ANÁLISIS DEL RETO

Isabella Sarquis Buitrago, 202221542, i.sarquis@uniandes.edu.co

Sara Valentina Rozo, 202124144, s.rozoo@uniandes.edu.co

Tomás Santiago Osuna, 202212716, t.osuna@uniandes.edu.co

Introducción

Para el desarrollo de este reto se empleó un catálogo cuya estructura, es un diccionario con 11 llaves, una por cada año en el que oscilan los datos, y un "todo" donde se guardan todos los impuestos.

Cada año tienen como valor un TAD Mapa con las siguientes especificaciones:

Numelements=43 (hay 21 subsectores por año)

Maptype= Linear Probing

Load factor=0.5.

Dentro de los mapas se almacenaron los datos de acuerdo al subsector económico al que pertenecen. Las llaves dentro de los mapas son los códigos de los subsectores y sus valores son diccionarios con la siguiente información:

```
def newcodigo(impuesto):
    diccionario={"codigo sector economico":impuesto["codigo sector economico"],
                 "nombre sector economico":impuesto["nombre sector economico"],
                 "codigo subsector economico":impuesto["codigo subsector economico"],
                 "nombre subsector economico":impuesto["nombre subsector economico"],
                 "suma descuentos tributarios":int(impuesto["descuentos tributarios"]),
                 "suma costos y gastos":int(impuesto["total costos gastos"]),
                 "suma costos y gastos nomina":int(impuesto["total costos gastos nomina"]),
                 "suma total retenciones":int(impuesto["total retenciones"]),
                 "suma ingresos netos":int(impuesto["total ingresos netos"]),
                 "suma saldo por pagar":int(impuesto["saldo a pagar"]),
                 "maximo saldo por pagar":int(impuesto["saldo a pagar"]),
                 "actividad maxima saldo por pagar":impuesto,
                 "suma saldo a favor":int(impuesto["saldo a favor"]),
                 "maximo saldo a favor":int(impuesto["saldo a favor"]),
                 "actividad maxima saldo a favor":impuesto,
                 "suma impuestos a cargo":int(impuesto["impuestos a cargo"]),

                 "actividades":None}
    diccionario["actividades"]=lt.newList(datastructure="ARRAY_LIST")

    return diccionario
```

En la lista correspondiente a la llave "actividades", se guardaron todos los registros pertenecientes a un subsector económico.

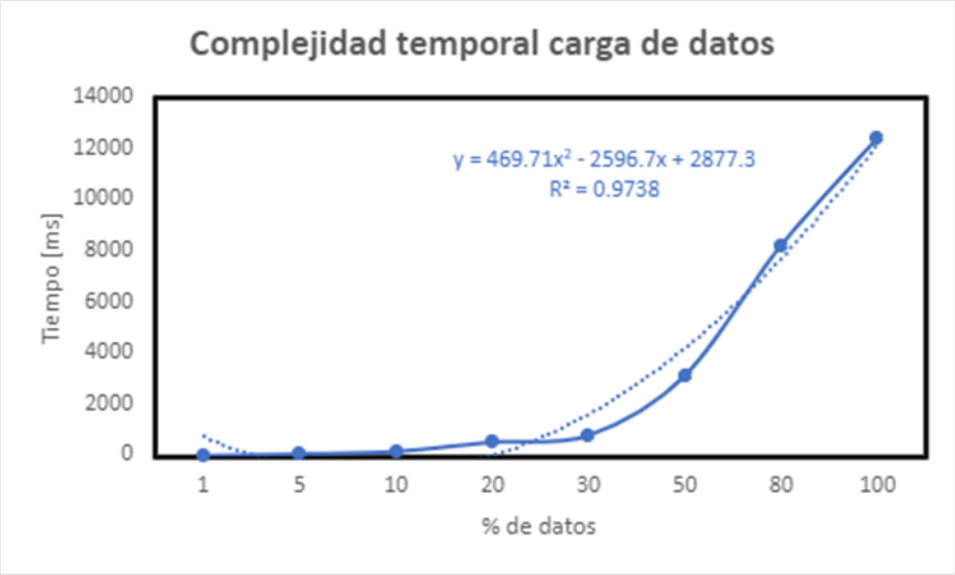
A continuación, se presenta el consumo de tiempo y memoria para la carga de datos:

% de datos cargados	Tiempo [ms]	Consumo de memoria [kB]
1	8.96	186.44
5	68.08	633.00
10	162.09	1134.04
20	525.30	2997.2
30	794.44	2997.2
50	3132.19	4819.9
80	8235.13	7541.88
100	12432.44	9345.33

Complejidad de la carga de datos **reto2**:O(N)

Complejidad de la carga de datos **reto1**:O(N^2)

Grafica Tiempo Vs % de datos cargados:



Requerimiento <<1>>

Descripción

```python
def req_1(catalogo,año,codigose):
    """
    Función que soluciona el requerimiento 1
    """

    valoresaño=catalogo[año]
    listacodigose=mp.keySet(valoresaño)


    maximo=0
    respuesta=None
    for subsector in lt.iterator(listacodigose):
        llavevalor=mp.get(valoresaño,subsector)
        dicatrabajar=me.getValue(llavevalor)

        if codigose == dicatrabajar["codigo sector economico"]:
            if int(dicatrabajar["maximo saldo por pagar"])>maximo:
                maximo= int(dicatrabajar["maximo saldo por pagar"])
                respuesta=dicatrabajar["actividad maxima saldo por pagar"]


    entrega=respuesta.copy()
    del entrega["año"]
    del entrega["codigo sector economico"]
    del entrega["total retenciones"]
    del entrega["impuestos a cargo"]

    return entrega
```

En este requerimiento se busca obtener la actividad económica con mayor saldo a pagar para un sector económico y un año especifico. El procedimiento a seguir a nivel general se basa en tres pasos, primero, encontrar el mapa dentro del catálogo con el cual se va a trabajar, luego, se recorren los subsectores que componen el mapa buscando que coincidan con el indicado por el usuario, si la respuesta es afirmativa, se realiza la comparación de su valor correspondiente a la llave "máximo saldo por pagar" y se extrae la actividad con el máximo valor de este criterio, una vez se obtiene, se formatea para ser enviado al controlador.

Entrada	Catálogo, año especifico y el código del sector.
Salidas	Lista con la información de la actividad a mayor saldo a pagar.
Implementado (Sí/No)	Si

Análisis de complejidad

Pasos	**Complejidad**
Asignación en el mapa del año a trabajar.	O(1)

Recorrido subsectores en lista de códigos.	O(N)
Conseguir la pareja llave-valor.	O(1)
Conseguir el valor de la pareja llave-valor.	O(1)
Determinar si el sector extraído es el mismo al del parámetro.	O(1)
Comparación actividad económica a mayor saldo a pagar.	O(1)
Copia respuesta	0(1)
Eliminación información innecesaria	0(1)
TOTAL	***O(N)***

Complejidad requerimiento 1 **reto2**: O(N)
Complejidad requerimiento 1 **reto1**: $O(N^{5/2})$

Pruebas Realizadas

Las pruebas se hicieron ingresando el año 2016 y el subsector con el código 1. Las especificaciones del computador utilizado para la toma de datos son las siguientes:

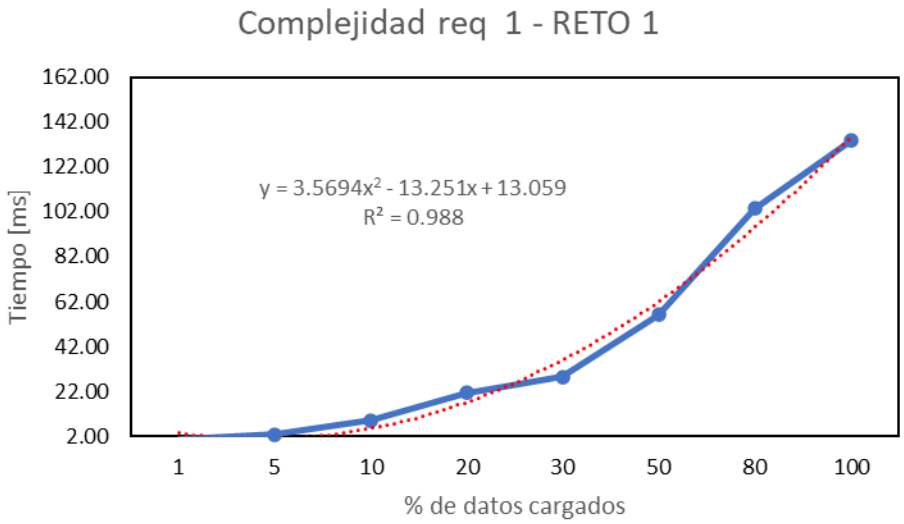
Procesadores	**AMD Ryzen 5 5500U with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

Entrada	**Tiempo (ms)**	**Memoria(kB)**
small	2.59	35.55
5 pct	2.90	35.80
10 pct	3.02	36.05
20 pct	3.20	36.30
30 pct	3.46	36.36
50 pct	3.95	36.36
80 pct	4.20	36.49
large	4.39	36.49

