ANÁLISIS DEL RETO

Jorge Solórzano, 202115798, j.solorzanod@uniandes.edu.co

Juan Camilo Lyons Bustamante,201913100, jc.lyons@uniandes.edu.co

Juan Andrés Vargas Bolaños, 202110398, ja.vargasb1@uniandes.edu.co

Requerimiento 1

Descripción

```python
def req_1(data_structs):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    lista = lt.newList("ARRAY_LIST")
    param = ["Año","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"]
    lt.addLast(lista,param)

    highest = lt.firstElement(data_structs)
    for element in lt.iterator(data_structs):
        if element["Año"] != highest["Año"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,highest[i])
            lt.addLast(lista,lista_aux["elements"])
            highest = element
        elif float(element["Total saldo a pagar"]) > float(highest["Total saldo a pagar"]):
            highest = element
        elif element == lt.lastElement(data_structs):
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,highest[i])
            lt.addLast(lista,lista_aux["elements"])
    return lista["elements"]
```

En este requerimiento se busca listar la actividad económica con mayor total saldo a pagar para todos los años disponibles. Para esto se creó una lista vacía y se le añadió como primer elemento una lista con los parámetros que se quieren mostrar al final. La lógica de la implementación es la siguiente:

Se establece el primer elemento de la estructura de datos como el más “alto”. Se recorre la estructura de datos del modelo, que ya viene organizada por año de menor a mayor, buscando un elemento cuyo

“Total saldo a pagar” sea mayor al del “highest”. Si se encuentra se reemplaza el highest por este elemento. En caso de que el año del elemento sea distinto al del highest, esto quiere decir que ya se recorrieron todos los elementos del año, por lo que agrego a la lista todos los parámetros (elementos de la lista param”) que requiero del “highest” y establezco el nuevo elemento como el “highest”. Y se repite el ciclo.

Por último, cuando se haya llegado al ultima elemento de la estructura de datos del modelo, agrego al último elemento que quedó como “highest” a la lista. Al final debería quedarme una lista de listas donde la primera lista son los parámetros y las demás son los valores de estos parámetros para cada año disponible. Esto con el propósito de después usar la función tabulate para mostrarlo.

Entrada	Estructura de datos del modelo
Salidas	Lista de Listas
Implementado (Sí/No)	Si. Implementado de manera grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear una nueva lista	O(1)
Asignar un valor al parámetro “highest”	O(1)
Recorrer la estructura de datos del modelo	O(n)
Comparaciones	O(n)
Agregar un elemento a la lista	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).

Procesadores	**AMD Ryzen 5 4600H with Radeon Graphics 3.00GHz**
Memoria RAM	8,00 GB (7,36 GB utilizable)
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	0.63
5pct	0.75
10pct	1.04
20pct	1.52
30pct	2.29
50pct	3.58

80pct	6.21
large	7.56

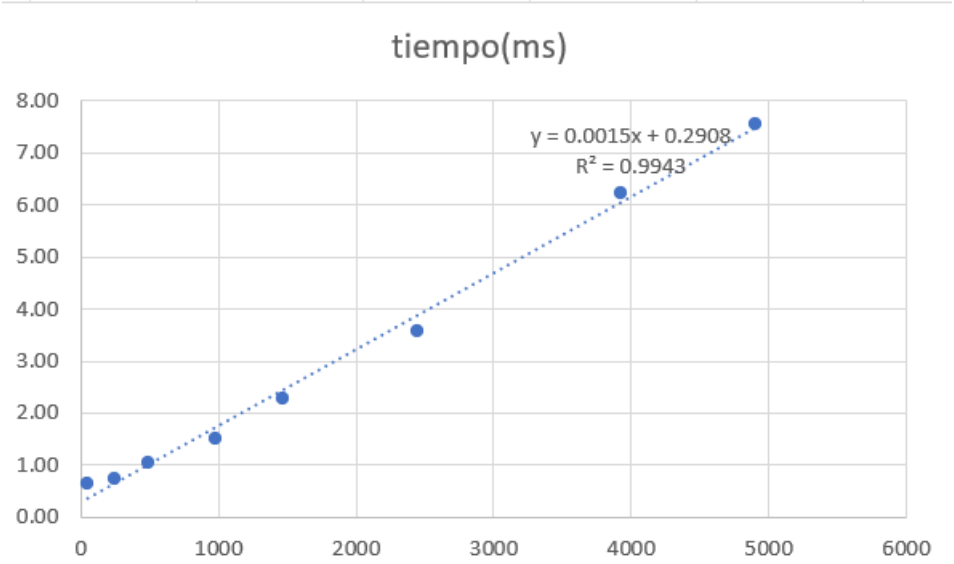
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Lista de Listas	0.63
5 pct	Lista de Listas	0.75
10 pct	Lista de Listas	1.04
20 pct	Lista de Listas	1.52
30 pct	Lista de Listas	2.29
50 pct	Lista de Listas	3.58
80 pct	Lista de Listas	6.21
large	Lista de Listas	7.56

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

Observamos que efectivamente la implementación es de orden lineal O(n), esto se evidencia en la gráfica al tener un R^2 muy cercano a 1. Esto es así ya que siempre se va tener que recorrer la estructura de datos entera y también se harán el mismo número de comparaciones que número de elementos en la lista. Su incremento será lineal.

