Universidad de los Andes
Facultad de Ingeniería

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

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

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

Estudiante 2 Juan José Díaz, código 202220657, email jj.diazo1.edu.co

Estudiante Ángel Farfán Arcila, código 202222183, email a.farfana@uniandes.edu.co

Requerimiento 1-2

Plantilla para documentar y analizar cada uno de los requerimientos.

```
def req_1_2(data_structs,year,code_sector,cmp_function):
    """
    Función que soluciona el requerimiento 1_2
    """
    # TODO: Realizar el requerimiento 1
    for i in lt.iterator(data_structs['model']['data']['table']):
        if i['key']==str(year):
            for j in lt.iterator(sort(i['value'],cmp_function)):
                if code_sector==j['Código sector económico']:
                    return j
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	data_structs , year, code_sector,cmpfunction
Salidas	j→ Diccionario
Implementado (Sí/No)	Ángel Farfán/ Juan Diaz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Recorre lista de diccionarios de los años la cual es fija(10)	O(10)
Paso 2 Ordena la lista de diccionarios según el cmmp_function que es sorteado mediante quick sort, luego selecciona de acuerdo al sector económico un diccionario ya que está ordenado según el criterio.	O(m log m)

Paso 3 Ya seleccionado el año recorre la lista de diccionarios del año	O(n)
TOTAL	***O(n^2 log n)***

Pruebas Realizadas

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

Procesadores	**M1 Pro**
Memoria RAM	16 GB
Sistema Operativo	macOS

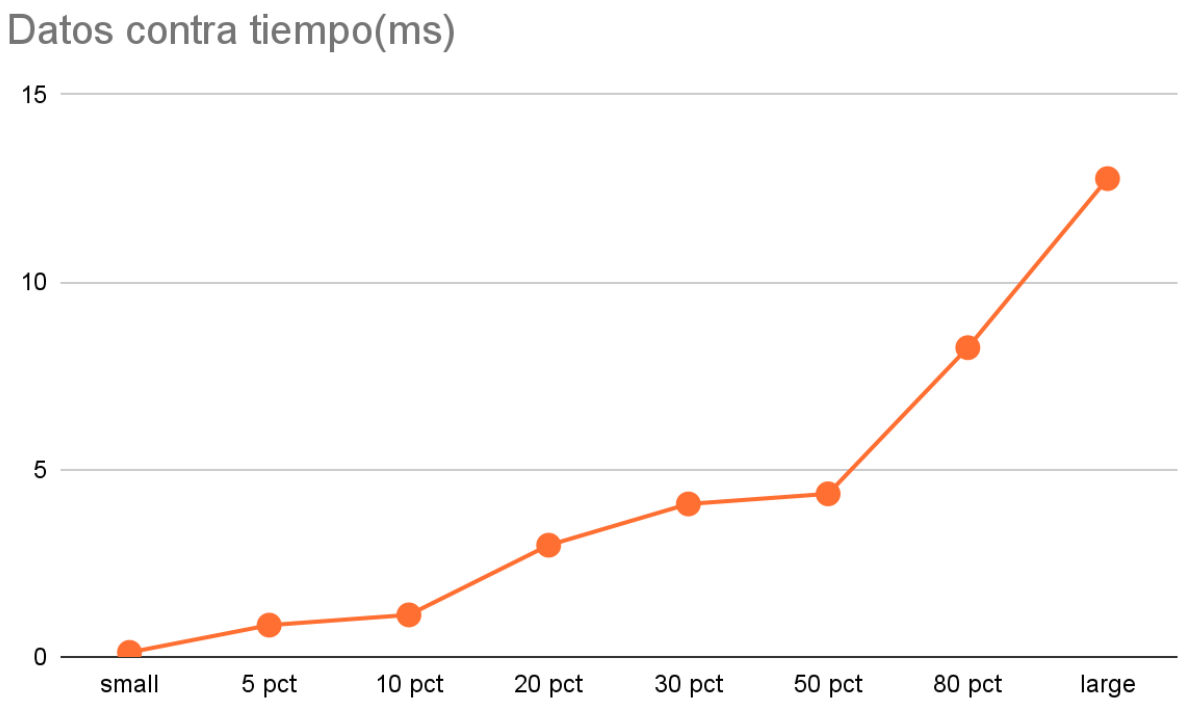
Entrada	**Tiempo (ms)**
small	0.120
5 pct	0.847
10 pct	1.12
20 pct	2.98
30 pct	4.08
50 pct	4.35
80 pct	8.25
large	12.76

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	0.120
5 pct	Dato2	0.847
10 pct	Dato3	1.12
20 pct	Dato4	2.98
30 pct	Dato5	4.08
50 pct	Dato6	4.35
80 pct	Dato7	8.25
large	Dato8	12.76

Gráficas



Análisis

-En esta función se hace un recorrido parcial en la primera iteración aunque en el peor de los casos se hace un recorrido de n el cual su límite es 10 por lo cual se ignora en la evaluación del big O Notation. Luego se realiza un sort function que en este caso es quicksort el cual es ideal para ordenar arraylist de diccionarios, esto permite que el tiempo de en el que se hace el ordenamiento no sea mucho ya que tiene de complejidad es su average case de m log m, luego selecciona el diccionario a devolver de acuerdo a unos parámetros dados. De hecho ,se puede observar que en la que el crecimiento del tiempo de acuerdo a la cantidad de datos no tiende a aumentar mucho mas bien es como si aumentara constantemente eaunque que con algunos sobresaltos debido a que los datos presentados no suelen ser constantes en el sentido que se salta del 30 por ciento al 50 cuando anteriormente solo aumentaban 10 por ciento por sample(archivo).

Requerimiento 3-4-5

Plantilla para documentar y analizar cada uno de los requerimientos.