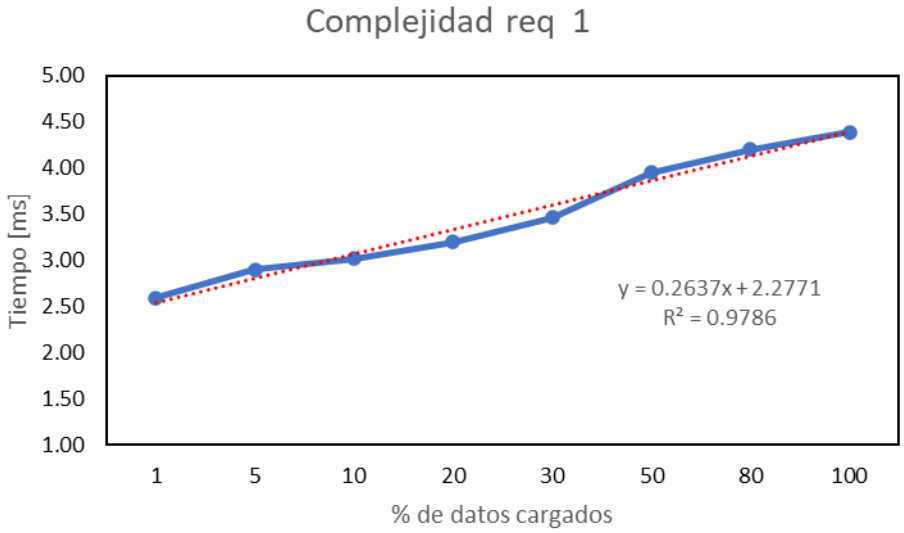
Gráficas

RETO 1:
Complejidad temporal: $O(N^{5/2})$



RETO 2:

Complejidad temporal: O(N)



Análisis

Por mucho que en este reto se haya implementado un mapa hash, en las gráficas se ve cómo la correlación entre el porcentaje de datos y el tiempo en el reto 1 llega ser más precisa; sin embargo, el reto 2 sobresale en varios aspectos a su manera, como lo son el tipo de gráfica (de una cuadrática a una lineal) y la dispersión en el tiempo, siendo más rápido los procesos (de tener valores mayores que 100 ahora se tienen valores menores que 5). Así mismo, se ve que la complejidad de recorrido disminuyó de $N^{5/2}$ a N, es decir se redujo 1 y medio dicha complejidad, evidenciado el uso de mapas, pues se redujeron los recorridos innecesarios que sucedían al usar únicamente listas a unos más óptimos. Además, en la tabla se puede ver que la memoria utilizada es óptima, pues no hay cambios bruscos.

Requerimiento <<2>>

Descripción

```
342  def req_2(catalogo,año,codigose):
343      """
344      Función que soluciona el requerimiento 2
345      """
346      valoresaño=catalogo[año]
347      listacodigose=mp.keySet(valoresaño)
348
349      maximo=0
350      respuesta=None
351
352      for subsector in lt.iterator(listacodigose):
353          llavevalor=mp.get(valoresaño,subsector)
354          dicatrabajar=me.getValue(llavevalor)
355
356          if codigose == dicatrabajar["codigo sector economico"]:
357              if int(dicatrabajar["maximo saldo a favor"])>maximo:
358                  maximo= int(dicatrabajar["maximo saldo a favor"])
359                  respuesta=dicatrabajar["actividad maxima saldo a favor"]
360
361
362
363      entrega=respuesta.copy()
364
365      del entrega["año"]
366      del entrega["codigo sector economico"]
367      del entrega["total retenciones"]
368      del entrega["impuestos a cargo"]
369
370      return entrega
```

En este requerimiento se busca obtener la actividad económica con mayor saldo a pagar para un sector económico y un año especifico. El procedimiento a seguir a nivel general se basa en tres pasos, primero, encontrar el mapa dentro del catálogo con el cual se va a trabajar, luego, se recorren los subsectores que componen el mapa buscando que coincidan con el indicado por el usuario, si la respuesta es afirmativa, se realiza la comparación de su valor correspondiente a la llave "máximo saldo a favor" y se extrae la actividad con el máximo valor de este criterio, una vez se obtiene se formatea para ser enviado al controlador.

Entrada	Catálogo, año especifico y el código del sector
Salidas	Lista con la información de la actividad a mayor saldo a favor
Implementado (Sí/No)	Si

Análisis de complejidad

Pasos	**Complejidad**
Asignación en el mapa del año a trabajar	O(1)
Recorrido subsectores en lista de códigos	O(N)
Conseguir la pareja llave-valor	O(1)
Conseguir el valor de la pareja llave-valor	O(1)
Determinar si el sector extraído es el mismo al del parámetro	O(1)

Comparación actividad económica a mayor saldo a favor.	O(1)
Copia respuesta	0(1)
Eliminación información innecesaria	0(1)
TOTAL	***O(N)***

Complejidad requerimiento 2 **reto2**: O(N)
Complejidad requerimiento 2 **reto1**: O(N^2logN)

Pruebas Realizadas

Las pruebas se hicieron ingresando el año 2016 y el subsector con el código 3. Las especificaciones del computador utilizado para la toma de datos son las siguientes:

Procesadores	**Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

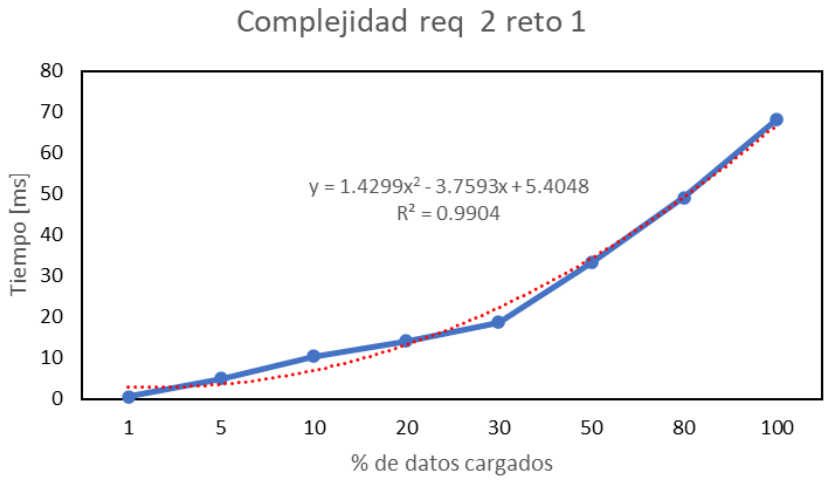
Tablas de datos

Entrada	**Tiempo (ms)**	**Memoria(kB)**
small	2,24	35,62
5 pct	2,40	35,88
10 pct	2,50	36,12
20 pct	2,58	36,38
30 pct	2,69	36,44
50 pct	2,63	36,44
80 pct	2,71	36,56
large	3,01	36,56

Gráficas

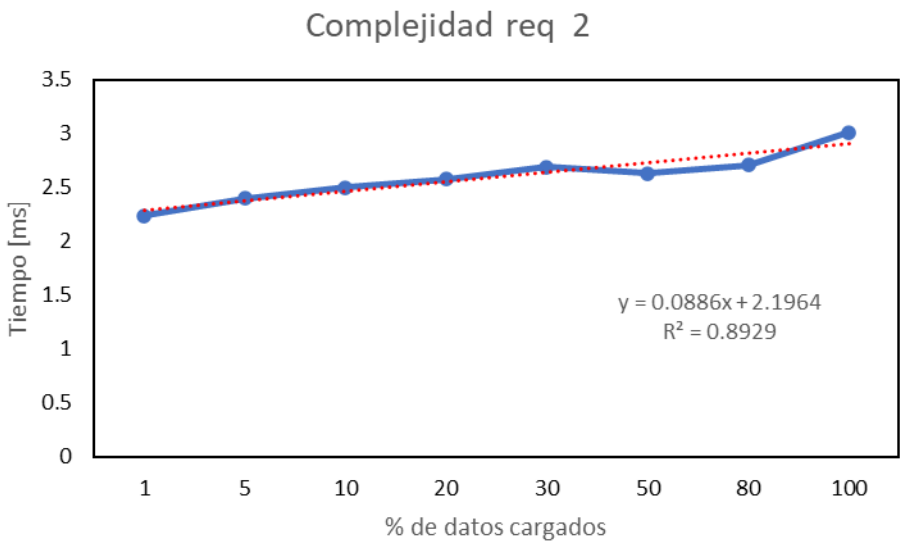
RETO 1:

Complejidad Temporal: O(N^2logN)



RETO 2:

Complejidad temporal: O(N)



Análisis

Según el análisis paso a paso realizado anteriormente, el requerimiento tiene una complejidad temporal de O(N) generada por recorrer los subsectores. Esto coincide con la gráfica obtenida, ya que, al hacer la regresión lineal, se obtiene un coeficiente R de 0,89, lo que se califica como alto.

Con respecto al reto anterior, se logró reducir la complejidad de $O(N^2 logN)$ a O(N) gracias a la carga de datos y a las tablas de hash, ya que no eran necesarios recorridos para acceder a los valores especificados, pues la información estaba organizada según las llaves que el requerimiento exigía (subsectores).

En cuanto a la gráfica, la línea de tendencia tiene una pendiente menor a 0,1, por lo que, además de conseguir una complejidad lineal, se logró que no aumentara exageradamente a medida que se incrementaba la cantidad de datos. Por último, se logró implementar un algoritmo estable que no varía mucho con respecto a los tiempos esperados, de hecho, los tiempos se encuentran por debajo de la línea de tendencia o justo encima de esta y, como complemento, se tiene un cambio en el consumo de memoria poco significativo.