Requerimiento 2

Descripción

```python
def req_2(data_structs):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    lista = lt.newList("ARRAY_LIST")

    param = ["Año","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"]
    lt.addLast(lista,param)

    highest = lt.firstElement(data_structs)
    for element in lt.iterator(data_structs):
        if element["Año"] != highest["Año"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,highest[i])
            lt.addLast(lista,lista_aux["elements"])
            highest = element
        elif float(element["Total saldo a favor"]) > float(highest["Total saldo a favor"]):
            highest = element
        elif element == lt.lastElement(data_structs):
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,highest[i])
            lt.addLast(lista,lista_aux["elements"])
    return lista["elements"]
```

En este requerimiento se busca listar la actividad económica con mayor total saldo a favor para todos los años disponibles. La implementación es prácticamente idéntica a la del requerimiento. La única diferencia es que la comparación se hace en base al “Total saldo a favor”.

Entrada	Estructura de datos del modelo
Salidas	Lista de Listas
Implementado (Sí/No)	Si. Implementado de manera grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear una nueva lista	O(1)
Asignar un valor al parámetro “highest”	O(1)
Recorrer la estructura de datos del modelo	O(n)
Comparaciones	O(n)
Agregar un elemento a la lista	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).

Procesadores	**AMD Ryzen 5 4600H with Radeon Graphics 3.00GHz**
Memoria RAM	8,00 GB (7,36 GB utilizable)
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	0.37
5pct	0.62
10pct	0.87
20pct	1.53
30pct	2.66
50pct	3.57
80pct	6.62
large	7.35

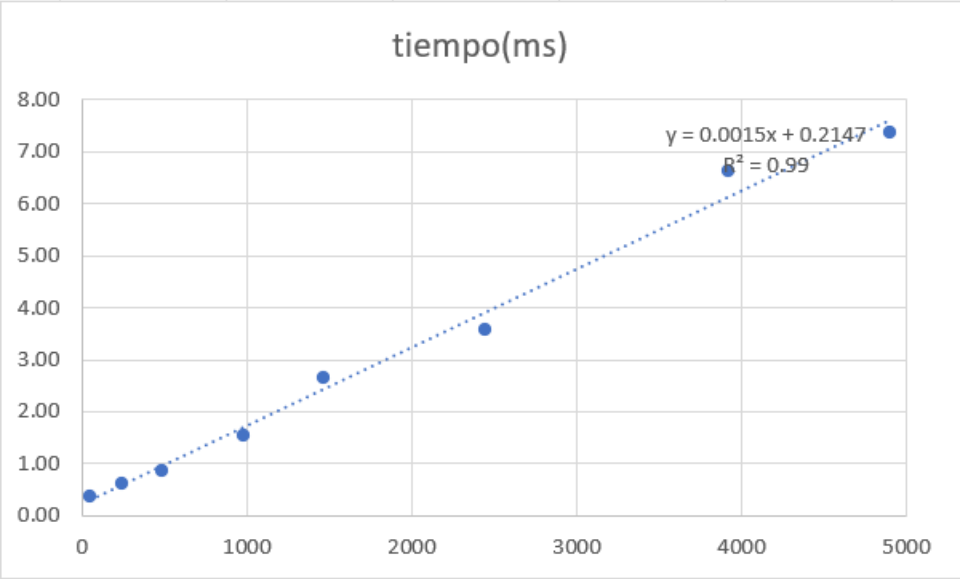
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Lista de Listas	0.37
5 pct	Lista de Listas	0.62
10 pct	Lista de Listas	0.87
20 pct	Lista de Listas	1.53
30 pct	Lista de Listas	2.66
50 pct	Lista de Listas	3.57
80 pct	Lista de Listas	6.62
large	Lista de Listas	7.35

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.

Este requerimiento presenta un comportamiento idéntico al del requerimiento anterior. El hecho de que se cambie el parámetro a comparar no afecta en nada a la complejidad de la implementación.

