# ANÁLISIS DEL RETO

Juan Sebastián Ardila López, Cod 202110171, js.ardilal1@uniandes.edu.co
Daniel Vargas Mayorga, Cod 201822068, d.vargasm@uniandes.edu.co
Daniela Echavarria Yepes, Cod 202011348, d.echavarria@uniandes.edu.co

## Requerimiento 1

### Descripción

```
def req_1(data_structs,anio,sector):
    """
    Función que soluciona el requerimiento 1
    """
    #Se obtiene la lista de las actividades de ese año
    list_activities_year=mp.getValue(mp.get(data_structs,anio))
    #Se crea una lista con las actividades que pertenecen alsector ingresado
    list_activities_sector=lt.newList(datastructure="ARRAY_LIST")
    #Se añaden los elementos que sean de esa actividad y de ese año a la lista list_activities_sector
    for actividad in lt.iterator(list_activities_year):
        if actividad["info"]["Código sector económico"]==sector:
            lt.addLast(list_activities_sector,actividad)
    #Se ordena la lista por merge sort con el criterio de saldo a pagar
    merg.sort(list_activities_sector, cmp_saldo_pagar)
    #Se retorna el elemento con mayor saldo a pagar
    return lt.firstElement(list_activities_sector)
```

Este requerimiento se encarga de retornar la actividad con el mayor saldo a pagar de un año y sector económico específico.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, año, sector |
| **Salidas** | Información de la actividad económica que mayor saldo a pagar tuvo de ese año |
| **Implementado (Sí/No)** | Si |

### Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener la lista de las actividades del año | O(1) |
| Crear la lista de las actividades del sector de dicho año | O(1) |
| Recorrer las actividades de ese año y añadir al final si es necesario | O(m) |
| Filtrar la lista de actividades | O(mlog(m)) |
| Retornar el primer elemento de la lista | O(1) |
| ***TOTAL*** | ***O(m+mlog(m))*** |

En este caso m corresponde a la cantidad de actividades de dicho año.

## Pruebas Realizadas

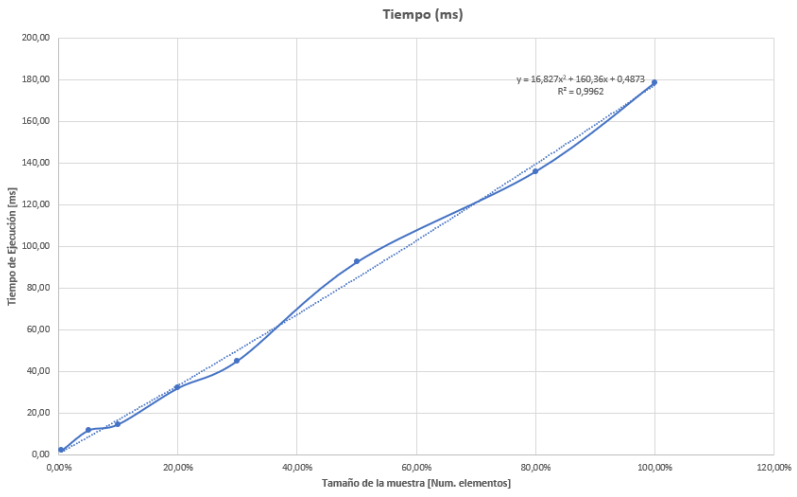
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron año 2016 y el sector 1.

| **Procesadores** | |
|---|---|
| **Memoria RAM** | 8GB |
| **Sistema operativo** | Microsoft Windows 10 Home Single Language |

## Tablas de datos.

| Muestra | Salida | Tiempo (ms) | Memoria utilizada (kB) |
|---|---|---|---|
| Small, 2016 | Actividad: 130 | 3,6 | 0,1875 |
| 5 pct, 2016 | Actividad:125 | 3,8 | 0,1875 |
| 10 pct, 2016 | Actividad:125 | 1,94 | 0,1875 |
| 20 pct, 2016 | Actividad:125 | 0,18 | 0,1875 |
| 30 pct, 2016 | Actividad:125 | 2,15 | 0,1875 |
| 50 pct, 2016 | Actividad:125 | 3,07 | 0,1875 |
| 80 pct, 2016 | Actividad:125 | 2,49 | 0,1875 |
| large, 2016 | Actividad:125 | 4,06 | 0,1875 |

## Graficas



## Análisis

A partir de la gráfica se observa un cambio casi lineal, O(nlogn). La razón principal de porque esto no ocurre es al momento de realizar el ordenamiento, debido a que se utiliza MergeSort para organizar los datos, y este algoritmo en su peor caso genera esta grafica en esta complejidad.

# Requerimiento 2

## Descripción

```python
def req_2(data_structs,anio,sector):
    """
    Función que soluciona el requerimiento 2
    """
    #Se obtiene la lista de las actividades de ese año
    list_activities_year=me.getValue(mp.get(data_structs,anio))
    #Se crea una lista con las actividades que pertenecen alsector ingresado
    list_activities_sector=lt.newList(datastructure="ARRAY_LIST")
    #Se añaden los elementos que sean de esa actividad y de ese año a la lista list_activities_sector
    for actividad in lt.iterator(list_activities_year):
        if actividad["info"]["Código sector económico"]==sector:
            lt.addLast(list_activities_sector,actividad)
    #Se ordena la lista por merge sort con el criterio de saldo a favor
    merg.sort(list_activities_sector, cmp_saldo_favor)
    #Se retorna el elemento con mayor saldo a pagar
    return lt.firstElement(list_activities_sector)
```

Este requerimiento se encarga de retornar la actividad con el mayor saldo a favor de un año y sector económico específico.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, año, sector |
| **Salidas** | Información de la actividad económica que mayor saldo a favor tuvo de ese año |
| **Implementado (Sí/No)** | Si |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener la lista de las actividades del año | O(1) |
| Crear la lista de las actividades del sector de dicho año | O(1) |
| Recorrer las actividades de ese año y añadir al final si es necesario | O(m) |
| Filtrar la lista de actividades | O(mlog(m)) |
| Retornar el primer elemento de la lista | O(1) |
| ***TOTAL*** | ***O(m+mlog(m))*** |