Requerimiento <<3>>

Descripción

```
372  def req_3(catalogo,año):
373      AñoEsp=catalogo[año]
374      CodSec=mp.keySet(AñoEsp)
375      MinRet=None
376      for RecCod in lt.iterator(CodSec):
377          KeyVal=mp.get(AñoEsp,RecCod)
378          Values=me.getValue(KeyVal)
379          ImpIni={"codigo sector economico":Values["codigo sector economico"],
380                  "nombre sector economico":Values["nombre sector economico"],
381                  "codigo subsector economico":Values["codigo subsector economico"]
382                  "nombre subsector economico":Values["nombre subsector economico"]
383                  "total retenciones":Values["suma total retenciones"],
384                  "suma total ingresos netos":Values["suma ingresos netos"],
385                  "total costos y gastos":Values["suma costos y gastos"],
386                  "total saldo a pagar":Values["suma saldo por pagar"],
387                  "total saldo a favor":Values["suma saldo a favor"],
388                  "actividades":Values["actividades"]}
389          if MinRet==None:
390              MinRet=ImpIni["total retenciones"]
391              ImpFin=ImpIni
392          elif ImpIni["total retenciones"]<MinRet:
393              MinRet=ImpIni["total retenciones"]
394              ImpFin=ImpIni
395
396      ActEco=ImpFin["actividades"]
397      sa.sort(ActEco,cmpTotalRet)
398
399      for Del in lt.iterator(ActEco):
400          del Del["año"]
401          del Del["total costos gastos nomina"]
402          del Del["descuentos tributarios"]
403          del Del["codigo sector economico"]
404          del Del["nombre sector economico"]
405          del Del["codigo subsector economico"]
406          del Del["nombre subsector economico"]
407      ActImp=None
408      ActMen=None
409      Tam=lt.size(ActEco)
```

```
410      if Tam<6:
411          ActImp=ActEco
412      else:
413          ActImp=lt.subList(ActEco,1,3)
414          ActMen=lt.subList(ActEco,Tam-2,3)
415      del ImpFin["actividades"]
416      Imp=lt.newList("ARRAY_LIST")
417      lt.addLast(Imp,ImpFin)
418      return Imp,ActImp,ActMen
```

En este on menor retenciones en un año especifico. El procedimiento a seguir a nivel general es el siguiente: primero se encuentra el mapa dentro del catálogo con el cual se va a trabajar, luego, se recorren los subsectores de manera que va comparando sus retenciones así hasta quedar el menor de todos frente a retenciones, después se extraen las actividades que colaboraron a dicho subsector, sacando las 3 que más aportaron y las 3 que menos, siendo así si llegan a haber menos

de 6 pues se muestran todas en orden, finalmente a una lista se añade dicho subsector y se proyecta esta junto a sus actividades.

Entrada	Catálogo con toda la información, año a buscar
Salidas	El subsector con más retenciones poseía en el año ingresado por parámetro retenciones descuentos tributarios aportaron al subsector y las 3 que menos aportaron
Implementado (Sí/No)	Si (Tomás Osuna)

Análisis de complejidad

Pasos	**Complejidad**
Asignar a AñoEsp (año especifico) el mapa del año a trabajar	O(1)
Usar la función keySet para poder obtener una lista con todas las llaves de los códigos de dicha llave año, códigos que corresponden son los subsectores dentro del año	O(1)
Iniciar contador de comparación de menor retención Minret (mínimo retención)	O(1)
Iterar en la lista de los subsectores RecCod para obtener las retenciones totales del subsector	O(N)
Asignar el 1er valor mínimo y después cambiar el valor mínimo si el valor de retención es menor	O(1)
Agregar las actividades económicas del subsector con menor retenciones totales a una lista	O(1)
Ordenar la lista de las actividades con Shell Sort	O(($N^{3/2}$)
Iterar las actividades económicas para crear el formato pedido, sea menor de 6 actividades solo mostrar dichas	O(N)
Crear las sublistas necesarias (3 actividades menores y 3 mayores que efectúan en dicho subsector)	O(1)
TOTAL	***O***($N^{3/2}$***)***

Complejidad requerimiento 3 **reto2**: O($N^{3/2}$)
Complejidad requerimiento 3 **reto1**: O(N^{3})

Pruebas Realizadas

Las pruebas se realizaron ingresando el año 2013. Las especificaciones del computador utilizado para la toma de datos son las siguientes:

Procesadores	**AMD Ryzen 5 5500U with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

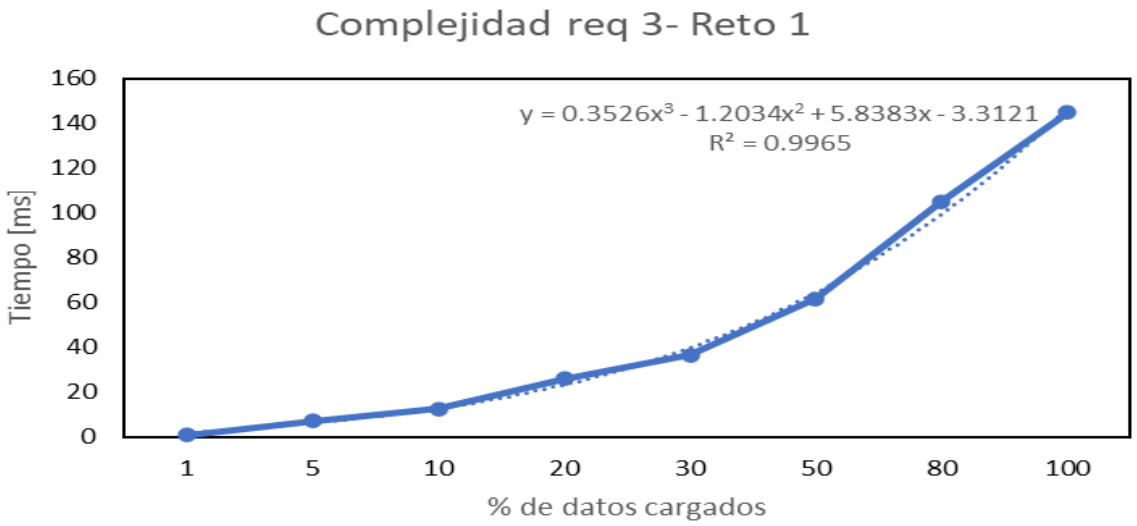
Entrada	**Tiempo (ms)**	**Memoria(kB)**

small	2.75	35.53
5 pct	2.91	35.91
10 pct	3.01	36.10
20 pct	3.07	36.28
30 pct	3.26	36.41
50 pct	3.58	36.53
80 pct	4.10	36.66
large	4.20	36.66

Gráficas

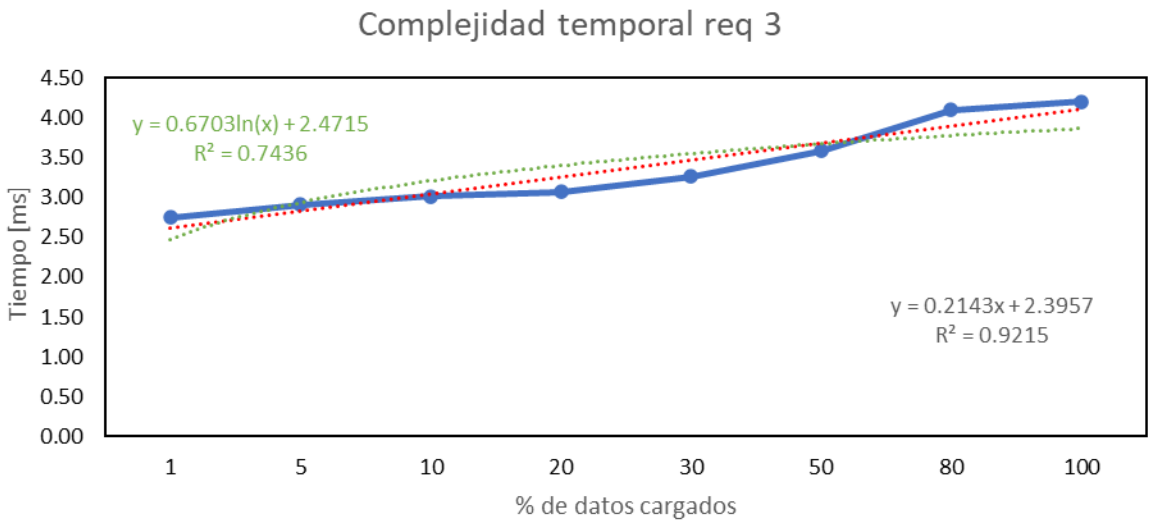
RETO 1:

Complejidad Temporal: $O(N^3)$



RETO 2:

Complejidad temporal: O(N)



Análisis

En este requerimiento es evidente la eficiencia de poder recorrer un mapa de listas a una lista de listas, de manera que anteriormente se tuvieron que hacer 3 recorridos de los datos para organizar, buscar y encontrar los datos necesarios, teniendo una complejidad de O(N^3), la cual disminuyó a $\boldsymbol{O}(N^{3/2})$ en este reto.

Referente al código, para el ordenamiento y el empleo de los datos fue eficiente usar el mapa para recorrer una única lista (que ya venía organizada) y organizar referente al parámetro por aparte, haciendo que dicho ordenamiento fuera el recorrido más largo a comparación de un recorrido sobre otro junto a otro ordenamiento haciendo que funcione de misma manera, pero con más demora en tiempo, es decir de tener valores de 150ms pasar a valores de 4ms. También cabe resaltar que la memoria cambió casi proporcionalmente con la cantidad de datos.