Requerimiento 3

Descripción

```python
def req_3(data_structs):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3

    #creo una copia del data_structs
    data_structs_aux = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs):
        data = {}
        for key,value in element.items():
            data[str(key)] = value
        lt.addLast(data_structs_aux,data)


    # Primera parte
    lista = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs_aux):
        if lt.isEmpty(lista):
            initial = lt.firstElement(data_structs_aux)
            lt.addLast(lista,initial)
        elif not lt.isEmpty(lista):
            if element["Código subsector económico"] == initial["Código subsector económico"]:
                initial["Total retenciones"] = str(int(initial["Total retenciones"])+int(element["Total retenciones"]))
                initial["Total ingresos netos"] = str(int(initial["Total ingresos netos"])+int(element["Total ingresos netos"]))
                initial["Total costos y gastos"] = str(int(initial["Total costos y gastos"])+int(element["Total costos y gastos"]))
                initial["Total saldo a pagar"] = str(int(initial["Total saldo a pagar"])+int(element["Total saldo a pagar"]))
                initial["Total saldo a favor"] = str(int(initial["Total saldo a favor"])+int(element["Total saldo a favor"]))
            elif element["Código subsector económico"] != initial["Código subsector económico"]:
                initial = element
                lt.addLast(lista,initial)
```

```
lista_min = lt.newList("ARRAY_LIST")
minimum = lt.firstElement(lista)
for element in lt.iterator(lista):
    if element == lt.lastElement(lista):
        if float(element["Total retenciones"]) < float(minimum["Total retenciones"]):
            minimum = element
        lt.addLast(lista_min,minimum)
    elif element["Año"] == minimum["Año"]:
        if float(element["Total retenciones"]) < float(minimum["Total retenciones"]):
            minimum = element
    elif not element["Año"] == minimum["Año"]:
        lt.addLast(lista_min,minimum)
        minimum = element

param = ["Año","Código sector económico","Nombre sector económico","Código subsector económico",
         "Nombre subsector económico","Total retenciones","Total ingresos netos","Total costos y gastos",
         "Total saldo a pagar","Total saldo a favor"]

lista_tablas_anios = lt.newList("ARRAY_LIST")
for minimum in lt.iterator(lista_min):
    lista = lista_por_anio_req3(data_structs,minimum)
    lt.addLast(lista_tablas_anios,lista)


lista_min_def = lt.newList("ARRAY_LIST")
for x in lista_min["elements"]:
    lista_min_aux = lt.newList("ARRAY_LIST")
    for i in param:
        lt.addLast(lista_min_aux,x[i])
    lt.addLast(lista_min_def,lista_min_aux["elements"])
```

```
param2 = ["Año","Código sector económico","Nombre sector económico","Código subsector económico",
          "Nombre subsector económico","Total retenciones del subsector económico",
          "Total ingresos netos del subsector económico","Total costos y gastos del subsector económico",
          "Total saldo a pagar del subsector económico","Total saldo a favor del subsector económico"]
lt.addFirst(lista_min_def,param2)

return lista_min_def["elements"], lista_tablas_anios["elements"]
```

```
def lista_por_anio_req3(data_structs,minimum):
    param = ["Código actividad económica","Nombre actividad económica","Total retenciones",
             "Total ingresos netos","Total costos y gastos","Total saldo a pagar","Total saldo a favor"]
    lista = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs):
        if element["Año"] == minimum["Año"]:
            if element["Código subsector económico"] == minimum["Código subsector económico"]:
                lt.addLast(lista,element)

    sorted_list = merg.sort(lista,cmp_total_retenciones)
    lista = lt.newList("ARRAY_LIST")
    lt.addLast(lista,param)
    if lt.size(sorted_list) > 6:
        lista_anio = first_and_last3(sorted_list)
        for x in lista_anio["elements"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,x[i])
            lt.addLast(lista,lista_aux["elements"])
    else:
        lista_anio = sorted_list
        for x in lista_anio["elements"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,x[i])
            lt.addLast(lista,lista_aux["elements"])
    return lista["elements"]
```

```
def cmp_total_retenciones(impuesto1, impuesto2):
    return float(impuesto1["Total retenciones"]) < float(impuesto2["Total retenciones"])
```

En este requerimiento se nos pide encontrar el subsector económico que tuvo el menor total de retenciones (“Total retenciones”) para cada año disponible en los datos cargados. Adicionalmente, se debe encontrar las tres actividades económicas que menos aportaron y las tres actividades económicas que más aportaron al valor total de retenciones en cada año. Dividí el requerimiento en varias secciones. Primero copio la estructura de datos del modelo para evitar modificar los datos al hacer las operaciones que requiero. Después creo una lista vacía y empiezo a recorre la copia de la estructura de datos del modelo. Es importante resaltar que la estructura ya viene organizada por año y actividad económica. Establezco el primero valor de la estructura como “initial”. Recorro la estructura buscando elementos que tengan el mismo subsector económico que el “initial”. Si lo encuentra sumo al total retenciones del “initial”, el total retenciones del elemento encontrado. Si tiene un código de subsector económico distinto quiere decir que ya se acabaron las actividades económicas para ese subsector económico, por lo que agrego “initial” a la lista que creé al inicio y establezco el nuevo elemento como el “initial”. Después vuelvo a regresar el recorrido. Al final de este recorrido me va a quedar una lista con todos los subsectores económicos organizados por año.

Ahora quiero sacar para cada año el subsector económico con el menor “Total retenciones”. Para esto, creo una nueva lista vacía y recorro la lista con todos los subsectores económicos organizados por año. Establezco el primer elemento como “mínimum” y avanzo buscando un subsector económico con un valor “Total retenciones” menor que el mínimum. Si lo encuentra, establezco este elemento como

mínimum. Cuando haya un elemento con un año distinto del mínimum quiere decir que ya recorrí todos los subsectores económicos para ese año, por lo que agrego el mínimum a la lista que creé al inicio del recorrido y establezco el elemento como mínimum. Y así hasta que acabe toda la lista con todos los subsectores económicos organizados por año.

Para mostrar el top 3 de menores y top 3 de mayores actividades económicas que aportaron al subsector económico, creó una nueva función ("lista_por_anio_req3") que me permita volver a recorrer la estructura de datos del modelo y sacar todas las actividades económicas que aportaron al subsector económico con menor "Total retenciones". Almaceno estas actividades en una lista y después las organizó por "Total retenciones".

Cuando ya tengo todas las actividades, saco en una nueva lista todos los parámetros con ayuda de una lista nativa de python "param" para mostrarlos después en el view. Por último cambió el nombre de los parámetros para que cuadren correctamente con los requeridos en el reto.

Al final, se devuelven dos listas de listas. Una con los subsectores económicos con menor "Total retenciones" para cada año y otra lista de listas de las 3 actividades económicas que más aportaron a ese subsector económico y las 3 actividades económicas que menos aportaron.