```
def req_3_4_5(data_structs,year,cmp_function,cmp_function_dict,condition_1,condition_2):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    list_by_code_best_dict=lt.newList(datastructure='ARRAY_LIST')
    list_by_code_general=lt.newList(datastructure='ARRAY_LIST')
    for i in lt.iterator(data_structs['model']['data']['table']):
        if i['key']==year:
            for j in codes_sector:
                list_by_code=lt.newList(datastructure='ARRAY_LIST')
                list_by_code['key']=j
                for k in codes_sub_sector:
                    list_by_code_sub=lt.newList(datastructure="ARRAY_LIST")
                    list_by_code_sub['key']=k
                    for l in lt.iterator(i['value']):
                        if l['Año']==str(year) and l['Código sector económico']==j and l['Código subsector económico']==k:
                            lt.addFirst(list_by_code_sub,l)
                    if list_by_code_sub['size']!=0:
                        dict_={'Código sector económico':j,'Nombre sector económico':lt.firstElement(list_by_code_sub)['Nombre sec
                        for q in lt.iterator(list_by_code_sub):
                            dict_[condition_1]+=int(q[condition_2])
                            dict_['Total ingresos netos del subsector económico']+=int(q['Total ingresos netos'])
                            dict_['Total costos y gastos del subsector económico']+=int(q['Total costos y gastos'])
                            dict_['Total saldo a pagar del subsector económico']+=int(q['Total saldo a pagar'])
                            dict_['Total saldo a favor del subsector económico']+=int(q['Total saldo a favor'])
                        lt.addLast(list_by_code_best_dict,dict_)
                        list_by_code_sub=sort(list_by_code_sub,cmp_function)
                        lt.addFirst(list_by_code,list_by_code_sub)
                        lt.addFirst(list_by_code_general,list_by_code)
    list_by_code_best_dict=lt.firstElement(sort(list_by_code_best_dict,cmp_function_dict))
    for i in lt.iterator(list_by_code_general):
        if i['key']==list_by_code_best_dict['Código sector económico']:
            for j in lt.iterator(i):
                if j['key']==list_by_code_best_dict['Código subsector económico']:
                    return j,list_by_code_best_dict
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	datastructs, year,cmpfunction,cmpfunctiondict,condition1,condition2
Salidas	lista de diccionarios, un solo diccionario
Implementado (Sí/No)	Ángel Farfán/ Juan Jose/ Ándres Caceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1 Recorre lista de diccionarios de los años la cual es fija(10)	O(10)
Paso 2 Ya seleccionado el diccionario solo recorre el valor de una año dado por parámetro y luego recorre la lista de recortes	O(12)

Paso 3 Por cada sector hay 25 subsector entonces se recorre un tuple que contiene 25 números que hacen de subsectores	O(25)
Paso 4 Recorre el valor del diccionario dado del año entonces se tienen n elementos dentro de él	O(n)
Paso 5 Se suma dentro de la lista agrupada de sectores y subsectores unos parámetro dados , se recorre una lista de m elementos.	O(m)
Paso 6 Luego se hace una compare function que ordena a los diccionarios de acuerdo a un parámetro dado con quick sort.	O(k log k)
TOTAL	***O(n*m*k*log k)***