Requerimiento <<4>>

```
def req_4(catalogo,año):
    """
    Función que soluciona el requerimiento 4
    """
    añoescogido=catalogo[año]

    listasubsectores=mp.keySet(añoescogido)


    valormaximo=0
    subsectormaximo=None

    for subsector in lt.iterator(listasubsectores):
        llavevalor=mp.get(añoescogido,subsector)

        diccionariosubsector=me.getValue(llavevalor)


        if valormaximo<int(diccionariosubsector["suma costos y gastos nomina"]):
            valormaximo=int(diccionariosubsector["suma costos y gastos nomina"])
            subsectormaximo=subsector


    parejallavevalor=mp.get(añoescogido,subsectormaximo)
    diccionarioactividades=me.getValue(parejallavevalor)

    merg.sort(diccionarioactividades["actividades"],cmpcostosygastonomina)
    if lt.size(diccionarioactividades["actividades"])<6:
        list6=lt.newList(datastructure="ARRAY_LIST")
        for actividad in lt.iterator(actividadesmas):

            diccionario1={"codigo actividad economica":actividad["codigo actividad economica"],
                          "nombre actividad economica":actividad["nombre actividad economica"],
                          "total costos gastos nomina":actividad["total costos gastos nomina"],
                          "total ingresos netos":actividad["total ingresos netos"],
                          "total costos gastos":actividad["total costos gastos"],
                          "total saldo por pagar":actividad["saldo a pagar"],
                          "total saldo a favor":actividad["saldo a favor"]}
            lt.addLast(list6,diccionario1)
        copia=diccionarioactividades.copy()
        del copia["suma descuentos tributarios"]
        del copia["suma total retenciones"]
        del copia["actividades"]

        return copia, list6

    else:
        actividadesmas=lt.subList(diccionarioactividades["actividades"],1,3)
        actividadesmenos=lt.subList(diccionarioactividades["actividades"],(lt.size(diccionarioactividades["actividades"])-2),3)

        actividadesmasultimo=lt.newList(datastructure="ARRAY_LIST")
        actividadesmenossultimo=lt.newList(datastructure="ARRAY_LIST")
        for actividad in lt.iterator(actividadesmas):

            diccionario1={"codigo actividad economica":actividad["codigo actividad economica"],
                          "nombre actividad economica":actividad["nombre actividad economica"],
                          "total costos gastos nomina":actividad["total costos gastos nomina"],
                          "total ingresos netos":actividad["total ingresos netos"],
                          "total costos gastos":actividad["total costos gastos"],
                          "total saldo por pagar":actividad["saldo a pagar"],
                          "total saldo a favor":actividad["saldo a favor"]}

            lt.addLast(actividadesmasultimo,diccionario1)


        for actividad in lt.iterator(actividadesmenos):
            diccionario2={"codigo actividad economica":actividad["codigo actividad economica"],
                          "nombre actividad economica":actividad["nombre actividad economica"],
                          "total costos gastos nomina":actividad["total costos gastos nomina"],
                          "total ingresos netos":actividad["total ingresos netos"],
                          "total costos gastos":actividad["total costos gastos"],
                          "total saldo por pagar":actividad["saldo a pagar"],
                          "total saldo a favor":actividad["saldo a favor"]}

            lt.addLast(actividadesmenossultimo,diccionario2)

    copia=diccionarioactividades.copy()
    del copia["suma descuentos tributarios"]
    del copia["suma total retenciones"]
    del copia["actividades"]



    return  copia,actividadesmasultimo,actividadesmenossultimo
```

Descripción

Es importante recordar que cada año presente en el archivo con los datos cuenta con su propio mapa en la solución implementada. Las llaves dentro de cada mapa son los códigos de los subsectores económicos y su valor corresponde a un diccionario con la información alrededor de ese subsector. Para resolver esta problemática se iteran las parejas llave valor dentro del mapa del año deseado y se compara el valor de la llave "suma costos gastos nomina" para sacar el máximo y el diccionario correspondiente con dicho valor. Una vez se obtiene el diccionario, se organizan de menor a mayor los impuestos que conforman la lista de la llave "actividades", y se saca una sub lista de los tres primeros impuestos y los tres últimos.

Entrada	Para poder resolver el requerimiento es necesario que el usuario indique el año sobre el cual desea obtener información.
Salidas	El algoritmo devuelve un diccionario con la información del subsector económico con los mayores costos y gastos de nómina, una lista que contiene las tres actividades económicas que más aportaron al subsector y las tres actividades económicas que menos aportaron.
Implementado (Sí/No)	Sí por Sara Valentina Rozo Oviedo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Instrucción: For "subsector" in lt.iterator (listasubsectores) Recorrido de las parejas llave valor del mapa año.	O(N)
Instrucción: mp.getvalue(llavevalor) Obtención de las parejas llave valor en el mapa año.	O(1)
Condicional: Comparación del valor correspondiente a la llave ["suma costos y gastos nomina] de un subsector con el valor establecido en "valormaximo".	O(1)
Asignación de variables: Asignar a la variable "valor maximo", el valor del subsector correspondiente a la llave "suma costos y gastos nómina" y asimismo asignar el diccionario a la variable "subsectormaximo".	O(1)
Instrucción: merg.sort(diccionarioactividades["actividades"],cmpcostosygastosnomina) Organización de las actividades del subsector con mayor costos y gastos de nómina.	O (NlogN)
Instrucción: For actividad in lt.iterator(actividadesmas) Iteración de las tres actividades con mayores costos y gastos de nómina del subsector seleccionado.	O(N)
Instrucción: For actividad in lt.iterator(actividadesmenos) Iteración de las tres actividades con menores costos y gastos de nómina del subsector seleccionado.	O(N)
Procedimiento	***O(N(1+1+1) + NlogN+N+N)***

Total	***O(5N+NlogN)***

Complejidad requerimiento 4 **reto2**: O(NLogN)
Complejidad requerimiento 4 **reto1**: $O(N^3)$

Pruebas Realizadas

Se probó el requerimiento ingresando el año 2021. Las especificaciones del computador para la toma de datos son las siguientes:

Procesadores	**Intel(R) Core(TM) i5-8250u CPU @ 1.60GHZ 1.80GHZ**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home

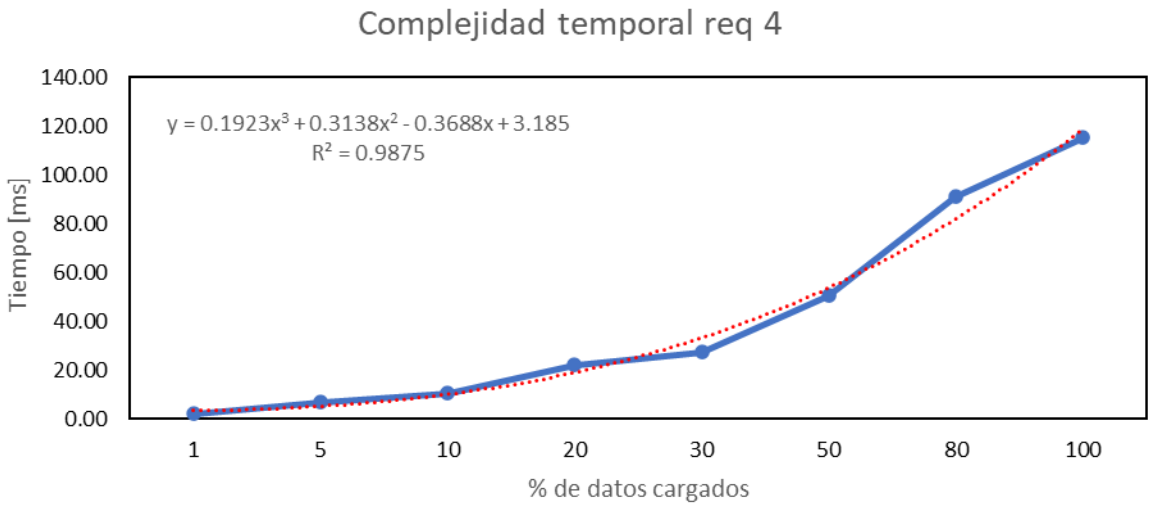
Tablas de datos

% de datos cargados	Tiempo [ms]	Consumo de memoria [kB]
1	2.86	42.07
5	3.05	42.98
10	3.35	43.42
20	4.27	44.34
30	4.89	45.35
50	5.96	46.01
80	6.83	46.11
100	7.36	45.31

Gráficas

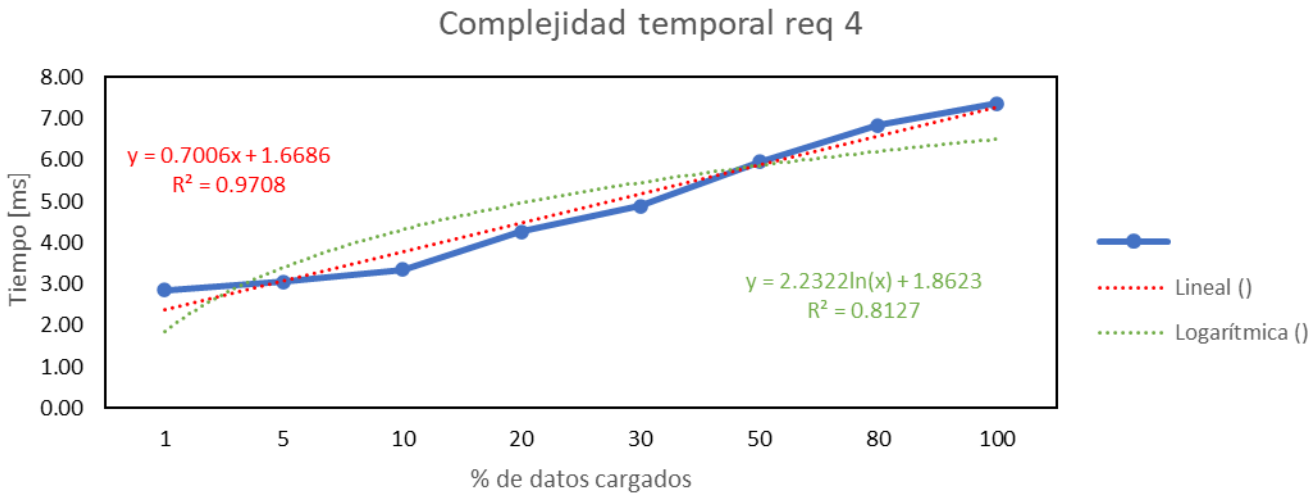
RETO 1:

Complejidad: $O(N^3)$



RETO 2:

Complejidad: O(Nlog(N))



Análisis

En primera instancia se logra evidenciar que la gráfica se ajusta con un alto coeficiente de correlación al modelo lineal, mejor que al modelo logarítmico. Para los porcentajes más altos de datos, la gráfica se encuentra por encima de ambas líneas de tendencia indicando una complejidad mayor para este grupo de datos, aspecto que es coherente con la complejidad calculada de NlogN.