Entrada	Estructura de datos del modelo
Salidas	Una lista de listas con los subsectores económicos con menor "Total retenciones" para cada año y otra lista de listas las 3 actividades económicas que más aportaron a ese subsector económico y las 3 actividades económicas que menos aportaron.
Implementado (Sí/No)	Si. Implementado por Juan Andrés Vargas B.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Copia de la estructura de datos	O(n)
Creaciones listas (5*)	O(1) cada una
Recorrido estructura de datos (2)	O(n) cada uno
Recorrido listas (4*)	O(a) cada uno (a < n)
Ordenamiento de una lista por "merge_sort"	Despreciable por que el número de elementos es mucho menor a 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).

Procesadores	**AMD Ryzen 5 4600H with Radeon Graphics 3.00GHz**
Memoria RAM	8,00 GB (7,36 GB utilizable)

Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	1.60
5pct	4.79
10pct	9.04
20pct	17.59
30pct	25.29
50pct	43.96
80pct	67.90
large	87.55

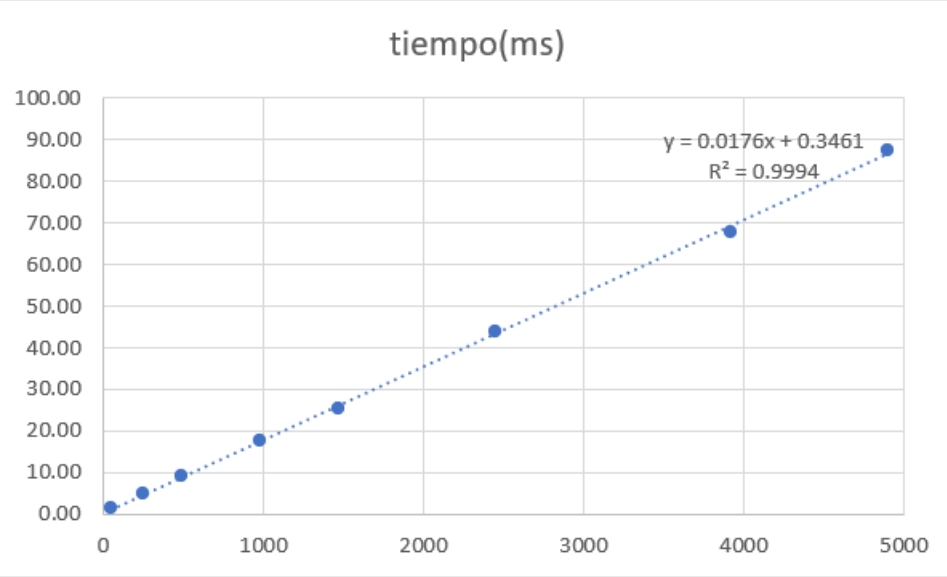
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Lista de Listas x 2	1.60
5 pct	Lista de Listas x 2	4.79
10 pct	Lista de Listas x 2	9.04
20 pct	Lista de Listas x 2	17.59
30 pct	Lista de Listas x 2	25.29
50 pct	Lista de Listas x 2	43.96
80 pct	Lista de Listas x 2	67.90
large	Lista de Listas x 2	87.55

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.

Observamos que la implementación es efectivamente lineal. Gracias a la gráfica observamos su comportamiento lineal. Esto tiene sentido con lo expresado anteriormente ya que siempre se tiene que recorrer la estructura de datos completa, este proceso tiene complejidad O(n). Aunque se haga mas de una vez esto no afectara su comportamiento. Los demás pasos son constantes o de complejidad menor a O(n) por lo que finalmente la complejidad total será O(n).

Requerimiento 4

Descripción

```
def req_4(data_structs):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    lista = lt.newList("ARRAY_LIST")

    titulo = ["Año", "Código sector económico","Nombre sector económico","Código subsector económico",
            "Nombre subsector económico","Costos y gastos nómina", "Total ingresos netos","Total costos y gastos",
            "Total saldo a pagar","Total saldo a favor"]
    lt.addLast(lista,titulo)

    highest = lt.firstElement(data_structs)
    for element in lt.iterator(data_structs):
        if element["Año"] != highest["Año"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in titulo:
                lt.addLast(lista_aux,highest[i])
            lt.addLast(lista,lista_aux["elements"])
            highest = element
        elif float(element["Costos y gastos nómina"]) > float(highest["Costos y gastos nómina"]):
            highest = element
        elif element == lt.lastElement(data_structs):
            lista_aux = lt.newList("ARRAY_LIST")
            for i in titulo:
                lt.addLast(lista_aux,highest[i])
            lt.addLast(lista,lista_aux["elements"])
    lt.removeFirst(lista)
    tit= ["Año","Código sector económico","Nombre sector económico","Código subsector económico","Nombre subsector económico",
         "Total de costos y gastos nómina del subsector económico","Total ingresos netos del subsector economico","Total costos y gastos del subsector económico",
         "Total saldo a pagar del subsector económico","Total saldo a favor del subsector económico"]
    lt.addFirst(lista,tit)

    anios = ["2012","2013","2014","2015","2016","2017","2018","2019","2020","2021"]
    lista_anios = lt.newList("ARRAY_LIST")
    for anio in anios:
        lista2 = lista_anio_req4(data_structs,anio)
        lt.addLast(lista_anios,lista2)
    return lista["elements"], lista_anios["elements"]
```

```
def lista_anio_req4(data_structs,anio):
    tit = ["Código actividad económica","Nombre actividad económica","Costos y gastos nómina",
           "Total ingresos netos","Total costos y gastos","Total saldo a pagar","Total saldo a favor"]
    lista = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs):
        if element["Año"] == anio:
            lt.addLast(lista,element)

    sorted_list = merg.sort(lista,cmp_total_retenciones)
    lista = lt.newList("ARRAY_LIST")
    lt.addLast(lista,tit)
    if lt.size(sorted_list) > 6:
        lista_anio = first_and_last3(sorted_list)
        for x in lista_anio["elements"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in tit:
                lt.addLast(lista_aux,x[i])
            lt.addLast(lista,lista_aux["elements"])
    else:
        lista_anio = sorted_list
        for x in lista_anio["elements"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in tit:
                lt.addLast(lista_aux,x[i])
            lt.addLast(lista,lista_aux["elements"])
    return lista["elements"]
```

```
def cmp_gastos_nomina(gasto_1, gasto_2):
    return float(gasto_1["Costos y gastos nómina"]) > float(gasto_2["Costos y gastos nómina"])
```