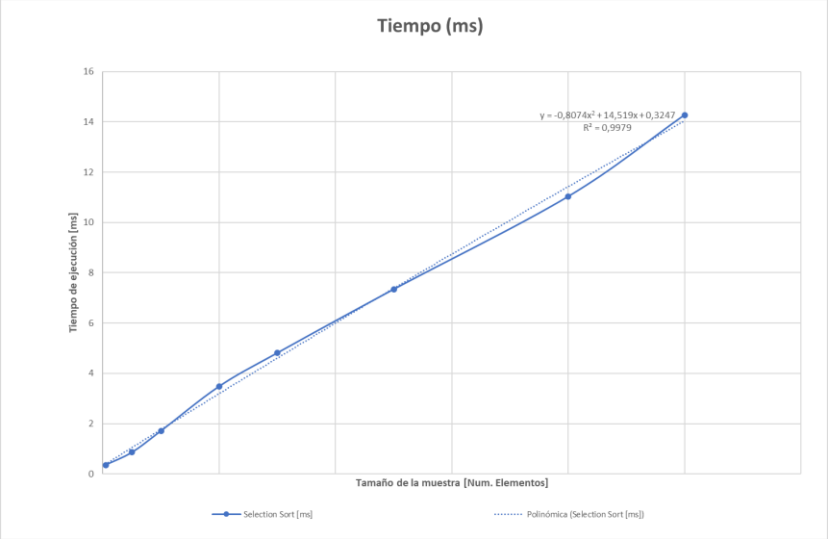
## Pruebas Realizadas

| **Procesadores** | **11th Gen Intel(R) Core (TM) i7-1165G7 @ 2.80GHz 2.80 GHz** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 11 Home-64 bits |

## Tablas de datos

| Muestra | Salida | Tiempo (ms) | Memoria utilizada (kB) |
|---|---|---|---|
| Small,2021 | Actividad: 2750 | 0,36 | 0,1875 |
| 5 pct, 2021 | Actividad: 1811 | 0,86 | 0,1875 |
| 10 pct, 2021 | Actividad: 1811 | 1,71 | 0,1875 |
| 20 pct, 2021 | Actividad: 1709 | 3,48 | 0,1875 |
| 30 pct, 2021 | Actividad: 2023 | 4,815 | 0,1875 |
| 50 pct, 2021 | Actividad: 3312 | 7,34 | 0,1875 |
| 80 pct, 2021 | Actividad: 3312 | 11,03 | 0,1875 |
| large, 2021 | Actividad: 3312 | 14,27 | 0,1875 |

## Graficas




## Análisis

Se puede observar un comportamiento lineal y los cambios y variaciones pueden ser causa de que el orden de crecimiento es O(nlog(n)), la razón de este es que se utiliza MergeSort() para organizar una lista de elementos y este orden corresponde al peor caso.

# Requerimiento 3

## Descripción

```
def req_3(data_structs,anio):
    """
    Función que soluciona el requerimiento 3
    """
    #Se obtiene la lista de las actividades de ese año
    list_activities_year=mo.getValue(mp.get(data_structs,anio))
    #Se crea un mapa para cada subsector económico
    map_subsector=mp.newMap(22,
                            maptype="PROBING",
                            loadfactor=0.8)
    #Se recorre la lista de registros de ese año
    for actividad in lt.iterator(list_activities_year):
        #Se crea el mapa del subsector si este no existe
        if actividad["info"]["Código subsector económico"] not in lt.iterator(mp.keySet(map_subsector)):
            #Se crea un mapa con la información de cada subsector que será el valor de map_subsector
            map_subsector_value=mp.newMap(9,
                                          maptype='PROBING',
                                          loadfactor=0.8)
            mp.put(map_subsector_value,"Código sector económico",actividad["info"]["Código sector económico"])
            mp.put(map_subsector_value,"Nombre sector económico",actividad["info"]["Nombre sector económico"])
            mp.put(map_subsector_value,"Código subsector económico",actividad["info"]["Código subsector económico"])
            mp.put(map_subsector_value,"Nombre subsector económico",actividad["info"]["Nombre subsector económico"])
            mp.put(map_subsector_value,"Total de retenciones del subsector económico",int(actividad["info"]["Total retenciones"]))
            mp.put(map_subsector_value,"Total ingresos netos del subsector económico",int(actividad["info"]["Total ingresos netos"]))
            mp.put(map_subsector_value,"Total costos y gastos del subsector económico",int(actividad["info"]["Total costos y gastos"]))
            mp.put(map_subsector_value,"Total saldo a pagar del subsector económico",int(actividad["info"]["Total saldo a pagar"]))
            mp.put(map_subsector_value,"Total saldo a favor del subsector económico",int(actividad["info"]["Total saldo a favor"]))
            #Se crea una lista que servirá para guardar todas las actividades de ese subsector
            lt_activities_in_subsector=lt.newList(datastructure="ARRAY_LIST")
            #Se añade la actividad que tenemos en este momento
            lt.addLast(lt_activities_in_subsector,actividad)
            #Se añade dicha lista al mapa
            mp.put(map_subsector_value,"Actividades del subsector",lt_activities_in_subsector)
            #Se añade la tabla de información del subsector al índice del subsector
            mp.put(map_subsector,actividad["info"]["Código subsector económico"],map_subsector_value)

        #Si el valor del subsector ya existe
        else:
            #Encontrar valores del subsector
            map_subsector_value=mo.getValue(mp.get(map_subsector,actividad["info"]["Código subsector económico"]))
            total_retenciones=mo.getValue(mp.get(map_subsector_value,"Total de retenciones del subsector económico"))
            value_ingresos=mo.getValue(mp.get(map_subsector_value,"Total ingresos netos del subsector económico"))
            value_gastos=mo.getValue(mp.get(map_subsector_value,"Total costos y gastos del subsector económico"))
            value_saldo_pagar=mo.getValue(mp.get(map_subsector_value,"Total saldo a pagar del subsector económico"))
            value_saldo_favor=mo.getValue(mp.get(map_subsector_value,"Total saldo a favor del subsector económico"))
            lt_activities_in_subsector=mo.getValue(mp.get(map_subsector_value,"Actividades del subsector"))
            lt.addLast(lt_activities_in_subsector,actividad)

            #Añadir valores al subsector
            mp.put(map_subsector_value,"Total de retenciones del subsector económico",int(actividad["info"]["Total retenciones"])+ total_retenciones)
            mp.put(map_subsector_value,"Total ingresos netos del subsector económico",value_ingresos+int(actividad["info"]["Total ingresos netos"]))
            mp.put(map_subsector_value,"Total costos y gastos del subsector económico",value_gastos+int(actividad["info"]["Total costos y gastos"]))
            mp.put(map_subsector_value,"Total saldo a pagar del subsector económico",value_saldo_pagar+int(actividad["info"]["Total saldo a pagar"]))
            mp.put(map_subsector_value,"Total saldo a favor del subsector económico",value_saldo_favor+int(actividad["info"]["Total saldo a favor"]))
            mp.put(map_subsector_value,"Actividades del subsector",lt_activities_in_subsector)

    #Encontrar el sector con el mínimo de retenciones
    min_retenciones = float('inf')
    min_sectores = None

    for sector in lt.iterator(mp.keySet(map_subsector)):
        #Se obtiene la información de cada subsector
        info_subsector= mo.getValue(mp.get(map_subsector,sector))
        #Se revisa si el valor del subsect
        if  mo.getValue(mp.get(info_subsector,"Total de retenciones del subsector económico"))< min_retenciones:
            min_retenciones =  mo.getValue(mp.get(info_subsector,"Total de retenciones del subsector económico"))
            min_sectores = info_subsector

    #Obtener las actividades del subsector con el mínimo de retenciones
    actividades_subsector= mo.getValue(mp.get(min_sectores,"Actividades del subsector"))
    #Organizar actividades de menor a mayor con respecto a sus retenciones
    merg.sort(actividades_subsector,cmp_retenciones)

    if lt.size(actividades_subsector) > 6:
        choosen_activities= lt.subList(actividades_subsector,1,3)
        menores= lt.subList(actividades_subsector,lt.size(actividades_subsector)-2,3)
        for elemento in menores:
            lt.addLast(choosen_activities,elemento)
    else:
        choosen_activities=actividades_subsector
```