Evidentemente, las tablas de hash representan una ventaja ante la implementación de las listas para el cumplimiento del requerimiento. La diferencia principal radica en que los datos ya se encuentran organizados de acuerdo al subsector, la sumatoria de los costos y gastos de nómina se encuentra guardada en una pareja llave valor y se realiza mientras se guardan los registros durante la carga. En ese orden de ideas, acceder a esta información es prácticamente de forma inmediata o, en otras palabras, tiene una

complejidad constante, lo único que resta es organizar los diccionarios de cada subsector de acuerdo al criterio de interés y posteriormente las actividades que componen al mayor subsector con costos y gastos de nómina. En el caso de las listas el algoritmo realizado era ineficiente, pues, solo para realizar la sumatoria, es necesario iterar los datos para identificar el subsector en el que se encuentran, lo cual, ya cuenta con una complejidad O(N). La implementación de una lista de listas, para obtener los resultados de cada subsector, dificulta el buscar, comparar, y acceder a los datos deseados y tiene una complejidad de $O(N^2)$.

Por otra parte, al comparar los resultados con los obtenidos en el reto 1, es importante recalcar la disminución significativa de los tiempos para cada uno de los tamaños del archivo. El tiempo requerido para ejecutar el algoritmo del presente reto, con el archivo más grande, es aproximadamente 20 veces más pequeño que en el primer reto.

Por último, vemos que la memoria representa un cambio proporcionalmente menor en relación al cambio en el tamaño de los archivos. Teniendo en cuenta todo lo mencionado anteriormente, se puede establecer una complejidad temporal y de memoria superior y aceptable en comparación a los resultados obtenidos en la entrega anterior.

Requerimiento <<5>>

Descripción

```
500
501 def req_5(data_structs,año):
502     """
503     Función que soluciona el requerimiento 5
504     """
505     catalog_año=data_structs[año]
506     max=0
507     subsector_max=0
508     codigos=mp.keySet(catalog_año)
509     for cod in lt.iterator(codigos):
510         llave_valor=mp.get(catalog_año, cod)
511         impuestos=me.getValue(llave_valor)
512         impuestos=formato_5(impuestos)
513         if impuestos["total descuentos tributarios"]>max:
514             max=impuestos["total descuentos tributarios"]
515             subsector_max=impuestos
516
517
518     lst_actividades= subsector_max["actividades"]
519     merg.sort(lst_actividades,cpm_descuentos_tributarios)
520
521     for dic in lt.iterator(lst_actividades):
522         del dic["año"]
523         del dic["total costos gastos nomina"]
524         del dic["total retenciones"]
525         del dic["codigo sector economico"]
526         del dic["nombre sector economico"]
527         del dic["codigo subsector economico"]
528         del dic["nombre subsector economico"]
529     if lt.size(lst_actividades)<=3:
530         mayores=lst_actividades
531         menores=None
```

```
531         menores=None
532     else:
533         mayores=lt.subList(lst_actividades,1,3)
534         menores=lt.subList(lst_actividades,(lt.size(lst_actividades))-2,3)
535     del subsector_max["actividades"]
536     subsector=lt.newList("ARRAY_LIST")
537     lt.addLast(subsector,subsector_max)
538
539     return subsector,mayores,menores
```

Se hace un contador para encontrar el máximo total de descuentos tributarios dentro del mapa del año a buscar. Se inicia con un valor inicial de comparación de 0 y este se va modificando si hay un descuento tributario mayor. Finalmente se extraen las actividades económicas del subsector que quedó asignado en la variable de subsector_max y se crean las sublistas necesarias.

Entrada	Catálogo con toda la información, año a buscar.
Salidas	El subsector que más descuentos tributarios aportó en el año ingresado por parámetro y las 3 actividades económicas que más descuentos tributarios aportaron al subsector y las 3 que menos aportaron.

Implementado (Sí/No)	Si (Isabella Sarquis Buitrago)

Análisis de complejidad

Pasos	**Complejidad**
Asignar a catalog_año el mapa del año a trabajar.	O(1)
Usar la función keySet para tener una lista con todas las llaves del mapa, las cuales son los subsectores dentro del año.	O(1)
Iniciar 2 contadores para comparar el máximo aportador de descuentos tributarios.	O(1)
Iterar en la lista de los subsectores para obtener los descuentos tributarios totales del subsector.	O(n)
Cambiar el valor máximo si el valor de descuentos tributarios es mayor al anterior.	O(1)
Agregar las actividades económicas del subsector con mayores descuentos tributarios mayores a una lista.	O(1)
Ordenar la lista de las actividades con Merge Sort.	O(nlogn)
Iterar las actividades económicas para crear el formato pedido.	O(n)
Crear las sublistas necesarias.	O(1)
TOTAL	***O(nlogn)+ 2O(n)+ (1)≈O(nlogn)***

Complejidad requerimiento 5 **reto2**: O(NLogN)
Complejidad requerimiento 5 **reto1**: $O(N^3)$

Pruebas Realizadas

Se realizaron las pruebas ingresando el año 2012. Las especificaciones del computador utilizado para la toma de datos son las siguientes:

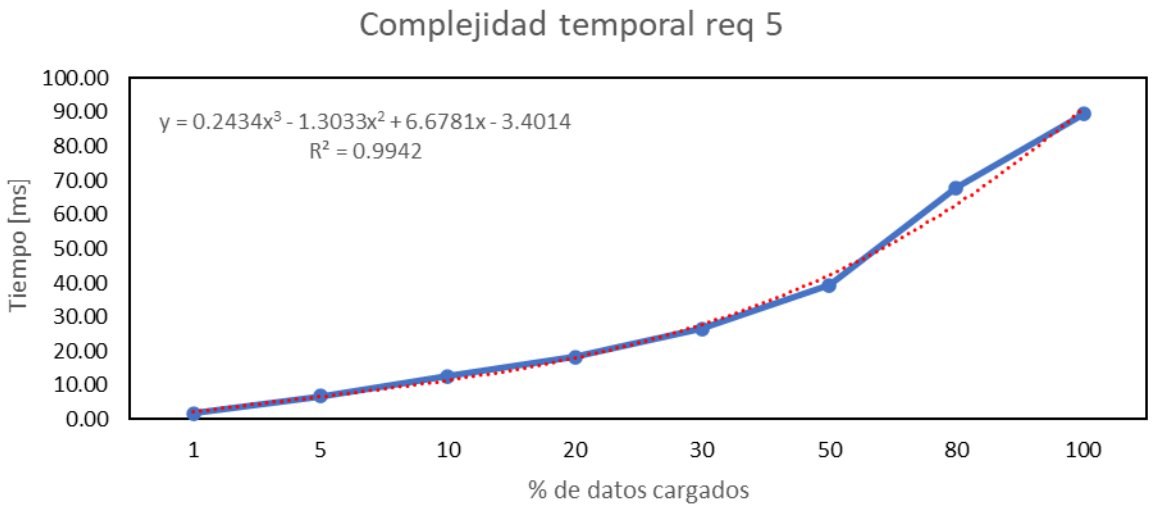
Procesadores	**Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Tablas de datos

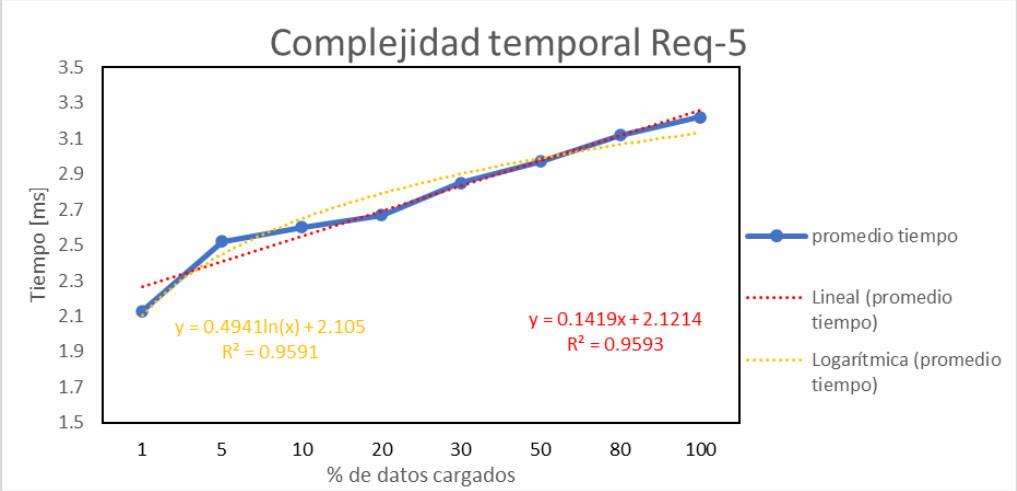
Entrada	**Tiempo (ms)**	**Memoria(kB)**
small	2,13	35,65
5 pct	2,52	36,36
10 pct	2,60	36,67
20 pct	2,67	36,80
30 pct	2,85	37,05
50 pct	2,97	37,49
80 pct	3,12	37,64
large	3,22	37,70

Gráficas

RETO 1:
Complejidad Temporal: $O(N^3)$



RETO 2:
Complejidad Temporal: O(nlogn)



Análisis

En este requerimiento se consiguió una complejidad logarítmica de O(NlogN), la cual se da por implementar el algoritmo de ordenamiento merge sort para obtener los valores de mayor a menor de las actividades del subsector. La gráfica confirma que se obtuvo una complejidad de este tipo, pues la línea de tendencia logarítmica tiene un R de 0,96, lo cual es bastante alto; sin embargo, al hacer la comparación con la tendencia lineal, se obtiene un resultado similar, por lo que se da a entender que, si no fuera por el ordenamiento, se conseguiría una complejidad O(N).