Pruebas Realizadas

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

Procesadores	**M1 Pro**
Memoria RAM	16 GB
Sistema Operativo	macOS

Entrada	**Tiempo (ms)**
small	2.12
5 pct	9.5
10 pct	12.8
20 pct	18.255
30 pct	19.46
50 pct	29.17
80 pct	43.10
large	48.32

Tablas de datos

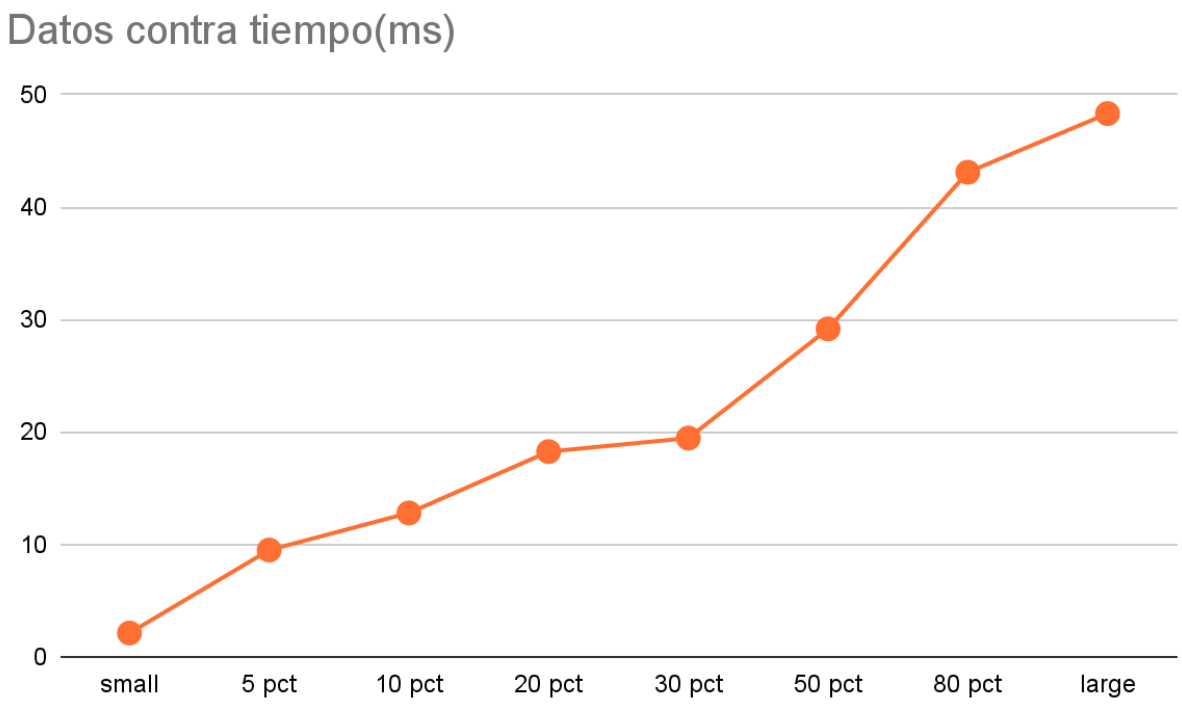
Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	2.12
5 pct	Dato2	9.5
10 pct	Dato3	12.8
20 pct	Dato4	18.255
30 pct	Dato5	19.46

50 pct	Dato6	29.17
80 pct	Dato7	43.10
large	Dato8	48.32

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Aunque puede parecer complejo la función el orden de crecimiento no es grande debido a que en la suma de diccionarios estos no pueden pasar de ser 25 ya que solo hay 25 diccionarios de subsectores y sectores solo pueden haber 12, entonces en realidad son pocos los datos que hay que comparar luego de la suma ,aunque en el proceso de la sumatoria de datos si tiene n datos ya que puede haber cualesquiera datos de combinaciones entre sectores y subsectores. Para los sort de esta función se utilizó solo quick sort el cual permite un rápido ordenamiento del arraylist de diccionarios en este caso estas comparaciones tuvieron una complejidad de k log k.