Este requerimiento recibe como parámetro la estructura de datos del modelo y el año de interés, en este caso se crea un mapa en el que las llaves son subsectores, y su valor es la información de interés del subsector (la suma de los aportes de cada actividad) sumado a una lista de actividades de cada subsector. Se verifica si el subsector ya existe en el mapa, en caso de que no se inicializa, en caso de que si se actualiza. Posteriormente se obtiene el subsector con el menor total de retenciones y se filtran las actividades de dicho subsector.

| | |
|---|---|
| **Entrada** | Estructura de datos y año de interés |
| **Salidas** | Subsector y actividades |
| **Implementado (Sí/No)** | Si, implementado por Daniela Echavarría |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| Pasos | Complejidad |
|---|---|
| Obtener la lista de las actividades de dicho año | O(1) |
| Inicializar el mapa de subsectores | O(1) |
| Recorrer las actividades del año | O(m) |
| Inicializar el mapa del subsector si no existe | O(1) |
| Actualizar el mapa del subsector si existe | O(1) |
| Obtener el subsector de interés | O(k) |
| Se hace un sort de las actividades del subsector | O(plog(p)) |
| ***TOTAL*** | ***O(m+plog(p))*** |

En este caso m es el numero de actividades del año de interés, y p es el número de actividades del subsector que tuvo más descuentos, k número de subsectores que corresponden al año ingresado

## Pruebas Realizadas

| **Procesadores** | **11th Gen Intel(R) Core (TM) i7-1165G7 @ 2.80GHz 2.80 GHz** |
|---|---|
| **Memoria RAM** | 8 GB |

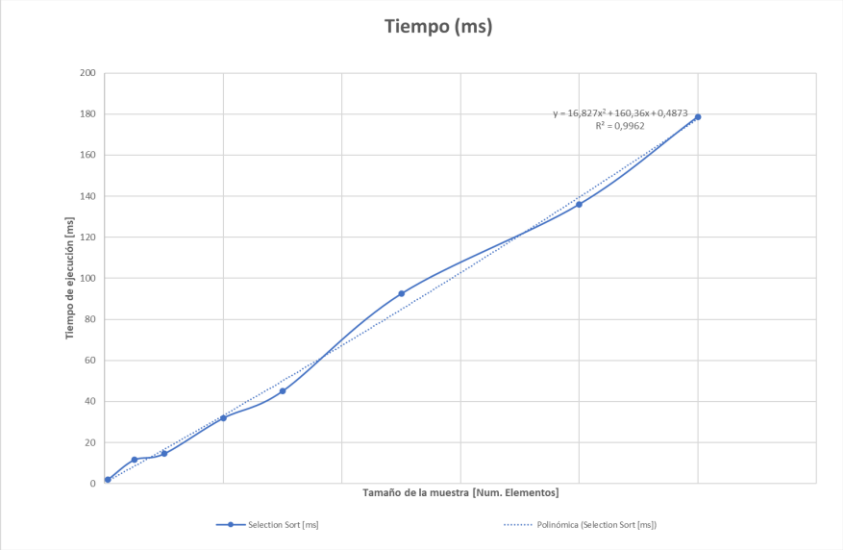
### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| Muestra | Salida | Tiempo (ms) | Memoria utilizada (kB) |
|---|---|---|---|
| Small,2021 | Subsector:3<br>Actividades: 3092,3210 | 1,87 | 2,03 |
| 5 pct, 2021 | Subsector:15<br>Actividades:8421,8424 | 11,59 | 12,40 |
| 10 pct, 2021 | Subsector:15<br>Actividades: 8421,8424 | 14,53 | 13,26 |
| 20 pct, 2021 | Subsector:15<br>Actividades: 8421,8424 | 31,92 | 15,21 |
| 30 pct, 2021 | Subsector:15<br>Actividades: 8421,8424 | 45,05 | 19,14 |
| 50 pct, 2021 | Subsector:5<br>Actividades: 3812,3700 | 92,45 | 21,80 |
| 80 pct, 2021 | Subsector:20<br>Actividades: 9700 | 135,96 | 22,19 |
| large, 2021 | Subsector:20<br>Actividades: 9810,9820,9700 | 178,6 | 19,22 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

En este caso se puede observar que el algoritmo presenta principalmente una complejidad lineal que varía en algunos casos, esto se debe principalmente debido a que nuestra complejidad está definida en términos del número de elementos del año de interés y el número de actividades del subsector, y al estar parametrizado en dos variables, se puede aumentar la tasa de crecimiento o disminuir en algunos intervalos. Además, se puede presentar una variación debido a que en la complejidad se tiene un factor log(n).