Al comparar los resultados con el requerimiento 5 del reto anterior es evidente la disminución en la complejidad temporal, ya que se pasó de $O(N^3)$ a O(nlogn). Lo anterior se debe a que en la carga de datos se ordenaron los datos por subsector, por lo que la complejidad no aumentaba al obtener las llaves a

comparar. Además, no fueron necesarios recorridos dentro de recorridos, pues las tablas de hash permiten que la información sea más accesible con las funciones get y getValue.

Finalmente, se lograron tiempos hasta 25 veces más rápidos que en el reto anterior y el consumo de memoria cambiaba únicamente 2kB al aumentar los datos en un 100%, por lo que se puede considerar que el algoritmo alcanza un equilibrio óptimo entre la complejidad temporal y el espacio en memoria.

Requerimiento <<6>>

```
def req_6(catalogo,año):
    """
    Función que soluciona el requerimiento 6
    """
    diccionariossectores=mp.newMap(numelements=23,
                                   maptype="PROBING",
                                   loadfactor=0.5)
    diccionarioaño=catalogo[año]
    llavesdicaño=mp.keySet(diccionarioaño)

    for subsectores in lt.iterator(llavesdicaño):
        diccionariosubsector=me.getValue(mp.get(diccionarioaño,subsectores))

        if mp.contains(diccionariossectores,diccionariosubsector["codigo sector economico"]):
            sector=me.getValue(mp.get(diccionariossectores,diccionariosubsector["codigo sector economico"]))
            sector["suma ingresos netos"]+=int(diccionariosubsector["suma ingresos netos"])
            sector["suma costos y gastos"]+=int(diccionariosubsector["suma costos y gastos"])
            sector["suma saldo por pagar"]+=int(diccionariosubsector["suma saldo por pagar"])
            sector["suma saldo a favor"]+=int(diccionariosubsector["suma saldo a favor"])

        else:
            nuevosector=newsector(diccionariosubsector)
            mp.put(diccionariossectores,diccionariosubsector["codigo sector economico"],nuevosector)

        formatoainsertar={"codigo subsector economico":diccionariosubsector["codigo subsector economico"],
                          "nombre subsector economico":diccionariosubsector["nombre subsector economico"],
                          "suma ingresos netos":diccionariosubsector["suma ingresos netos"],
                          "suma costos gastos":diccionariosubsector["suma costos y gastos"],
                          "suma saldo por pagar":diccionariosubsector["suma saldo por pagar"],
                          "suma saldo a favor":diccionariosubsector["suma saldo a favor"]}
        lt.addLast(me.getValue(mp.get(diccionariossectores,diccionariosubsector["codigo sector economico"]))["subsectores"],formatoainsertar)

    valoresdicsectores=mp.valueSet(diccionariossectores)

    merg.sort(valoresdicsectores,cmpingresosnetos)
```

Descripción

En un principio se crea un nuevo mapa cuyas llaves corresponderán a los códigos de los sectores del año indicado y sus valores serán diccionarios donde se realizará la sumatoria de los criterios de interés. Para el cumplimiento del requerimiento se iteraron las parejas llave-valor que componen el mapa correspondiente al año que indica el usuario, a medida que se iteran se busca conocer el sector económico al cual pertenece el subsector que se está iterando. Si este es una llave que se encuentra en el nuevo mapa creado, se buscara dicho valor y se harán las sumatorias correspondientes, posteriormente se guardará dicho subsector en la lista de subsectores del sector al que pertenece. Una vez se realiza este procedimiento con todos los subsectores, se procede a extraer los valores del mapa creado y organizarlos de acuerdo al "Total de ingresos netos", de ahí se obtiene el subsector con el mayor valor. Al obtenerlo, se organiza la lista de subsectores que posee, de acuerdo al criterio ya mencionado y se extraen el primer y último elemento de dicha lista.

Por último, se procede a organizar las listas de actividades con las que cuentan los dos subsectores seleccionados y se sacan el primer y último elemento.

Entrada	Para poder resolver el requerimiento es necesario que el usuario indique el año sobre el cual desea obtener información.

Salidas	El algoritmo devuelve 7 diccionarios: Uno con el sector económico con mayores Costos y gastos, los subsectores que más y menos aportaron, y las actividades que más aportaron y menos aportaron a sus respectivos subsectores.
Implementado (Sí/No)	Sí (Grupal)

Análisis de complejidad

Pasos	Complejidad
Instrucción: For "subsector" in lt.iterator(listasubsectores) Iterar los subsectores para conocer el sector donde pertenecen y realizar la sumatoria de los criterios a pedir.	O(N)
Instrucción: mp.put(diccionariosectores,diccionariosubsector["codigo sector economico nuevo sector"],nuevosector) Creación de los valores correspondientes al nuevo mapa.	O(1)
Instrucción: lt.addlast(me.getValue(mp.get(diccionariossectores,diccionariosubsector["codigo sector economico"]))["subsectores"],formatoainsertar) Organización de las actividades dentro del nuevo mapa creado	O(1)
Implementación del algoritmo de organización recursiva merge sort: merg.sort(valoresdicsectores,cmpingresosnetos) Organización de los sectores de acuerdo al criterio "ingresos netos"	O(1)
Implementación del algoritmo de organización recursiva merge sort: merg.sort(sectorescodigo["subsectores"],cmpingresosnetos) Organización de los subsectores correspondientes al sector hallado de acuerdo al criterio "ingresos netos"	O (NlogN)
Implementación del algoritmo de organización recursiva merge sort: merg.sort(diccompletossmax["actividades"],cmpingresosnetos2) Organización de las actividades correspondientes al subsector con el máximo valor "ingresos netos".	O (NlogN)
Implementación del algoritmo de organización recursiva merge sort: merg.sort(diccompletossmin["actividades"],cmpingresosnetos2) Organización de las actividades correspondientes al subsector con el mínimo valor "ingresos netos".	O (NlogN)
Procedimiento	***O(N(1+1)+3Nlog)***
Total	***O(2N+3Nlog)***

Complejidad requerimiento 6 **reto2**: ***O(2N+3Nlog)***

Complejidad requerimiento 6 **reto1**: No se implementó

Pruebas Realizadas

Para las pruebas se ingresó el año 2021. Las especificaciones del computador utilizado para la toma de datos son las siguientes:

Procesadores	**Intel(R) Core(TM) i5-8250u CPU @ 1.60GHZ 1.80GHZ**
Memoria RAM	8 GB

Sistema Operativo	Windows 11 Home

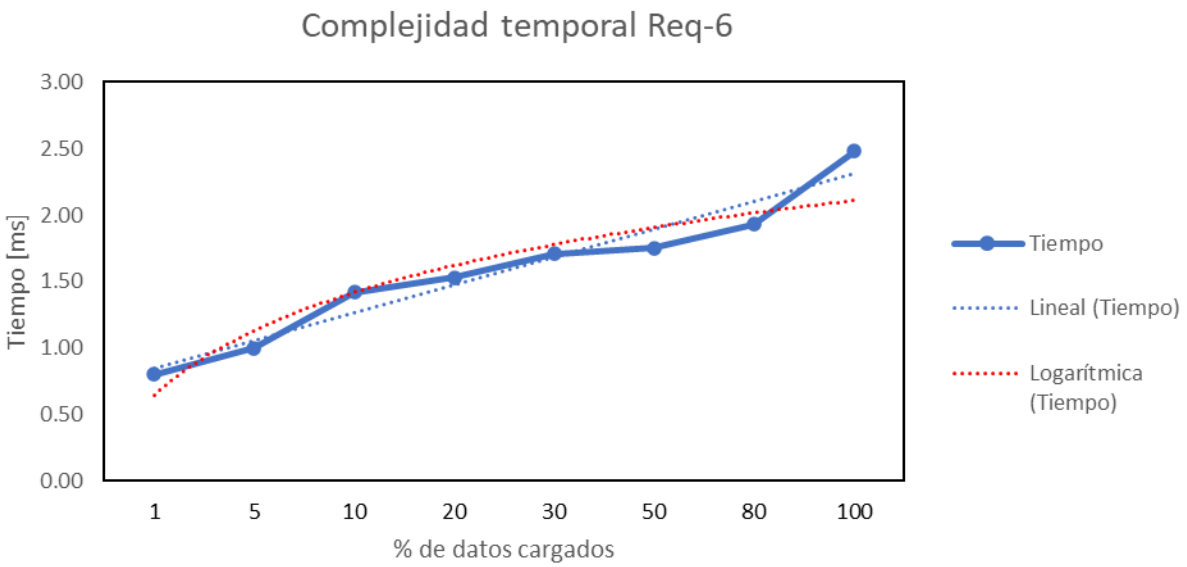
Tablas de datos

% datos cargados	Tiempo[ms]	Consumo de Memoria [Kb]
1	0.80	1,49
5	1.00	5,23
10	1.42	9,99
20	1.53	10,48
30	1.71	11,25
50	1.75	11,19
80	1.93	11,19
100	2.48	11,19

Gráficas

RETO 2

Complejidad temporal: *O(2N+3Nlog)*



Análisis

Para el requerimiento 6, se evidencia que la gráfica oscila entre una línea de tendencia logarítmica y otra lineal. En este caso no fue posible realizar la comparación con el reto 1 puesto que no se desarrolló en la primera ocasión. Lo anterior se debió a la existencia de problemas en la organización y en las iteraciones de las listas para sacar de formar jerárquica la información, es decir primero el sector, el subsector y luego las actividades. Este problema pudo evitarse en este reto a través del uso de los mapas, que permite guardar los datos con la organización necesaria. A medida que se crea el mapa de sectores se suman los criterios pedidos y se guarda una lista con los diccionarios de los subsectores ya armados con su propia sumatoria

de criterios. De esta forma, es posible organizar las parejas llave valor del nuevo mapa creado que contiene los sectores, y extraer mayor con costos y gastos de nómina, una vez hecho esto, el proceso se repite iterando la lista de subsector del sectore escogido y posteriormente la de las actividades de los subsectores máximos y mínimos. En este sentido, los datos se encuentran organizados, de tal forma que el de menor nivel (Las actividades), se encuentra contenido en el de mayor nivel como resultado, es fácil acceder a los datos requeridos y el costos modelo del reto se centra en la creación del nuevo mapa y en la organización de las listas, la cual, no es mayor a NlogN (Vviéndolo desde cualquiera de los casos).