En este requerimiento busca listar la actividad económica con mayor total costo y gasto de nómina de todos los años disponibles. Para esto se creó una lista vacía y se le añadió como primer elemento una lista con los parámetros que se quieren mostrar al final. La lógica de la implementación es la siguiente:

Se establece el primer elemento de la estructura de datos como el más “alto”. Se recorre la estructura de datos del modelo, que ya viene organizada por año de menor a mayor, buscando un elemento cuyo “Total Costo y Gastos nómina” sea mayor al del “highest”. Si se encuentra se reemplaza el highest por este elemento. En caso de que el año del elemento sea distinto al del highest, esto quiere decir que ya se recorrieron todos los elementos del año, por lo que agrego a la lista todos los parámetros dados por la lista “títulos”.

Luego, cuando se haya llegado al ultima elemento de la estructura de datos del modelo, agrego al último elemento que quedó como “highest” a la lista. Para las listas por años se creó otra función que recibe como parámetro el año y el data estructure para crear y organizar una lista de listas con los datos solicitados usando una función de cpm de 1 sola línea y lo retorne a la función principal. Al final debería quedarme una lista de listas donde la primera lista son los parámetros y las demás son los valores de estos parámetros para cada año disponible. Esto con el propósito de después usar la función tabulate para mostrarlo.

Entrada	Estructura de datos organizada del modelo
Salidas	Una lista de listas con los subsectores económicos con mayor “Total costo y gastos Nómina” para cada año y otra lista de listas las 3 actividades económicas que más aportaron a ese subsector económico y las 3 actividades económicas que menos aportaron.
Implementado (Sí/No)	Si. Implementado por Jorge Solórzano Diaz.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Copia de la estructura de datos	O(n)
Creaciones listas (5*)	O(1) cada una
Recorrido estructura de datos (2)	O(n) cada uno
Recorrido listas (6)	O(a) cada uno (a < n)
Ordenamiento de una lista por “merge_sort”	Constante menor o igual a 6 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).

Procesadores	**AMD Ryzen 5 3400G with Radeon Vega Graphics 3.70 GHz**

Memoria RAM	16,0 GB (13,9 GB usable)
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	2.68
5pct	7.32
10pct	10.21
20pct	23.16
30pct	35.34
50pct	56.26
80pct	79.40
large	92.46

Tablas de datos

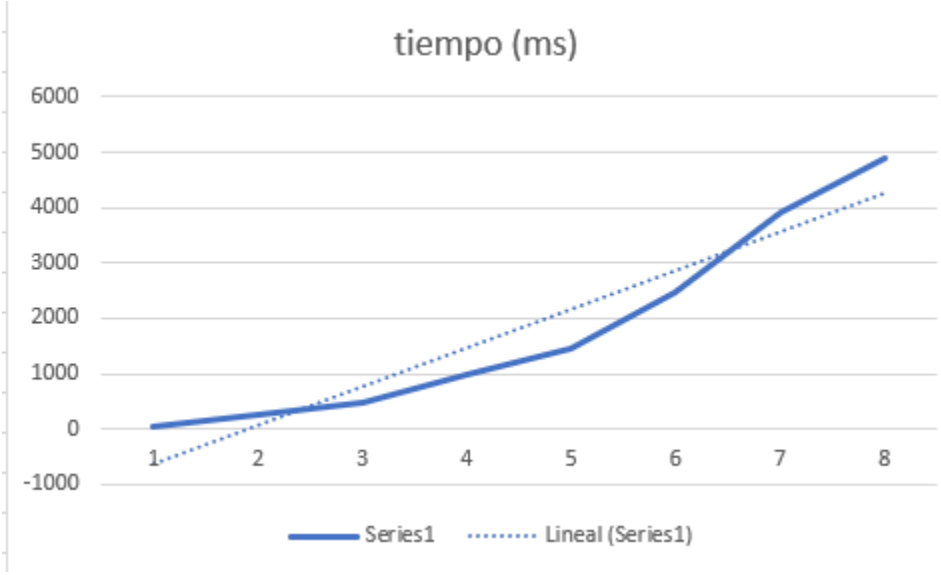
Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Lista de listas de listas de listas	2.68
5 pct	Lista de listas de listas de listas	7.32
10 pct	Lista de listas de listas de listas	10.21
20 pct	Lista de listas de listas de listas	23.16
30 pct	Lista de listas de listas de listas	35.34
50 pct	Lista de listas de listas de listas	56.26
80 pct	Lista de listas de listas	79.40

	de listas	
large	Lista de listas de listas de listas	92.46

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

Observamos que la implementación tiende a ser lineal. Gracias a la gráfica observamos su comportamiento. Esto tiene sentido con lo expresado anteriormente ya que siempre se tiene que recorrer la estructura de datos completa, este proceso tiene complejidad O(n). Aunque se haga más de n veces esto no afectara su comportamiento. Los demás pasos son constantes o de complejidad menor a O(n) por lo que tenemos como complejidad total O(n).