# Requerimiento 4

```
def req_4(data_structs, anio):
    """
    Función que soluciona el requerimiento 4
    """
    #Se obtiene la lista de las actividades de ese año
    list_activities_year=mo.getValue(mp.get(data_structs,anio))
    #Se crea un mapa para cada subsector económico
    map_subsector=mp.newMap(22,
                            maptype="PROBING",
                            loadFactor=0.8)
    #Se recorre la lista de registros de ese año
    for actividad in lt.iterator(list_activities_year):
        #Se crea el mapa del subsector si este no existe
        if not mp.contains(map_subsector,actividad["info"]["Código subsector económico"]):
            #Se crea un mapa con la información de cada subsector que será el valor de map_subsector
            map_subsector_value=mp.newMap(9,
                                maptype="PROBING",
                                loadFactor=0.8)
            mp.put(map_subsector_value,"Código sector económico",actividad["info"]["Código sector económico"])
            mp.put(map_subsector_value,"Nombre sector económico",actividad["info"]["Nombre sector económico"])
            mp.put(map_subsector_value,"Código subsector económico",actividad["info"]["Código subsector económico"])
            mp.put(map_subsector_value,"Nombre subsector económico",actividad["info"]["Nombre subsector económico"])
            mp.put(map_subsector_value,"Total de costos y gastos de nomina del subsector económico",int(actividad["info"]["Costos y gastos nómina"]))
            mp.put(map_subsector_value,"Total ingresos netos del subsector económico",int(actividad["info"]["Total ingresos netos"]))
            mp.put(map_subsector_value,"Total costos y gastos del subsector económico",int(actividad["info"]["Total costos y gastos"]))
            mp.put(map_subsector_value,"Total saldo a pagar del subsector económico",int(actividad["info"]["Total saldo a pagar"]))
            mp.put(map_subsector_value,"Total saldo a favor del subsector económico",int(actividad["info"]["Total saldo a favor"]))
            #Se crea una lista que servirá para guardar todas las actividades de ese subsector
            lt_activities_in_subsector=lt.newList(datastructure="ARRAY_LIST")
            #Se añade la actividad que tenemos en este momento
            lt.addLast(lt_activities_in_subsector,actividad)
            #Se añade dicha lista al mapa
            mp.put(map_subsector_value,"Actividades del subsector",lt_activities_in_subsector)
            #Se añade la tabla de información del subsector al índice del subsector
            mp.put(map_subsector,actividad["info"]["Código subsector económico"],map_subsector_value)
        #Si el valor del subsector ya existe
        else:
            #Encontrar valores del subsector
            map_subsector_value=mo.getValue(mp.get(map_subsector,actividad["info"]["Código subsector económico"]))
            total_costos_y_gastos_de_nomina=mo.getValue(mp.get(map_subsector_value,"Total de costos y gastos de nomina del subsector económico"))
            value_ingresos=mo.getValue(mp.get(map_subsector_value,"Total ingresos netos del subsector económico"))
            value_gastos=mo.getValue(mp.get(map_subsector_value,"Total costos y gastos del subsector económico"))
            value_saldo_pagar=mo.getValue(mp.get(map_subsector_value,"Total saldo a pagar del subsector económico"))
            value_saldo_favor=mo.getValue(mp.get(map_subsector_value,"Total saldo a favor del subsector económico"))
            lt_activities_in_subsector=mo.getValue(mp.get(map_subsector_value,"Actividades del subsector"))
            lt.addLast(lt_activities_in_subsector,actividad)
            #Añadir valores al subsector
            mp.put(map_subsector_value,"Total de costos y gastos de nomina del subsector económico",float(actividad["info"]["Costos y gastos nómina"])+ total_costos_y_gastos_de_nomina)
            mp.put(map_subsector_value,"Total ingresos netos del subsector económico",value_ingresos+float(actividad["info"]["Total ingresos netos"]))
            mp.put(map_subsector_value,"Total costos y gastos del subsector económico",value_gastos+float(actividad["info"]["Total costos y gastos"]))
            mp.put(map_subsector_value,"Total saldo a pagar del subsector económico",value_saldo_pagar+float(actividad["info"]["Total saldo a pagar"]))
            mp.put(map_subsector_value,"Total saldo a favor del subsector económico",value_saldo_favor+float(actividad["info"]["Total saldo a favor"]))
            mp.put(map_subsector_value,"Actividades del subsector",lt_activities_in_subsector)
    #Encontrar el sector con el mínimo de retenciones
    costos_gastos_nomina = -1
    max_sectores = None
    for sector in lt.iterator(mp.keySet(map_subsector)):
        #Se obtiene la información de cada subsector
```

```
#Encontrar el sector con el mínimo de retenciones
costos_gastos_nomina = -1
max_sectores = None
for sector in lt.iterator(mp.keySet(map_subsector)):
    #Se obtiene la información de cada subsector
    info_subsector= mo.getValue(mp.get(map_subsector,sector))
    #Se revisa si el valor del subsect
    if  mo.getValue(mp.get(info_subsector,"Total de costos y gastos de nomina del subsector económico")) > costos_gastos_nomina:
        costos_gastos_nomina =  mo.getValue(mp.get(info_subsector,"Total de costos y gastos de nomina del subsector económico"))
        max_sectores = info_subsector
#Obtener las actividades del subsector con el mínimo de retenciones
actividades_subsector= mo.getValue(mp.get(max_sectores,"Actividades del subsector"))
#Organizar actividades de menor a mayor con respecto a sus retenciones
merg.sort(actividades_subsector,cmp_costosygastos)

if lt.size(actividades_subsector) > 6:
    choosen_activities= lt.sublist(actividades_subsector,1,3)
    i = lt.size(actividades_subsector)
    while i > lt.size(actividades_subsector)-2:
        lt.addLast(choosen_activities, lt.getElement(actividades_subsector, i))
        i-=1
else:
    choosen_activities=actividades_subsector
return max_sectores,choosen_activities
```

## Descripción

Este requerimiento recibe como parámetro la estructura de datos del modelo y el año de interés, en este caso se crea un mapa en el que las llaves son subsectores, y su valor es la información de interés del subsector sumado a una lista de actividades de cada subsector. Se comprueba si el subsector ya existe en el mapa, en caso de que no exista este subsector en el mapa, se inicializa, en caso de que ya exista, este se actualizará. Después, se obtiene el subsector con el mayor costo y gasto de nómina para el año que se especifique.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, año |

| | |
|---|---|
| **Salidas** | Mapa del subsector con mayores costos y gastos, lista de actividades del subsector filtrada |
| **Implementado (Sí/No)** | Si, implementado por Daniel Vargas |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener la lista de las actividades de dicho año | O(1) |
| Inicializar el mapa de subsectores | O(1) |
| Recorrer las actividades del año | O(m) |
| Inicializar el mapa del subsector si no existe | O(1) |
| Actualizar el mapa del subsector si existe | O(1) |
| Obtener el subsector de interés | O(k) |
| Hacer un sort de las actividades del subsector | O(nlog(n)) |
| Organizar las 3 primeras y las 3 últimas actividades | O(1) |
| ***TOTAL*** | ***O(m+nlog(n))*** |

## Pruebas Realizadas

**Procesadores**

| | |
|---|---|
| **Memoria RAM** | 8GB |
| **Sistema Operativo** | Microsoft Windows 10 Home Single Language |

Comentado [JL1]: Completar

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).

## Tablas de datos

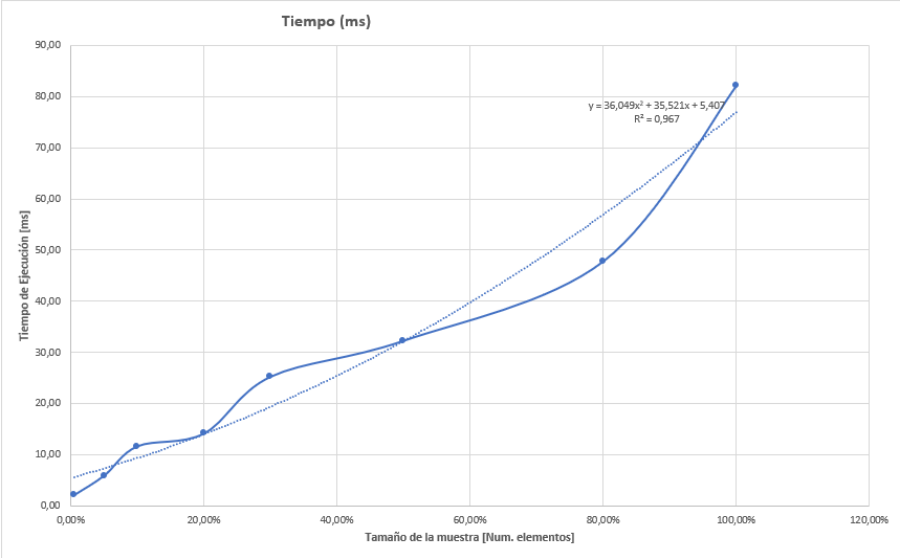
Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** | **Memoria utilizada (kB)** |
|---|---|---|---|
| Small,2021 | Subsector: 3<br>Actividades: 3320, 1392, 2750 | 2,12 | 3,625 |
| 5 pct, 2021 | Subsector: 7<br>Actividades: 4782, 4512, 4541, 4520, 4663 | 5,94 | 14,57 |
| 10 pct, 2021 | Subsector: 7<br>Actividades: 4782, 4512, 4541, 4645, 4690 | 11,67 | 15,09 |