Requerimiento 6

Plantilla para documentar y analizar cada uno de los requerimientos.

```python
def req_6(data_structs,year):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    list_by_code_best_dict=lt.newList(datastructure='ARRAY_LIST')
    list_by_code_general=lt.newList(datastructure='ARRAY_LIST')
    for i in lt.iterator(data_structs['model']['data']['table']):
        if i['key']==year:
            for j in codes_sector:
                list_by_code=lt.newList(datastructure='ARRAY_LIST')
                list_by_code['key']=j
                dict_={'Código sector económico':j,'Nombre sector económico':None,'Total ingresos netos del subsector económico':0,'Total costos y gastos del subsector econ
                for k in codes_sub_sector:
                    list_by_code_sub=lt.newList(datastructure="ARRAY_LIST")
                    list_by_code_sub['key']=k
                    for l in lt.iterator(i['value']):
                        if l['Año']==str(year) and l['Código sector económico']==j and l['Código subsector económico']==k:
                            lt.addFirst(list_by_code_sub,l)
                    if list_by_code_sub['size']!=0:
                        dict_['Nombre sector económico']=lt.firstElement(list_by_code_sub)['Nombre sector económico']
                        for q in lt.iterator(list_by_code_sub):
                            dict_['Total ingresos netos del subsector económico']+=int(q['Total ingresos netos'])
                            dict_['Total costos y gastos del subsector económico']+=int(q['Total costos y gastos'])
                            dict_['Total saldo a pagar del subsector económico']+=int(q['Total saldo a pagar'])
                            dict_['Total saldo a favor del subsector económico']+=int(q['Total saldo a favor'])
                        lt.addLast(list_by_code_best_dict,dict_)
                        list_by_code_sub=sort(list_by_code_sub,cmp_req6)
                        lt.addFirst(list_by_code,list_by_code_sub)
                        lt.addFirst(list_by_code_general,list_by_code)

    list_by_code_best_dict=lt.firstElement(sort(list_by_code_best_dict,cmp_req6_dict))
    codigo_sector=list_by_code_best_dict['Código sector económico']

    best_worst_list=lt.newList(datastructure='ARRAY_LIST')

    for i in lt.iterator(list_by_code_general):
        if i['key']==codigo_sector:
            for j in lt.iterator(i):
                dict_={'Código subsector económico':j['key'],'Nombre subsector económico':None,'Total ingresos netos del subsector económico':0,'Total costos y gastos del su
                for q in lt.iterator(j):
                    dict_['Nombre subsector económico']=q['Nombre subsector económico']
                    dict_['Total ingresos netos del subsector económico']+=int(q['Total ingresos netos'])
                    dict_['Total costos y gastos del subsector económico']+=int(q['Total costos y gastos'])
                    dict_['Total saldo a pagar del subsector económico']+=int(q['Total saldo a pagar'])
                    dict_['Total saldo a favor del subsector económico']+=int(q['Total saldo a favor'])
                lt.addFirst(best_worst_list,dict_)
```

```python
                    dict_['Total saldo a favor del subsector económico']+=int(q['Total saldo a favor'])
                lt.addFirst(best_worst_list,dict_)
    best_worst_list=sort(best_worst_list,cmp_req6_dict)

    best_worst_list=lt.firstElement(best_worst_list),lt.lastElement(best_worst_list)

    list_by_code_best_dict['Subsector económico que más aporto'],list_by_code_best_dict['Subsector económico que menos aporto']=best_worst_list[0]['Código subsector económico


    list_best_aportation_best_worst,list_worst_aportation_best_worst=lt.newList(datastructure='ARRAY_LIST'),lt.newList(datastructure='ARRAY_LIST')

    for i in lt.iterator(list_by_code_general):
        if i['key']==codigo_sector:
            for j in lt.iterator(i):
                if j['key']==be (variable) list_best_aportation_best_worst: Any | None ist_best_aportation_best_worst['size'])<2:#mayor
                    lt.addLast(
