsector, de otra manera envió una lista con el número de actividades pedidas.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: organizar sector (dic según sectores)	O(n) pasa por todos los datos
Paso 2: dic (según el año) = devolver_value(devolver año buscado)	O(n) encuentra el valor en las llaves (menor el n en el que busca)
Paso 3: esta = reviso si el subsector se encuentra en este año	O(n) revisa en las llaves si se encuentra el subsector buscado
Paso 4: si es verdad: =	O(1)
Paso 5: lista_sub_sector = devolver value(devuelve la lista del subsector buscado, ósea los id)	O(N) n es la cantidad de subsectores que hay
Paso 6 : merg = organizo estos datos de menor a mayor por el total de costos y gastos	O(nlogn)
Paso 7 : tamanio = encuentro el tamaño de este subsector	O(1)
Paso 8 : respuesta = lista de donde va a ir la respuesta	O(1)
Paso 9 : if = si el tamanio es mayor a el número de actividades	O(1)
Paso 10: while= agrego a respuesta el número de actividades que me están pidiendo	O(n) es el número de actividades que se están pidiendo
Paso 11: si es falso = respuesta se vuelve los subsectores que están	O (1)
Paso 12: res = filtrar dic con llaves (organiza el dic, para lo que quiera devolver)	O(n)las llaves de un valor (n muy pequeño)
Paso 13: si esta no es verdadera, no hay subsector indicado	O(1)
TOTAL	***O(nlogn)***

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).

Memoria RAM	**8 GB**
Sistema Operativo	Windows 10

Procesador : AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx 2.60 GHz

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

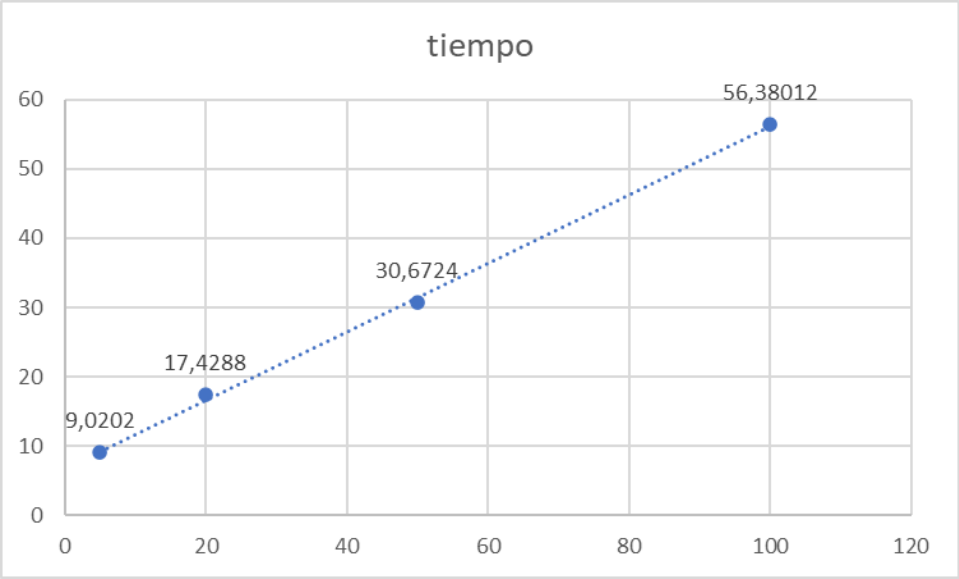
Se realizo con el año 2020 con el subsector 11 y con 9 actividades a mencionar

Entrada	Memoria utilizada
5 pct	11.14
20 pct	14.5156
50 pct	16.71875
large	15.9375

Entrada	Tiempo (ms)
5 pct	9.0202
20 pct	17.4288
50 pct	30.6724
large	56.38012

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 complejidad espacial de este requerimiento es de orden O(nlogn). Esto se debe a que se realiza una búsqueda de los mismos datos en todo momento, incluso al ir subdividiéndose, la búsqueda se realiza en una cantidad cada vez menor de datos debido a la subdivisión. El mayor recorrido se puede observar en el primer caso, donde se divide por subsectores en cada año, después se vuelve menor por cada subsector.

La complejidad se incrementa aún más cuando se utiliza la operación "merge", ya que esta tiene una complejidad espacial de O(nlogn) al ordenar los datos del subsector por el total de gastos, esta se le agrega una función de devolver una cantidad especifica, pero no afecta el tiempo, porque sigue siendo mayor o(nlogn). Adicionalmente, respecto al reto 1 hubo un cambio en la organización, cambiándolo por mapas haciendo mucho más fácil la organización y la velocidad en la que se encuentran las cosas. De resto, se mantuvo muy parecido al primer reto.