Requerimiento <<7>>

Descripción

```
def req_7(catalogo,año,codigo,topn):
    """
    Función que soluciona el requerimiento 7
    """

    llavevalor=mp.get(catalogo[año],codigo)
    diccionario_subsector=me.getValue(llavevalor)
    actividades=diccionario_subsector["actividades"]
    merg.sort(actividades,cmp_costos_gastos)

    final=lt.newList("ARRAY_LIST")
    if lt.size(actividades)>=int(topn):
        actividades=lt.subList(actividades,1,int(topn))

    for actividad in lt.iterator(actividades):
        respuesta=formato_7(actividad)
        lt.addLast(final,respuesta)

    return final
```

Se saca del catálogo la llave que corresponde al subsector ingresado por el usuario y de esta se extrae y ordena la lista que contiene las actividades económicas. Finalmente se crea una sublista de las actividades ordenadas según los costos y gastos hasta el topn que se ingresa por parámetro.

Entrada	El catálogo con toda la información, el año que se desea buscar, el código del subsector que se quiere buscar y la cantidad de elementos que se quiere en el top.
Salidas	Lista con las actividades económicas que menos costos y gastos aportaron al subsector.
Implementado (Sí/No)	Si (Grupal)

Análisis de complejidad

Pasos	**Complejidad**

Extraer la llave del mapa correspondiente al subsector ingresado y la lista de actividades de este subsector	O(1)
Ordenar la lista de menor a mayor según el costos y gastos con Merge Sort	O(nlogn)
Se crea la sublista con la cantidad de elementos que quiere el usuario	O(n)
Se itera la lista para dejar los datos con el formato que se pide	O(n)
TOTAL	***O(nlogn)+O(n)+O(1)***

Complejidad requerimiento 7 **reto2**: ***O(nlogn)+O(n)+O(1)***

Complejidad requerimiento 7 **reto1**: $O(N^2)$

Pruebas Realizadas

Se hicieron las pruebas con el año 2017, el subsector económico 3 y se pidió el top 4. Las características del computador que se utilizó son las siguientes:

Procesadores	**Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

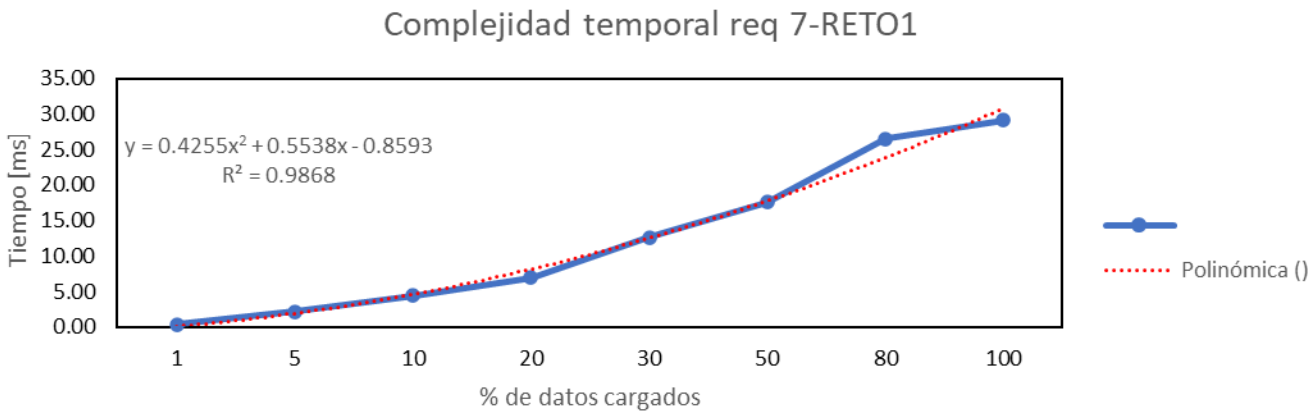
Tablas de datos

Entrada	**Tiempo (ms)**	**Memoria(kB)**
small	1,56	1,09
5 pct	4,45	1,76
10 pct	4,89	1,88
20 pct	5,09	2,01
30 pct	6,03	2,13
50 pct	6,88	2,26
80 pct	7,43	2,26
large	8,29	2,38

Gráficas

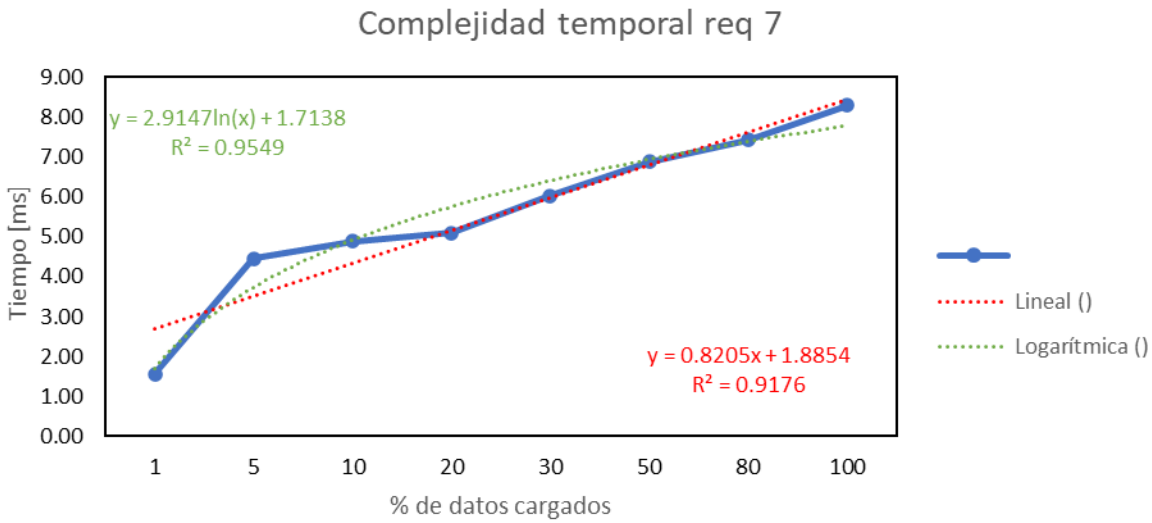
RETO 1:

Complejidad: $O(N^2)$



RETO 2:

Complejidad: O(nlogn)



Análisis

Según el análisis paso a paso realizado anteriormente, el requerimiento 7 tiene una complejidad O(NlogN) por implementar el algoritmo de ordenamiento merge sort. Esto se corrobora en la gráfica, pues, si bien tanto la regresión logarítmica como la lineal tienen un R mayor a 0,9, la primera se ajusta de mejor manera a los datos que la segunda.

En cuanto a la comparación con el requerimiento 7 del reto 1, se logró reducir la complejidad considerablemente de $O(N^2)$ a O(NlogN), además de disminuir los tiempos hasta 4 veces. Esto se debe a que en este reto no se tiene que iterar para obtener los valores a comparar, únicamente se itera para ordenar y darle el formato pedido a los datos.

Finalmente, se tuvo un consumo de memoria mínimo a comparación de los demás requerimientos, lo cual se debe a que únicamente se creó una estructura adicional a las que ya se tenían (la lista final). Además, sin importar el tamaño del archivo se tenían pocos datos a trabajar, pues se seleccionaban únicamente las actividades económicas de 1 subsector económico de 1 año desde el inicio.

Requerimiento <<8>>

Descripción

```python
def req_8(catalog,año,topn):
    mapaaño=catalog[año]
    listallaves=mp.keySet(mapaaño)
    merg.sort(listallaves,codigosubsector)
    listasubsectores=lt.newList(datastructure="ARRAY_LIST")
    listaactividades=lt.newList(datastructure="ARRAY_LIST")

    if lt.size(listallaves)>12:
        lista3primeros=lt.subList(listallaves,1,3)
        lista3ultimos=lt.subList(listallaves,lt.size(listallaves)-2,3)

        for subsectores in lt.iterator(lista3primeros):
            diccionarioatrabajar=me.getValue(mp.get(mapaaño,subsectores))
            copia=diccionarioatrabajar.copy()

            del copia["actividades"]
            del copia["suma descuentos tributarios"]
            del copia["suma costos y gastos nomina"]
            del copia["suma total retenciones"]
            del copia["maximo saldo por pagar"]
            del copia["maximo saldo a favor"]
            del copia["actividad maxima saldo por pagar"]
            del copia["actividad maxima saldo a favor"]
            lt.addLast(listasubsectores,copia)

            merg.sort(diccionarioatrabajar["actividades"],cmpimpuestosacargo)

            if lt.size(diccionarioatrabajar["actividades"])<6:
                for actividades in lt.iterator(diccionarioatrabajar["actividades"]):
                    formato={"codigo sector economico":actividades["codigo sector economico"],
                             "nombre sector economico":actividades["nombre sector economico"],
                             "codigo subsector economico":actividades["codigo subsector economico"],
                             "nombre subsector economico":actividades["nombre subsector economico"],
                             "codigo actividad economica":actividades["codigo actividad economica"],
                             "nombre actividad economica":actividades["nombre actividad economica"],
                             "total impuestos a cargo":actividades["impuestos a cargo"],
                             "total costos y gastos":actividades["total costos gastos"],
                             "total saldo por pagar":actividades["saldo a pagar"],
                             "total saldo a favor":actividades["saldo a favor"]}
                    lt.addLast(listaactividades,formato)