                    lt.addLast(list_best_aportation_best_worst,lt.lastElement(j))#menor
                if j['key']==best_worst_list[1]['Código subsector económico']and int(list_worst_aportation_best_worst['size'])<2: #menor

                    lt.addLast(list_worst_aportation_best_worst,lt.firstElement(j))#mejor
                    lt.addLast(list_worst_aportation_best_worst,lt.lastElement(j))#menor


    best_worst_list[0]['Actividad económica que más aporto']=lt.firstElement(list_best_aportation_best_worst)
    best_worst_list[0]['Actividad económica que menos aporto']=lt.lastElement(list_best_aportation_best_worst)

    best_worst_list[1]['Actividad económica que más aporto']=lt.firstElement(list_worst_aportation_best_worst)
    best_worst_list[1]['Actividad económica que menos aporto']=lt.lastElement(list_worst_aportation_best_worst)

    # #diccionario mejor sector  #diccionario subsector que contribuyo mas #diccioanrio subsector que contribuyo menos #acts_econo_mas aportaron  #acts_econom_meno aportaron
    return list_by_code_best_dict,best_worst_list[0],best_worst_list[1]
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	data structs, year
Salidas	diccionario(principal),diccionario secundario(mayor), diccionario terciario(menor)
Implementado (Sí/No)	Ángel Farfán/Ándres Caceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Recorre lista de diccionarios de los años la cual es fija(10)	O(10)
Paso 2 Ya seleccionado el diccionario solo recorre el valor de una año dado por parámetro y luego recorre la lista de diccionarios	O(12)
Paso 3 Luego recorre la lista de diccionarios del año la cual tiene n elementos, para luego sumarlos de acuerdo a un parámetro	O(n)
Paso 4 Ordena los diccionarios (los nuevos diccionario que salen de lo sumado)de acuerdo un parámetro	O(m log m)
Paso 5 ordena los elementos del la lista de diccionario de acuerdo a un parámetro dado con quick sort	O(n log n)
Paso 6 Recorre una lista pequeña de diccionarios para luego sumarlos de acuerdo a unas características para luego ordenar los diccionarios (los nuevos diccionarios que salen de lo sumado)de acuerdo un parámetro con quick sort.	O(k log k)
Paso 7 Luego recorre esta lista de diccionarios pequeña filtrando por parámetros	O(k)
TOTAL	***O(n^2 *k^2*m log k log n log m)***

Pruebas Realizadas

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

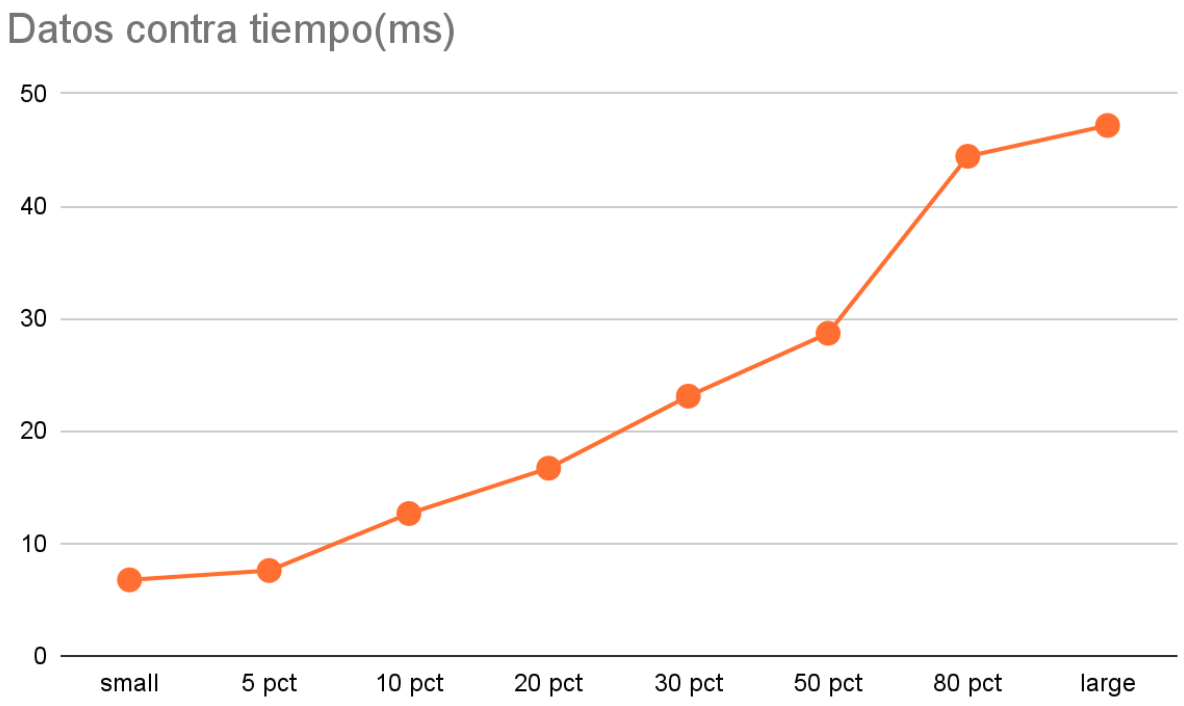
Procesadores	**M1 Pro**
Memoria RAM	16 GB
Sistema Operativo	macOS

Entrada	Tiempo (ms)
small	6.76
5 pct	7.59
10 pct	12.66
20 pct	16.69
30 pct	23.1
50 pct	28.69
80 pct	44.43
large	47.17

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

Muestra	Salida	Tiempo (ms)
small	Dato1	6.76
5 pct	Dato2	7.59
10 pct	Dato3	12.66
20 pct	Dato4	16.69
30 pct	Dato5	23.1
50 pct	Dato6	28.69
80 pct	Dato7	44.43
large	Dato8	47.17

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

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

Aunque la función es de hecho compleja se tienen que tener en cuenta que la suma de diccionarios los máximos posibles por sector son 12 y por subsector 25, y el ordenamiento de los mismos no es complicado porque son pocos datos , aunque cuando se ordenan los n datos par mirar las mayor y las menore aportación se obtiene que la complejidad es n log n ya que pueden haber muchos datos repetidos entre las combinación posibles entre sector y subsectores, aunque esto resultó muy rapido y no supone una gran suma de tiempo. De hecho, el propósito de haber hecho estos ordenamiento y recorridos es con el fin de ocupar la makro memoria ram posible ya que es un mismo for se hace varía acciones las cuales decrementar el uso de memoria en la función, como sumar mientras se ordenan los datos, entre otros. No obstante, es inevitable el uso de memoria ram ya que la función misma requiere ciertos datos para dar el resultado que no precisamente son pocos , pero en si la función tiene un orden de crecimiento casi constante como lo observado en la gráfica, si no fuera por el hecho que se salta de 30 pct a 50 pct en las tomas de tiempo se podría dar cuenta del mismo.

Requerimiento 7

Plantilla para documentar y analizar cada uno de los requerimientos.