| | | | |
|---|---|---|---|
| 20 pct, 2021 | Subsector: 7<br>Actividades: 4782, 4512, 4643, 4645, 4690 | 14,17 | 19,6 |
| 30 pct, 2021 | Subsector: 3<br>Actividades: 2680, 3220, 2818, 2023, 2229 | 25,27 | 12,88 |
| 50 pct, 2021 | Subsector: 3<br>Actividades: 2680, 2826, 3220, 3312, 2023, | 32,31 | 16,58 |
| 80 pct, 2021 | Subsector: 3<br>Actividades: 1922, 2680, 2826, 1410, 2100 | 47,83 | 20,35 |
| large, 2021 | Subsector: 3<br>Actividades: 1922, 2680, 2826, 1410, 2100 | 82,18 | 17,96 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

En este caso se puede observar una gráfica que presenta momentos de linealidad dependiendo del tamaño de los datos que se estén cargando en un año especificado, las actividades que se cargan en el año dado hacen que el crecimiento de la gráfica se vea afectado, siendo esto nlog(n) en el peor de los casos debido a que también se hace un ordenamiento con MergeSort.

# Requerimiento 5

## Descripción

Este requerimiento recibe como parámetro la estructura de datos del modelo y el año de interés, en este caso se crea un mapa en el que las llaves son subsectores, y su valor es la información de interés del subsector (la suma de los aportes de cada actividad) sumado a una lista de actividades de cada subsector. Se verifica si el subsector ya existe en el mapa, en caso de que no se inicializa, en caso de que si se actualiza. Posteriormente se obtiene el subsector con el mayor descuento y se filtran las actividades de dicho subsector.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, año |
| **Salidas** | Mapa del subsector con mayores descuentos, lista de actividades del subsector filtrada |
| **Implementado (Sí/No)** | Si, implementado por Juan Ardila |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| Pasos | Complejidad |
|---|---|
| Obtener la lista de las actividades de dicho año | O(1) |
| Inicializar el mapa de subsectores | O(1) |
| Recorrer las actividades del año | O(m) |
| Inicializar el mapa del subsector si no existe | O(1) |
| Actualizar el mapa del subsector si existe | O(1) |
| Obtener el subsector de interés | O(1) |
| Se hace un sort de las actividades del subsector | O(plog(p)) |
| ***TOTAL*** | ***O(m+plog(p))*** |

En este caso m es el numero de actividades del año de interés, y p es el número de actividades del subsector que tuvo más descuentos

## Pruebas Realizadas

| | |
|---|---|
| **Procesador** | Intel(R) Core (TM) i7-10870H CPU @ 2.20GHz 2.21 GHz |
| **Memoria RAM** | 32,0 GB |
| **Sistema Operativo** | Windows 11 Home-64 bits |

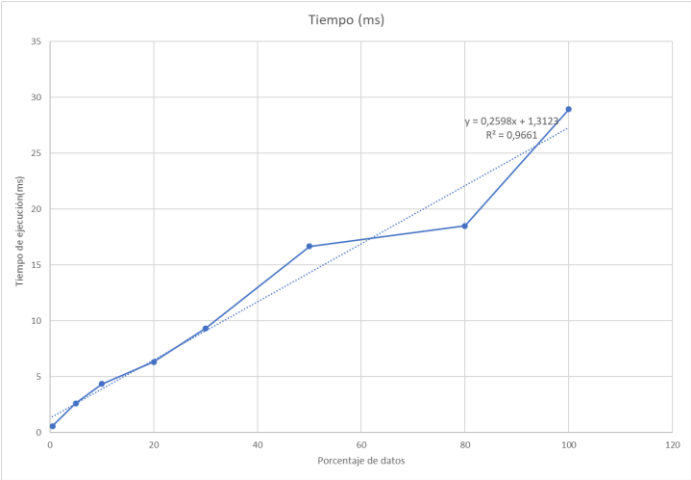
### Tablas de datos

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron año 2021.

| Muestra | Salida | Tiempo (ms) | Memoria utilizada (kB) |
|---|---|---|---|
| Small,2021 | Subsector:3<br>Actividades: 3320, 1392, 2750 | 0,58 | 7,69 |
| 5 pct, 2021 | Subsector:7<br>Actividades: 4782,4512,4520,4541,4663 | 2,61 | 17,45 |
| 10 pct, 2021 | Subsector:7<br>Actividades: 4782,4512,4520,4719,4645,4690 | 4,35 | 19,00 |
| 20 pct, 2021 | Subsector:7<br>Actividades: 4782,4512,4643,4719,4645,4690 | 6,32 | 20,30 |
| 30 pct, 2021 | Subsector:3<br>Actividades: 3099,3220,2680,1090,2229,1104 | 9,33 | 19,51 |
| 50 pct, 2021 | Subsector:3<br>Actividades: 3099,3220,2680,1090,2229,1104 | 16,66 | 21,71 |
| 80 pct, 2021 | Subsector:11<br>Actividades: 6432,6513,6532,6511,6613,6411 | 18,49 | 18,83 |
| large, 2021 | Subsector:3<br>Actividades: 1922,2520,3099,1104,2394,1103 | 28,93 | 18,66 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

En este caso se puede observar que el algoritmo presenta principalmente una complejidad lineal que varía en algunos casos, esto se debe principalmente debido a que nuestra complejidad está definida en términos del número de elementos del año de interés y el número de actividades del subsector, y al estar parametrizado en dos variables, se puede aumentar la tasa de crecimiento o disminuir en algunos intervalos. Además, se puede presentar una variación debido a que en la complejidad se tiene un factor log(n)