Requerimiento 5

Descripción

```
def req_5(data_structs):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    param2 = ["Año","Código sector económico","Nombre sector económico","Código subsector económico",
              "Nombre subsector económico","Total ingresos netos","Total costos y gastos","Descuentos tributarios",
              "Total saldo a pagar","Total saldo a favor"]

    param3 = lt.newList('ARRAY_LIST')
    for i in param2:
        lt.addLast(param3,i)

    lst = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs):
        l = lt.newList("ARRAY_LIST")
        for i in element.keys():
            if i in param2:
                lt.addLast(l,element[i])
        if lt.isEmpty(lst):
            lt.addLast(lst,l)
        elif not lt.isEmpty(lst):
            if lt.getElement(l,4) == lt.getElement(lt.lastElement(lst),4):
                Des = str(int(lt.getElement(lt.lastElement(lst),6))+int(lt.getElement(l,6)))
                ing = str(int(lt.getElement(lt.lastElement(lst),7))+int(lt.getElement(l,7)))
                cos_y_gas = str(int(lt.getElement(lt.lastElement(lst),8))+int(lt.getElement(l,8)))
                sal_a_pa = str(int(lt.getElement(lt.lastElement(lst),9))+int(lt.getElement(l,9)))
                sal_a_fa = str(int(lt.getElement(lt.lastElement(lst),10))+int(lt.getElement(l,10)))
                lt.changeInfo(lt.lastElement(lst),6,Des)
                lt.changeInfo(lt.lastElement(lst),7,ing)
                lt.changeInfo(lt.lastElement(lst),8,cos_y_gas)
                lt.changeInfo(lt.lastElement(lst),9,sal_a_pa)
                lt.changeInfo(lt.lastElement(lst),10,sal_a_fa)
            elif lt.getElement(l,4) != lt.getElement(lt.lastElement(lst),4):
                lt.addLast(lst,l)
```

```
    lista_max = lt.newList("ARRAY_LIST")
    maximum = lt.firstElement(lst)
    for element in lt.iterator(lst):
        if element == lt.lastElement(lst):
            if float(lt.getElement(element,8)) > float(lt.getElement(maximum,8)):
                maximum = element
            lt.addLast(lista_max,maximum)
        elif lt.getElement(element,1) == lt.getElement(maximum,1):
            if float(lt.getElement(element,8)) > float(lt.getElement(maximum,8)):
                maximum = element
        elif not lt.getElement(element,1) == lt.getElement(maximum,1):
            lt.addLast(lista_max,maximum)
            maximum = element


    lt.addFirst(lista_max,param3)

    lista_tablas_anios = lt.newList("ARRAY_LIST")
    for i in lt.iterator(lista_max):
        if lt.getElement(i,1) != "Año":
            lista = lista_por_anio_req5(data_structs,lt.getElement(i,1),lt.getElement(i,4))
            lt.addLast(lista_tablas_anios,lista)

    return lista_max["elements"], lista_tablas_anios["elements"]
```

```
def lista_por_anio_req5(data_structs,anio,codigo_subsector):
    param = ["Código actividad económica","Nombre actividad económica","Descuentos tributarios",
             "Total ingresos netos","Total costos y gastos","Total saldo a pagar","Total saldo a favor"]
    lista = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs):
        if element["Año"] == anio and element["Código subsector económico"] == codigo_subsector:
            lt.addLast(lista,element)

    sorted_list = merg.sort(lista,cmp_descuentos_tributarios)
    lista = lt.newList("ARRAY_LIST")
    lt.addLast(lista,param)
    if lt.size(sorted_list) > 6:
        lista_anio = first_and_last3(sorted_list)
        for x in lista_anio["elements"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,x[i])
            lt.addLast(lista,lista_aux["elements"])
    else:
        lista_anio = sorted_list
        for x in lista_anio["elements"]:
            lista_aux = lt.newList("ARRAY_LIST")
            for i in param:
                lt.addLast(lista_aux,x[i])
            lt.addLast(lista,lista_aux["elements"])
    return lista["elements"]
```

Para el requerimiento se tienen cuenta el nombre de los encabezados que se necesitan como son los años, entre otros. Se crea una lista para almacenar los valores de estos encabezados para cada uno de los subsectores. Se toma la lista contenida en el modelo donde están todos los datos y se recorre. Para cada uno de estos se miran solo los valores de los encabezados mencionados anteriormente. Si la lista donde se guardan los valores de los parámetros está vacía simplemente se agrega los valores de los parámetros. Si la lista no está vacía se mira si tienen el mismo subsector si lo tiene se suman los parámetros de total ingresos netos, total costos y gastos, descuentos tributarios entre otros si no están en el mismo subsector se agrega al final de la lista todos los parámetros. Luego, mediante una lista se toman por cada año aquellos subsectores que tienen el mayor descuento. Posteriormente, se mira por año las actividades económicas que más y menos aportaron. Se itera y se toma por cada año y subsector las actividades con mayor descuento mediante la última función. La última función llamada lista_por_anio_req5 se encarga de recibir la estructura de datos y determinar si son del año y la actividad económica deseada para posteriormente ordenarlas teniendo en cuenta el parámetro de descuentos tributarios. Si hay más de 6 actividades económicas se hayan las tres primeras y las tres últimas actividades económicas, es decir, las tres que más y menos dieron. Luego, se sacan solo los parámetros deseados (actividad, código, descuentos, ingresos, etc) y se agregan a una lista. Algo similar ocurre cuando se tienen menos de 6 actividades. En esta se obtiene las actividades económicas ordenadas y se obtienen los parámetros deseados para esto. Finalmente, esta lista es lo que retorna la función. La función principal retorna el valor retornado por la función anterior para cada año y los subsectores que tienen mayor descuento tributario para cada año.