```python
def req_7(data_structs,year,sub_code):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7

    list_1=lt.newList(datastructure="ARRAY_LIST")

    for i in lt.iterator(data_structs['model']['data']['table']):
        if i['key']==year:
            for element in lt.iterator(i['value']):
                if element['Año']==year and element['Código subsector económico']==sub_code:
                    lt.addFirst(list_1,element)
    return sort(list_1,cmp_req7)
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	data_structs,year, sub_code
Salidas	lista de diccionarios
Implementado (Sí/No)	Ángel Farfán/Ándres Caceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Recorre lista de diccionarios de los años la cual es fija(10)	O(10)
Paso 2 Ya seleccionado el diccionario solo recorre el valor de una año dado por parámetro y luego recorre la lista de diccionarios para ver si cumplen con un requisito lo añade a otra lista	O(n)
Paso 3 Hace un quick sort de acuerdo a un cmp function el cual lo ordena según el parámetro dado	O(m log m)
TOTAL	***O(n*m log m)***

Pruebas Realizadas

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

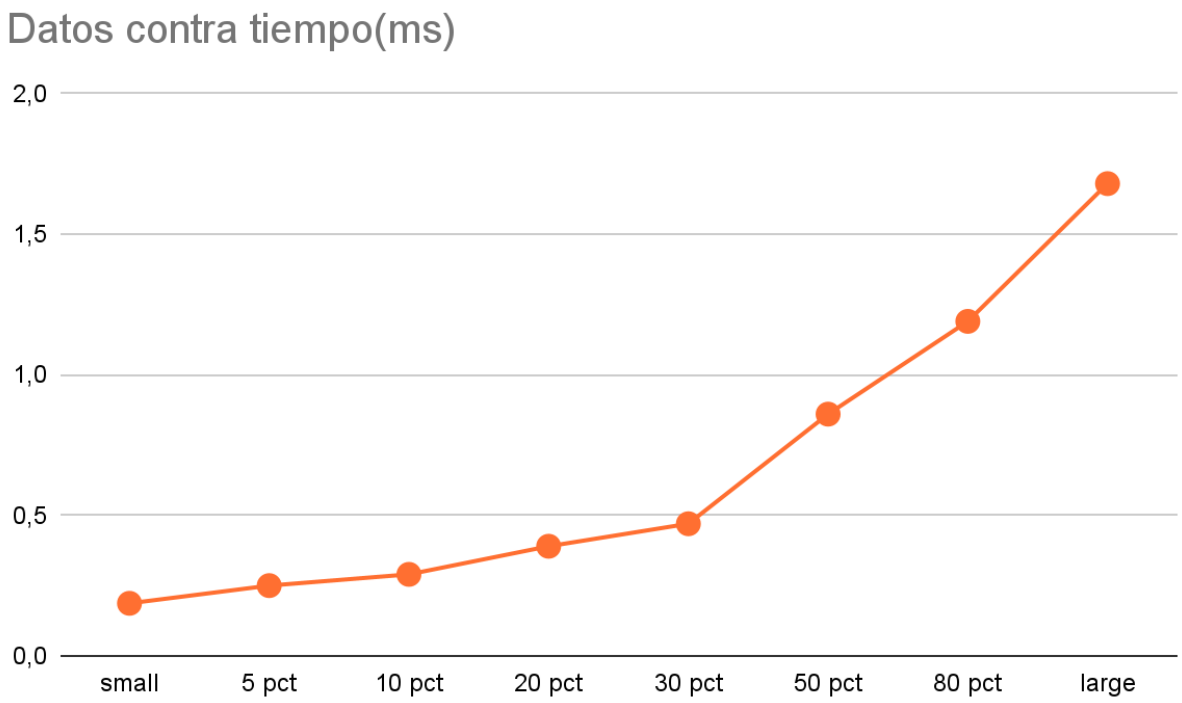
Procesadores	**M1 Pro**
Memoria RAM	16 GB
Sistema Operativo	macOS