# Requerimiento 6

## Descripción

```
def print_req_6(control,anio,memflag):
    """
        Función que imprime la solución del Requerimiento 6 en consola
    """
    #Obtener información relacionada al sector con mayor ingreso neto
    tuple_info=controller.req_6(control,anio,memflag)
    sector=tuple_info[0]
    subsector_most=tuple_info[1]
    subsector_less=tuple_info[2]
    actividad_mas_aporto_1=tuple_info[3]
    actividad_menos_aporto_1=tuple_info[4]
    actividad_mas_aporto_2=tuple_info[5]
    actividad_menos_aporto_2=tuple_info[6]
    time=tuple_info[7]
    if len(tuple_info)==9:
        memory=tuple_info[8]
    #Alistar los datos necesarios para tabular de la tabla 1
    headers_table_1=["Código sector económico",
            "Nombre sector económico",
            "Total ingresos netos del sector económico",
            "Total costos y gastos del sector económico",
            "Total saldo a pagar del sector económico",
            "Total saldo a favor del sector económico",
            "Subsector económico que más aporto",
            "Subsector económico que menos aporto"]
    table_1=[]
    for header in headers_table_1:
        table_1.append(lt.getValue(sector.get(sector,header)))
    #Se ajusta el formato de la tabla puesto a que solo es un valor, es decir hay una relacion 1-1 header-valor
    list2tabulate_1=[headers_table_1,table_1]
    #Imprimir el resultado con respecto al requerimiento
    print("============ Req No. 5 Inputs ============")
    print(f"Find the economic activity with the highest total net income for each economic sector in '{anio}'")
    print("============ Req No. 1 Answer ============")
    print(tabulate(list2tabulate_1,tablefmt="grid",maxcolwidths=20))
    #Alistar los datos necesarios para tabular de la tabla 2 y 3
    headers_table_2_and_3=["Código subsector económico",
            "Nombre 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",
            "Actividad económica que más aporto",
            "Actividad económica que menos aporto"]
    table_2=[]
    #Se recorren los headers de la tabla 2 y tres omitiendo
    #las actividades económicas
    for i in range(0,6):
        header=headers_table_2_and_3[i]
        table_2.append(lt.getValue(sector.get(subsector_most,header)))
    #Se crean las tablas internas
    table_mas_aporto_1=small_table(actividad_mas_aporto_1)
    table_menos_aporto_1=small_table(actividad_menos_aporto_1)
    #Se añaden las tablas internas
    table_2.append(table_mas_aporto_1)
    table_2.append(table_menos_aporto_1)
    #Se ajusta el formato de la tabla puesto a que solo es un valor, es decir hay una relacion 1-1 header-valor
    list2tabulate_2=[headers_table_2_and_3,table_2]
    #Imprimir el resultado con respecto al requerimiento de subsector mas contributivo
    print("============ Economic subsector that contributed the most ============")
    print(tabulate(list2tabulate_2,tablefmt="grid"))
    #Se alistan los datos
    table_3=[]
    #Se recorren los headers de la tabla 2 y 3 omitiendo
    #las actividades económicas
    for i in range(0,6):
        header=headers_table_2_and_3[i]
        table_3.append(lt.getValue(sector.get(subsector_less,header)))
    #Se crean las tablas internas
    table_mas_aporto_2=small_table(actividad_mas_aporto_2)
    table_menos_aporto_2=small_table(actividad_menos_aporto_2)
    #Se añaden las tablas internas
    table_3.append(table_mas_aporto_2)
    table_3.append(table_menos_aporto_2)
    #Se ajusta el formato de la tabla puesto a que solo es un valor, es decir hay una relacion 1-1 header-valor
    list2tabulate_3=[headers_table_2_and_3,table_3]
    #Imprimir el resultado con respecto al requerimiento de subsector mas contributivo
    print("============ Economic subsector that contributed the less ============")
    print(tabulate(list2tabulate_3,tablefmt="grid"))
    print("Tiempo de ejecución total[ms]: ",time)
    if len(tuple_info)==9:
        memory=tuple_info[8]
        print("Total memoria usada[kB]: ",memory)
```

Este requerimiento se encarga de retornar el sector económico que tuvo mayores ingresos netos para un año específico, además retorna los subsectores que más y menos aportaron, y de cada subsector retorna la actividad que más y menos aportó.

| **Entrada** | Estructuras de datos del modelo, año |
|---|---|
| **Salidas** | Sector con más ingresos(mapa), subsector con más ingresos del sector(map),subsector con menos ingresos del sector(map),actividad que más ingresos tuvo del sector que más |

| | |
|---|---|
| | ingresos tuvo, actividad que menos ingresos tuvo del sector que más aporto, actividad que más ingresos tuvo del sector que menos ingresos tuvo, actividad que menos ingresos tuvo del sector que menos ingresos tuvo |
| **Implementado (Sí/No)** | Si |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener la lista de las actividades del año | O(1) |
| Crear el mapa de sectores | O(1) |
| Recorrer las actividades de ese año | O(m) |
| Crear el mapa del sector y el subsector | O(1) |
| Obtener los valores de interés | O(1) |
| Filtrar la lista de actividades del subsector que más aportó | O(plog(p)) |
| Filtrar la lista de actividades del subsector que más aporto | O(klog(k)) |
| Obtener las actividades de interés | O(1) |
| ***TOTAL*** | ***O(m+plog(p) +klog(k))*** |

En este caso m corresponde a la cantidad de actividades de dicho año, p corresponde a las actividades del subsector que más aporto y k corresponde a las actividades del subsector que menos aportó.

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron año 2021.

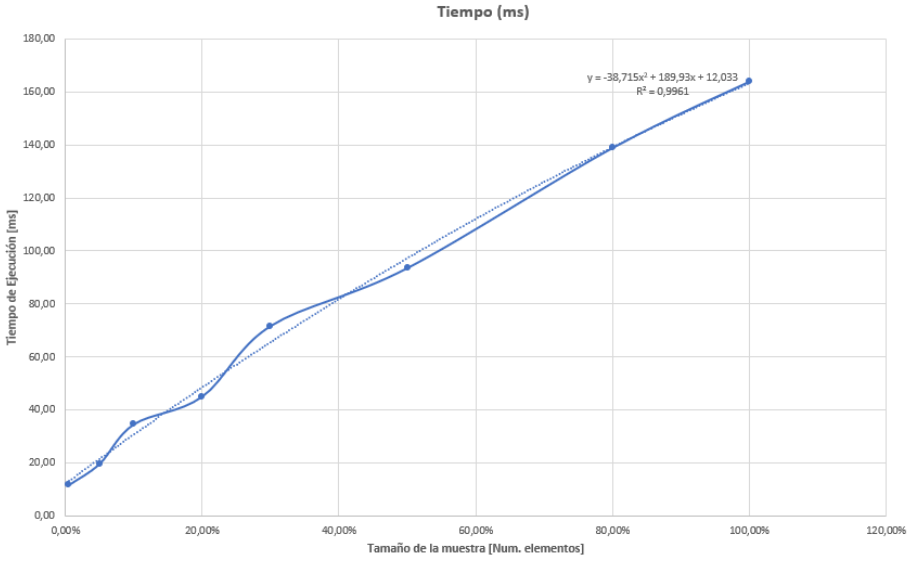
| **Procesadores** | |
|---|---|
| **Memoria RAM** | 8GB |
| **Sistema Operativo** | Microsoft Windows 10 Home Single Language |