Entrada	Estructuras de datos del modelo.
Salidas	Retorna los subsectores que tienen mayor descuento tributario por cada año y las actividades económicas que más y menos descuento tributario tienen para cada año y cada subsector

Implementado (Sí/No)	Si. Implementado por Juan Camilo Lyons Bustamante.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Copia de la estructura de datos	O(n)
Creaciones listas	O(1)
Recorrido estructura de datos	O(n)
Recorrido listas	O(n_1,n_2)
Ordenamiento de una lista por “merge_sort”	Despreciable por que el número de elementos es mucho menor a n.
TOTAL	***O(n)***

Se debe tener en cuenta que n_1 y n_2 es menor que n.

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 2.

Procesadores	**Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz**
Memoria RAM	8.00 GB (7.85 GB usable)
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	2.07
5 pct	11.11
10 pct	18.58
20 pct	39.52
30 pct	62.76
50 pct	95.96
80 pct	148.23
large	197.19

Tablas de datos

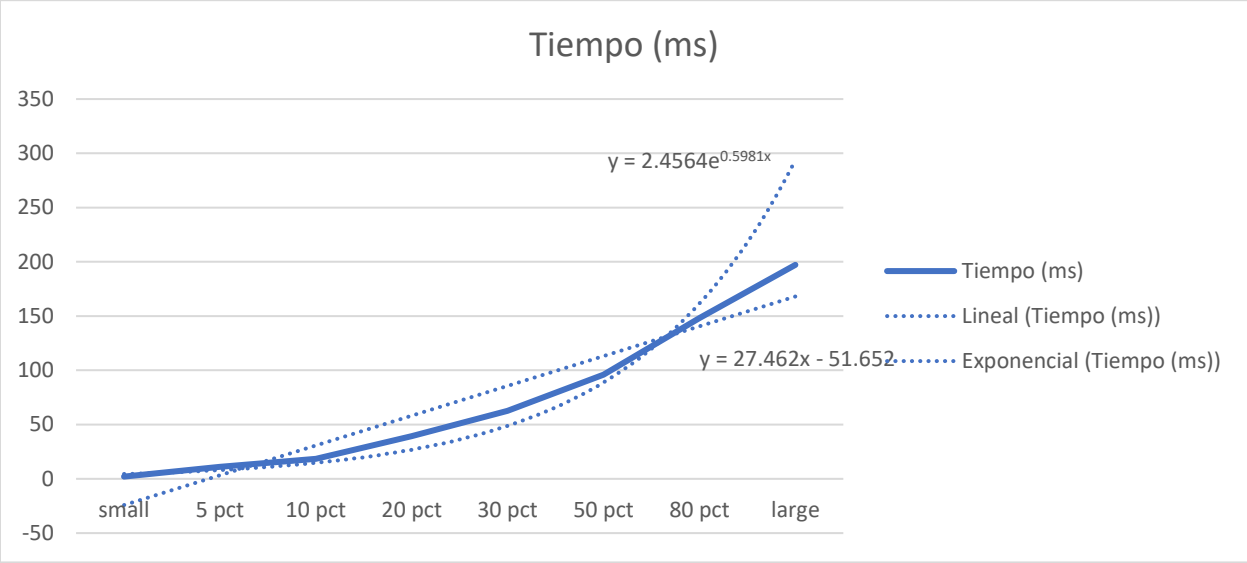
Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	2.07
5 pct	Dato2	11.11
10 pct	Dato3	18.58
20 pct	Dato4	39.52
30 pct	Dato5	62.76

50 pct	Dato6	95.96
80 pct	Dato7	148.23
large	Dato8	197.19

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *ArrayList*, la implementación de este requerimiento no tiene un orden de complejidad lineal como se puede observar en la gráfica. Esto debido a que se hacen muchas comparaciones con iteraciones. 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.