Entrada	Tiempo (ms)
small	0.187
5 pct	0.25
10 pct	0.79
20 pct	0.39
30 pct	0.479
50 pct	0.86
80 pct	1.19
large	1.68

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

Muestra	Salida	Tiempo (ms)
small	Dato1	0.187
5 pct	Dato2	0.25
10 pct	Dato3	0.79
20 pct	Dato4	0.39
30 pct	Dato5	0.479
50 pct	Dato6	0.86
80 pct	Dato7	1.19
large	Dato8	1.68

Gráficas
Las gráficas con la representación de las pruebas realizadas.



Análisis

Este algoritmo no tiene un orden de complejidad muy alto aunque se hace un búsqueda dentro de la lista de diccionarios de los años los cuales solo son 10 y solo hace un búsqueda parcial en el mejor de los casos y en el peor de los casos recorre toda la lista de diccionarios. Luego si recorre la n elementos dentro del valor de la lista de diccionarios del año para ir añadiendo elementos a una lista para luego ordenarlo según parámetro, esto se hace mediante el algoritmo de ordenamiento quick sort.

Requerimiento 8

Plantilla para documentar y analizar cada uno de los requerimientos.

```python
def print_req_8(control):
    """
        Función que imprime la solución del Requerimiento 8 en consola
    """
    # TODO: Imprimir el resultado del requerimiento 8
    year=input("In what 'Año' do you want to search? :")
    top= int(input(" which top 'Actividades económicas' do you want to see?: "))
    data=controller.req_8(control,year,top)

    header_2=["Código actividad económica","Nombre actividad económica","Código subsector económico","Nombre subsector económico","Código sect
    print(f"{f'Req No. 8 Answer':=^40}")
    tabulate_data(data[0]["elements"],list((data[0]["elements"][0]).keys()))

    for i in lt.iterator(data[1]):
        a=i['key']
        print(f"{f'Top {top} activities in subsector {a}':=^60}")
        tabulate_data(i['elements'][:top],header_2)
        print('\n')