```

```
            else:
                sublista=lt.subList(diccionarioatrabajar["actividades"],1,int(topn))
                for actividades in lt.iterator(sublista):
                    formato={"codigo sector economico":actividades["codigo sector economico"],
                            "nombre sector economico":actividades["nombre sector economico"],
                            "codigo subsector economico":actividades["codigo subsector economico"],
                            "nombre subsector economico":actividades["nombre subsector economico"],
                            "codigo actividad economica":actividades["codigo actividad economica"],
                            "nombre actividad economica":actividades["nombre actividad economica"],
                            "total impuestos a cargo":actividades["impuestos a cargo"],
                            "total costos y gastos":actividades["total costos gastos"],
                            "total saldo por pagar":actividades["saldo a pagar"],
                            "total saldo a favor":actividades["saldo a favor"]}

                    lt.addLast(listaactividades,formato)


    else:

        for subsectores in lt.iterator(listallaves):
            diccionarioatrabajar=me.getValue(mp.get(mapaaño,subsectores))
            copia=diccionarioatrabajar.copy()

            del copia["actividades"]
            del copia["suma descuentos tributarios"]
            del copia["suma costos y gastos nomina"]
            del copia["suma total retenciones"]
            del copia["maximo saldo por pagar"]
            del copia["maximo saldo a favor"]
            del copia["actividad maxima saldo por pagar"]
            del copia["actividad maxima saldo a favor"]
            lt.addLast(listasubsectores,copia)

            merg.sort(diccionarioatrabajar["actividades"],cmpimpuestosacargo)

            if lt.size(diccionarioatrabajar["actividades"])<6:
                for actividades in lt.iterator(diccionarioatrabajar["actividades"]):
                    formato={"codigo sector economico":actividades["codigo sector economico"],
                            "nombre sector economico":actividades["nombre sector economico"],
                            "codigo subsector economico":actividades["codigo subsector economico"],
                            "nombre subsector economico":actividades["nombre subsector economico"],
                            "codigo actividad economica":actividades["codigo actividad economica"],
                            "nombre actividad economica":actividades["nombre actividad economica"],
                            "total impuestos a cargo":actividades["impuestos a cargo"],
                            "total costos y gastos":actividades["total costos gastos"],
                            "total saldo por pagar":actividades["saldo a pagar"],
                            "total saldo a favor":actividades["saldo a favor"]}
                    lt.addLast(listaactividades,formato)
```

```
            else:
                sublista=lt.subList(diccionarioatrabajar["actividades"],1,int(topn))
                for actividades in lt.iterator(sublista):
                    formato={"codigo sector economico":actividades["codigo sector economico"],
                            "nombre sector economico":actividades["nombre sector economico"],
                            "codigo subsector economico":actividades["codigo subsector economico"],
                            "nombre subsector economico":actividades["nombre subsector economico"],
                            "codigo actividad economica":actividades["codigo actividad economica"],
                            "nombre actividad economica":actividades["nombre actividad economica"],
                            "total impuestos a cargo":actividades["impuestos a cargo"],
                            "total costos y gastos":actividades["total costos gastos"],
                            "total saldo por pagar":actividades["saldo a pagar"],
                            "total saldo a favor":actividades["saldo a favor"]}

                    lt.addLast(listaactividades,formato)

    return listasubsectores,listaactividades
```

Para resolver este requerimiento, se obtiene una lista con las llaves del mapa correspondiente al año indicado por el usuario. En primera instancia se revisa cuántos elementos hay en dicha lista, para luego proceder a sacar una sublista de los primero 3 y últimos 3 subsectores e iterarlas o iterar todos los subsectores de la lista (este paso depende de cuantos subsectores halla en la lista). A medida que se realizan las interacciones por subsector, se organizan la lista de actividades correspondientes de cada subsector y se saca una sublista de acuerdo al número indicado por el usuario, luego se itera dicha sublista para agregarla a una lista general que guarda el Top N de actividades de cada uno de los subsectores.

Entrada	Para resolver el requerimiento es necesario, aparte del catálogo, que el usuario indique el año, y número del top que se quiere realizar alrededor de las actividades económicas.
Salidas	El algoritmo retorno dos listas, una con los subsectores y las sumatorias correspondientes a los criterios solicitados y la otra con las topn actividades por cada subsector.
Implementado (Sí/No)	Si. (Grupal)

Análisis de complejidad

Pasos	**Complejidad**
Organización de las llaves de acuerdo al código del sector económico.	O(NlogN)
Creación de las listas a retornar donde se guardarán los subsectores y las actividades.	O(1)
Comprobación del número de sectores en el map del año indicado.	O(1)
Iteración de los subsectores del mapa indicado.	O(N) a 2 O(N)
Organización de la lista de actividades de cada subsector iterado	O(NlogN)
Extraer las sublista con el tamaño indicado por el usuario, de las actividades de cada subsector	O(1)
Ajustar el formato de las actividades de acuerdo a los criterios pedidos	O(N)
Procedimiento:	O(NlogN +2 (1) + 2O(N (N+1 +NlogN))
TOTAL	***O(2N^{2}LogN+2(N^2)+2N+NlogN)***

Complejidad requerimiento 8 **reto2**: (N^{2}Log N)

Complejidad requerimiento 8 **reto1**: No fue implementado

Pruebas Realizadas

Las pruebas se hicieron ingresando el año 2018 y pidiendo identificar 3 actividades económicas. Las especificaciones del computador que se utilizó para la toma de datos son las siguientes:

Procesadores	**Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

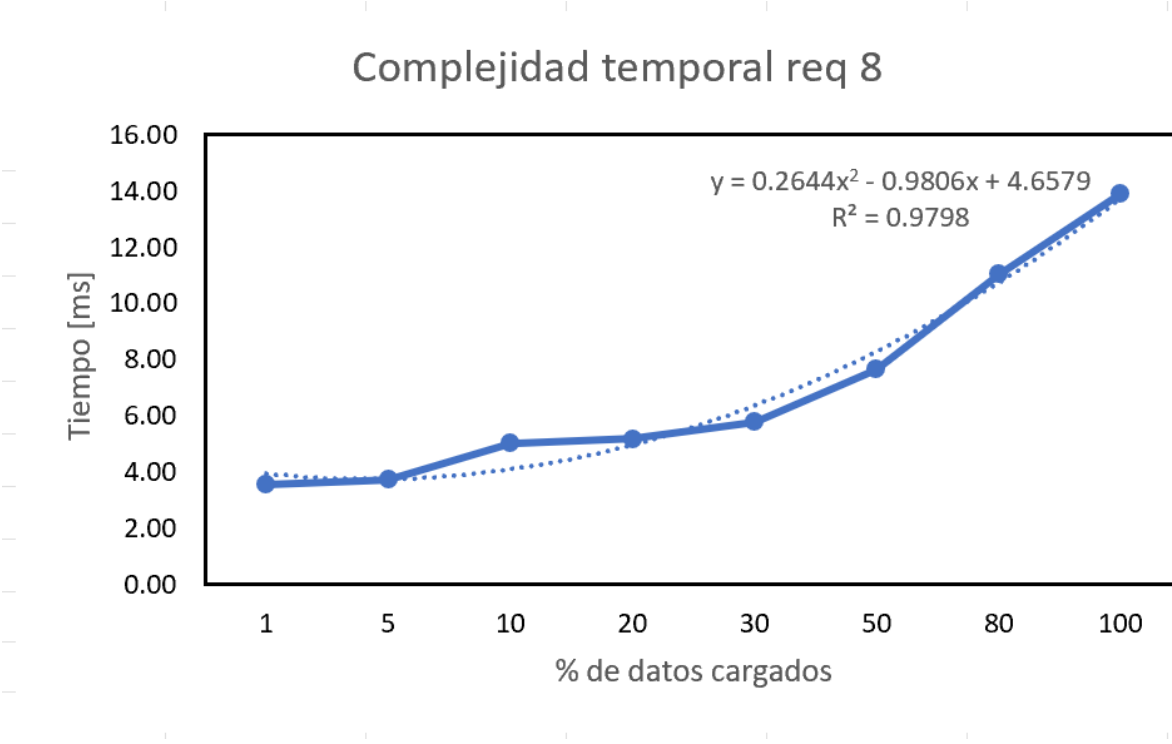
Tablas de datos

% datos cargados	Tiempo[ms]	Consumo de Memoria [Kb]
1	3.57	46.69
5	3.73	57.91
10	5.03	54.82
20	5.17	56.63
30	5.78	56.69
50	7.65	56.19
80	11.05	56.58
100	13.92	57.49

Gráficas

RETO 2:

Complejidad temporal: $N^2 Log\ N$



Análisis

En este caso la forma como se encuentran organizados los datos dificulta el acceso a los datos necesarios, esto debido a que no nos pide un único dato sino múltiples, los cuales deben seguir una organización. Los mapas utilizados no se encuentran organizados y el contenido de los valores tampoco; organizarlos aumenta

la complejidad de forma considerable pues es necesario, no solo el uso de algoritmos que realicen este proceso (merge sort), los cuales generan el O(nlogn) de la complejidad, sino también dobles ciclos, ya que la lista de actividades se encuentra contenidas dentro de los valores de cada subsector, los que explican el N^2 de la complejidad. En este sentido el algoritmo se vuelve ineficiente.

La gráfica presentada, refleja lo mencionado anteriormente, los datos se ajustan correctamente a un modelo polinómico grado 2. Indicando que a medida que aumentan el tamaño de los archivos o el número de datos, el proceso de búsqueda se alarga y, por lo tanto, el tiempo de respuesta del algoritmo incrementa de forma cuadrática.