Requerimiento 6

No se realizo

Requerimiento 7

Descripción

```
def req_7(data_structs,topN,anio_inicial,anio_final):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    #tomo los datos pertenecientes al rango de años
    lista = lt.newList("ARRAY_LIST")
    for element in lt.iterator(data_structs):
        if (int(element["Año"]) >= int(anio_inicial)) and (int(element["Año"])<= int(anio_final)):
            lt.addLast(lista,element)

    sorted_lista_min = merg.sort(lista,cmp_total_costos_gastos)
    lista_min = lt.subList(sorted_lista_min,1,int(topN))

    lista_min = ins.sort(lista_min,cmp_impuestos_anio_total_costos_gastos)
    param = ["Año","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"]
    param2 = ["Año","Código actividad económica","Nombre actividad económica",
             "Código sector económico","Nombre sector económico","Código subsector económico",
             "Nombre subsector económico","Total ingresos netos consolidados para el periodo",
             "Total costos y gastos consolidados para el periodo",
             "Total saldo a pagar consolidados para el periodo",
             "Total saldo a favor consolidados para el periodo"]

    lista_min_tabulate = lt.newList("ARRAY_LIST")
    lt.addLast(lista_min_tabulate,param2)
    for element in lt.iterator(lista_min):
        lista_aux = lt.newList("ARRAY_LIST")
        for i in param:
            lt.addLast(lista_aux,element[i])
        lt.addLast(lista_min_tabulate,lista_aux["elements"])

    return lista_min_tabulate["elements"]
```

```
def cmp_total_costos_gastos(impuesto1, impuesto2):
    return float(impuesto1["Total costos y gastos"]) < float(impuesto2["Total costos y gastos"])
```

```
def cmp_impuestos_anio_total_costos_gastos(impuesto1, impuesto2):
    if float(impuesto1["Año"]) == float(impuesto2['Año']):
        cod1 = impuesto1['Total costos y gastos']
        cod2 = impuesto2['Total costos y gastos']
        return float(cod1) <= float(cod2)
    else:
        return (float(impuesto1['Año']) > float(impuesto2['Año']))
```

En este requerimiento nos piden listar el TOP (N) de las actividades económicas con el menor total de costos y gastos para un periodo de tiempo. Para esto, creo una lista y recorro la estructura de datos sacando solo los elementos en el rango de años dado como entrada. Después sorteo esta lista en función del “Total costos y gastos”. Después creo una sublista con el topN menor de datos. Después esta sublista de N datos la sorteo por año y por “Total costos y gastos” esta vez de mayor a menor.

Por último, sacó en una lista nueva los parámetros, con ayuda de la lista param, que requiero mostrar en el view en forma de lista. Agrego al inicio la lista los parámetros que voy a tener como cabecera para el tabulate(param2). Al final voy a tener una lista de listas donde la primera lista son los parámetros que voy a mostrar y las demás son los valores de estos parámetros para el topN sorteado previamente.

Entrada	Estructura de datos del modelo, El número (N) de actividades económicas a identificar, Año inicial, Año final
Salidas	Lista de listas con los parámetros como cabecera y sus valores para cada elemento del topN.
Implementado (Sí/No)	Si. Implementado por Juan Andrés Vargas B.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido estructura de datos	O(n)
Creación lista (3)	O(1) cada una
Sorteamiento de listas	O(N) para N <n (N << n para un n grande)
Recorrido de lista	O(a) para a<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).

Procesadores	**AMD Ryzen 5 4600H with Radeon Graphics 3.00GHz**
Memoria RAM	8,00 GB (7,36 GB utilizable)
Sistema Operativo	Windows 11

Top 5 para rango de 2010 hasta 2021 (Rango mayor)

Entrada	**Tiempo (ms)**
small	0.79
5pct	3.92
10pct	8.24

20pct	18.04
30pct	29.60
50pct	49.78
80pct	85.70
large	111.14

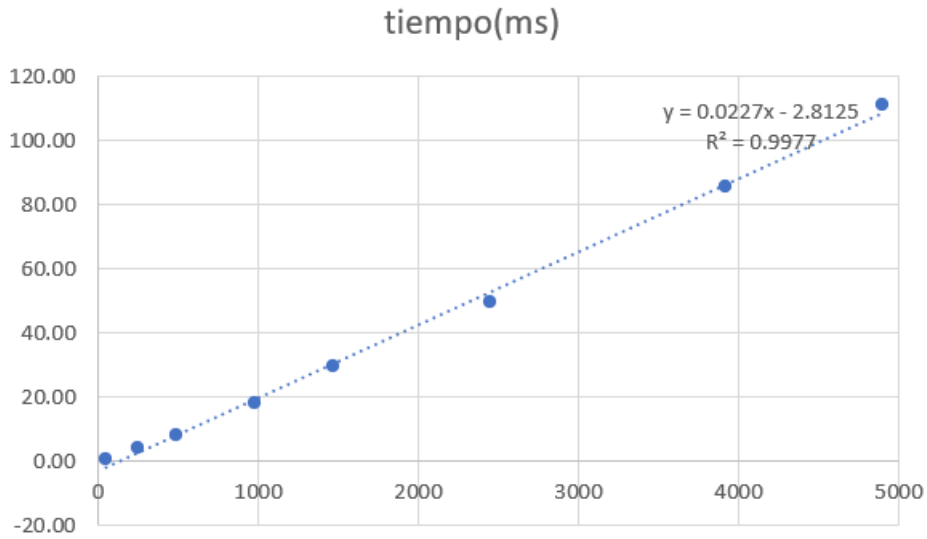
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Lista de listas	0.79
5 pct	Lista de listas	3.92
10 pct	Lista de listas	8.24
20 pct	Lista de listas	18.04
30 pct	Lista de listas	29.60
50 pct	Lista de listas	48.78
80 pct	Lista de listas	85.70
large	Lista de listas	11.14

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.

Observamos que, efectivamente, el comportamiento es lineal. Siempre toca recorrer la estructura de datos completa. Los sorteamientos aunque sean de orden mayor, merge sort es de orden “n*logn” por ejemplo, no afectan el comportamiento lineal general, ya que estos ordenamientos se hacen en listas con mucho menos datos que la estructura de datos. La correlación de la gráfica es muy cercana 1 por lo que esto confirme la linealidad de la implementación.