### Tablas de datos.

| **Muestra** | **Salida** | **Tiempo (ms)** | **Memoria utilizada (kB)** |
|---|---|---|---|
| Small,2021 | Código Sector: 8 Actividades: 6432, 6629, | 11,58 | 24,31 |
| 5 pct,2021 | Código Sector: 6 Actividades: 4663, 4782, 5612 | 19,65 | 42,29 |
| 10 pct,2021 | Código Sector: 6 Actividades: 4645, | 34,54 | 45,85 |

| | | | |
|---|---|---|---|
| | 4782, 5612 | | |
| 20 pct,2021 | Código Sector: 6 Actividades: 4625, 4782, 5612, 5513 | 45,00 | 36,47 |
| 30 pct,2021 | Código Sector: 6 Actividades: 4645, 4782, 5611, 5513 | 71,68 | 46,18 |
| 50 pct,2021 | Código Sector: 8 Actividades: 6532, 6513,6820, 6820 | 93,54 | 41,79 |
| 80 pct,2021 | Código Sector: 8 Actividades: 6532, 6513,6820, 6820 | 138,88 | 41,40 |
| Large, 2021 | Código Sector: 8 Actividades: 6532, 6513,6810, 6820 | 163,95 | 41,84 |

## Graficas




## Análisis

Se observa un crecimiento casi lineal, para este caso es un nlog(n) debido a que algunos de los datos para ciertos intervalos específicos, el orden de crecimiento afecta un poco más al momento de realizar la búsqueda, aunque en la mayoría de los casos se logra observar un comportamiento lineal.

# Requerimiento 7

## Descripción

```
def req_7(data_structs,anio,subsector,n):
    """
    Función que soluciona el requerimiento 7
    """
    #Se obtiene la lista de las actividades de ese año
    list_activities_year=me.getValue(mp.get(data_structs,anio))
    #Se obtiene la lista de las actividades del subsector que estan en el año
    list_activities_year_and_subsector=lt.newList(datastructure="ARRAY_LIST")
    for actividad in lt.iterator(list_activities_year):
        if actividad["info"]["Código subsector económico"]==subsector:
            lt.addLast(list_activities_year_and_subsector,actividad)
    #Se filtra la lista segn el criterio de menor total de costos y gastos
    merg.sort(list_activities_year_and_subsector,cmp_costos_gastos)
    #Se toman los n primeros valores si la lista excede el n
    if int(n)>=lt.size(list_activities_year_and_subsector):
        return list_activities_year_and_subsector
    else:
        list2return=lt.newList(datastructure="ARRAY_LIST")
        for i in range(1,int(n)+1):
            lt.addLast(list2return,lt.getElement(list_activities_year_and_subsector,i))
        return list2return
```

Este requerimiento se encarga de retornar el top N de actividades económicas con el menor total de costos y gastos para un subsector y año específico

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, año, subsector, N |
| **Salidas** | Lista(ARRAY_LIST) de las N actividades que tuvieron los menores descuentos y gastos. En caso de que se tenga un número de actividades menor que N, se retornan todas las actividades del subsector de interés |
| **Implementado (Sí/No)** | Si |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener la lista de las actividades del año | O(1) |
| Obtener la lista de actividades del subsector que están en el año | O(m) |
| Ordenar las actividades del subsector | O(mlog(m))) |
| Retornar la lista de actividades de interés | O(1) |
| ***TOTAL*** | ***O(m+mlog(m))*** |

En este caso m corresponde a la cantidad de actividades de dicho año.

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron año 2020 y el sector 11.

Comentado [JSAL2]: Completar

| **Procesadores** |
|---|

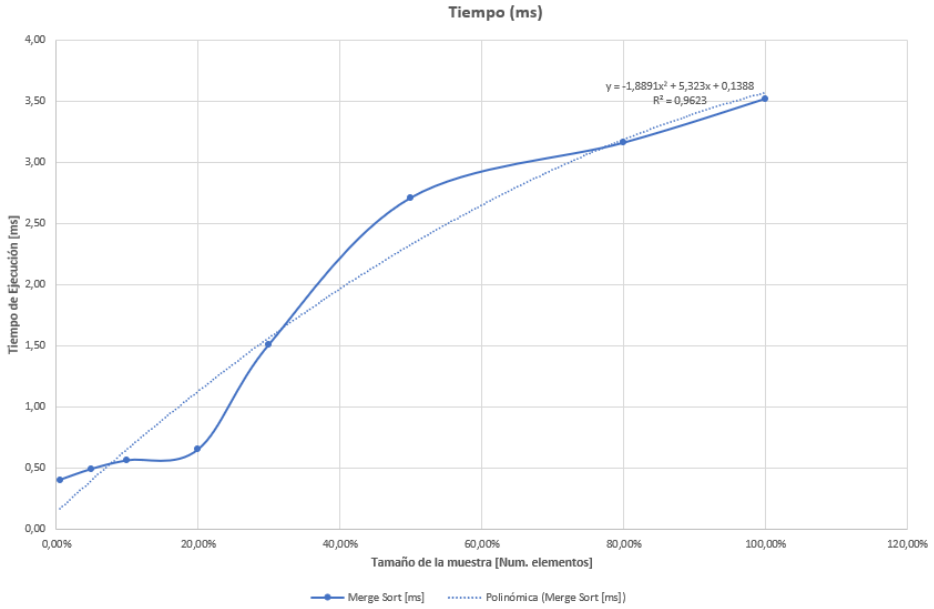
| | |
|---|---|
| **Memoria RAM** | 8GB |
| **Sistema Operativo** | Microsoft Windows 10 Home Single Language |

## Tablas de datos.

| Muestra | Salida | Tiempo (ms) | Memoria utilizada (kB) |
|---|---|---|---|
| Small, 2020 | SubSector: 11<br>Actividades: 6422 | 0,40 | 0,5078 |
| 5 pct, 2020 | SubSector: 11<br>Actividades: 6422 | 0,49 | 0,5078 |
| 10 pct, 2020 | SubSector: 11<br>Actividades: 6422, 6532 | 0,56 | 0,5078 |
| 20 pct, 2020 | SubSector: 11<br>Actividades: 6422, 6423, 6521, 6532 | 0,65 | 0,5078 |
| 30 pct, 2020 | SubSector: 11<br>Actividades: 6522, 6424, 6613, 6422, 6423, 6521, 6532 | 1,51 | 0,5078 |
| 50 pct, 2020 | SubSector: 11<br>Actividades: 6522, 6615, 6629, 6424, 6421, 6621, 6613, 6422, 6492 | 2,71 | 0,5078 |
| 80 pct, 2020 | SubSector: 11<br>Actividades: 6513 6531, 6522, 6615, 6629, 6493, 6424, 6421, 6494 | 3,16 | 0,5078 |
| large, 2020 | SubSector: 11<br>Actividades: 6614, 6513, 6531, 6522, 6491, 6615, 6496, 6629, 6611 | 3,52 | 0,5078 |

Comentado [JSAL3]: Completar

## Graficas




## Análisis

Al ser este algoritmo organizado a partir de un MergeSort su complejidad no va a ser completamente lineal, sino su carga en el peor de los casos ser nlog(n). Como se puede apreciar en la gráfica, algunos datos no presentan una consistencia lineal con respecto a los demás, por esta razón no se logra observa un comportamiento totalmente lineal.