```

```python
def req_8(data_structs,year,top):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    contador=0
    sub=lt.newList(datastructure="ARRAY_LIST")
    act=lt.newList(datastructure="ARRAY_LIST")
    for i in lt.iterator(data_structs['model']['data']['table']):
        if i['key']==year:
            for element in lt.iterator(i['value']):
                if element['Año']==year:
                    x={"Código actividad económica":element["Código actividad económica"],"Nombre actividad económica":element["Nombre actividad económica"],"Código subsector eco
                    lt.addFirst(act,x)
                    y={"Código subsector económico":element["Código subsector económico"],"Nombre subsector económico":element["Nombre subsector económico"],"Código sector económ
                    contador=0
                    resultado=-1
                    for cua in lt.iterator(sub):
                        if cua["Código subsector económico"] == y["Código subsector económico"]:
                            resultado=contador
                            sub["elements"][resultado]["Total Impuesto a cargo"] += int(y["Total Impuesto a cargo"])
                            sub["elements"][resultado]["Total ingresos netos"]+=int(y["Total ingresos netos"])
                            sub["elements"][resultado]["Total costos y gastos"]+=int(y["Total costos y gastos"])
                            sub["elements"][resultado]["Total saldo a pagar"]+=int(y["Total saldo a pagar"])
                            sub["elements"][resultado]["Total saldo a favor"]+=int(y["Total saldo a favor"])
                        contador +=1
                    if resultado==-1:
                        lt.addFirst(sub,y)
    sub_ord=sort(sub,cmp_req8)
    act_ord=sort(act,cmp_req81)
    act_ord_org_sub_listas=lt.newList(datastructure="ARRAY_LIST")

    for j in codes_sub_sector:
        sub_list=lt.newList(datastructure="ARRAY_LIST")
        sub_list['key']=j
        for i  in lt.iterator(act_ord):
            if i["Código subsector económico"]==j:
                lt.addLast(sub_list,i)
        if sub_list['size']!=0:
            lt.addLast(act_ord_org_sub_listas,sub_list)

    return sub_ord,act_ord_org_sub_listas
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	data structs, year, top
Salidas	diccionario subsectores, diccionario actividades
Implementado (Sí/No)	Ándres Caceres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1 Recorre lista de diccionarios de los años la cual es fija(10)	O(10)

Paso 2 Ya seleccionado el diccionario solo recorre el valor de una año dado por parámetro y luego recorre la lista de diccionarios	O(12)
Paso 3 Luego recorre la lista de diccionarios del año la cual tiene n elementos, de la que se sacan dos listas de diccionarios menores y se asigna un contador	O(n*m)
Paso 4 se hace otro ciclo en el que se van sumando los valores que se piden	O(m^2)
Paso 5 ordena los elementos del la lista de diccionarios subsectores de acuerdo a un parámetro dado, con quick sort	O(n log n)
Paso 6 ordena los elementos de la lista de diccionarios actividades de acuerdo a un parámetro dado, con quick sort.	O(k log k)
TOTAL	***O(n*m+m^2+nlogn+klogk)***

Pruebas Realizadas

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

Procesadores	**M1 Pro**
Memoria RAM	16 GB
Sistema Operativo	macOS

Entrada	**Tiempo (ms)**
small	5.69
5 pct	6.46
10 pct	11.43
20 pct	15.71
30 pct	22.93
50 pct	28.23
80 pct	43.82
large	46.79

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	5.69
5 pct	Dato2	6.46
10 pct	Dato3	11.43
20 pct	Dato4	15.71
30 pct	Dato5	22.93
50 pct	Dato6	28.23
80 pct	Dato7	43.82
large	Dato8	46.79

• Comparar los tiempos de ejecución los requerimientos del 3 al 7 (individuales y avanzados) en el Reto No. 1 con los obtenidos en este reto.

TODOS ESTOS ANÁLISIS SE HICIERON CON LA CANTIDAD DE DATOS LARGE.

req3-4-5: comparando los tiempos de este requerimiento con el reto 1 podemos ver que resultó más rápido en este reto, donde al utilizar el large tenemos un tiempo de 48.32ms, mientras que en el reto 1 fué de 0.40s.

req6: al igual que en el requerimiento anterior, el del reto 2 fué más rápido que el del reto 1 con tiempos respectivos de 47.17ms y 0.061s.

req7: Nuevamente el reto 2 es más rápido en este requerimiento con un tiempo de 1.68ms contra 0.85s.

• Comparar la complejidad para cada uno de los requerimientos implementados en el Reto No. 1 con los implementados en este reto.

Reto 1	Reto 2
Requerimiento 1: O(N)	Requerimiento 1: O(N)
Requerimiento 2: O(N)	Requerimiento 2: O(N)
Requerimiento 3: O(N*MLOG(M))	Requerimiento 3: O(n*m*k*log k)
Requerimiento 4: O(N*MLOG(M))	Requerimiento 4: O(n*m*k*log k)

Requerimiento 5: O(N*MLOG(M))	Requerimiento 5: O(n*m*k*log k)
Requerimiento 6: O(N^2)	Requerimiento 6: O(n^2 *k^2*m log k log n log m)
Requerimiento 7: O(N)	Requerimiento 7: O(n*m log m)

Al comparar los req1-2 de ambos retos podemos ver que tienen la misma complejidad. Los req3-4-5 son menos complejos en el reto 1 al igual que los requerimientos 6 y 7.