# Requerimiento 8

## Descripción

Este requerimiento permite visualizar los subsectores ordenados de mayor a menor con respecto a su total de impuestos a cargo y adicionalmente se puede observar el top N de actividades con mayores impuestos a cargo de cada subsector.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, año, N |
| **Salidas** | Lista (ARRAY_LIST) de los subsectores organizados de mayor a menor con respecto a los impuestos a cargo y un mapa con el top N de actividades que le corresponde a cada uno de los subsectores |
| **Implementado (Sí/No)** | Si |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| Pasos | Complejidad |
|---|---|
| Obtener la lista del año ingresado | O(1) |
| Creación de estructuras de datos | O(1) |
| Recorrer todas las actividades de un año | O(n) |
| Inicializar el mapa del subsector si no existe | O(1) |
| Actualizar el mapa del subsector si existe | O(1) |
| Añadir subsectores a una lista | O(m) |
| Ordenar los subsectores con respecto a sus impuestos a cargo | O(mlog(m)) |
| Verificar si hay más de 12 subsectores | O(1) |
| Agregar los subsectores que se imprimirán | O(4) |
| Recorrer la lista de subsectores | O(k) |
| Obtener actividades de cada subsector | O(1) |
| Organizar actividades | O(plog(p)) |
| Escoger actividades del top | O(1) |
| Agregar las actividades al mapa | O(1) |
| Retornar mapa y lista | O(1) |
| ***TOTAL*** | *O(*mlog(m))+ plog(p*)* |

## Pruebas Realizadas

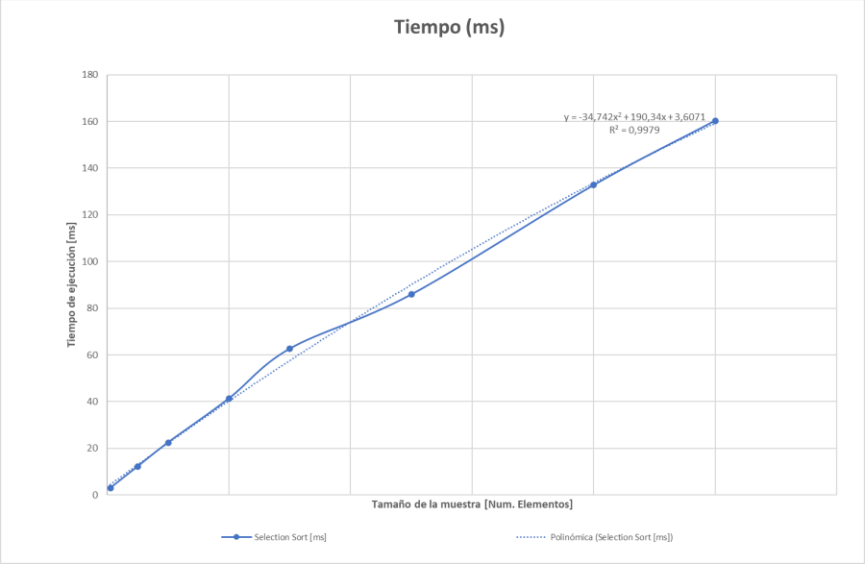
| **Procesadores** | **11th Gen Intel(R) Core (TM) i7-1165G7 @ 2.80GHz 2.80 GHz** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 11 Home-64 bits |

## Tablas de datos

| Muestra | Salida | Tiempo (ms) | Memoria utilizada (kB) |
|---|---|---|---|
| Small,2021,4 | Subsector 3: 2750,1392,3320<br>Subsector 7: 4512<br>Subsector 11: 6629,6432<br>Subsector 20: 9700 | 3.02 | 12,95 |
| 5 pct, 2021,10 | Subsector 7: 4663,4520,4541<br>Subsector 3: 1811, 2750,1392<br>Subsector 1: 9529<br>Subsector 8: 4922,5121,5011<br>Subsector 9: 5612<br>Subsector 11: 6629,6432<br>Subsector 14: 8292,7721<br>Subsector 18: 9319<br>Subsector 1: 161,123,127<br>Subsector 20: 9700 | 12,27 | 47,32 |

| | | | |
|---|---|---|---|
| 10 pct, 2021,6 | Subsector 10:5812<br>Subsector 20: 9700<br>Subsector 2: 2478<br>Subsector 3: 1090, 2011,1811<br>Subsector 15: 8424<br>Subsector 7: 4645,4663,4690 | 22,49 | 38,67 |
| 20 pct, 2021,6 | Subsector 7:4645,4663,4690<br>Subsector 2:610,610,510<br>Subsector 3: 1103,1130,2229<br>Subsector 20: 9700,9700,9820<br>Subsector 21: 9900<br>Subsector 15: 8412,8414,8424 | 41,34 | 42,3 |
| 30 pct, 2021,6 | Subsector 7:4645,4663,4690<br>Subsector 2:610,610,510<br>Subsector 3: 1103,1130,2100<br>Subsector 20: 9700,9700,9820<br>Subsector 21: 9900,9900<br>Subsector 15: 8412,8414,8413 | 62.63 | 43,67 |
| 50 pct, 2021,6 | Subsector 7:4645,4663,4661<br>Subsector 3: 2013,2229,2029<br>Subsector 20: 9700<br>Subsector 21: 9900,9900<br>Subsector 15: 8413,8424,8415<br>Subsector 8: 4930,4923,4921 | 85.88 | 43,83 |
| 80 pct, 2021,6 | Subsector 7:4645,4663<br>Subsector 3: 2013,1410,1103<br>Subsector 20: 9700,9820<br>Subsector 21: 9900<br>Subsector 2: 610,510,722<br>Subsector 19: 9603,9499,9512 | 132,72 | 42,47 |
| large, 2021,6 | Subsector 7:4645,4663,4661<br>Subsector 3: 2013,1410,1103<br>Subsector 20: 9700,9820,9810<br>Subsector 21: 9900<br>Subsector 2: 610,510,722<br>Subsector 19: 9603,9499,9512 | 160,36 | 42,85 |

## Graficas




## Análisis

Se puede observar que el logaritmo tiene un crecimiento lineal y es acorde al orden de crecimiento planteado con anterioridad.