Universidad de los Andes
Facultad de Ingeniería

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

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO 2

Eric Sebastian Alarcon Dolynko, e.alarcond@uniandes.edu.co

Brainer Steven Jimenez Gonzalez, 202222320, b.jimenezg@uniandes.edu.co

Carga de datos

```
def new_data_structs():
    """
    Inicializa las estructuras de datos del modelo. Las crea de
    manera vacía para posteriormente almacenar la información.
    """

    catalog = {"data":None}
    catalog["data"] = new_hash(10)

    return catalog
```

```
def new_hash(size):

    hash = mp.newMap(size,
                     maptype='PROBING',
                     loadfactor = 0.5,
                     cmpfunction = compareTridente)

    return hash
```

Primero creamos nuestro data structs, que es un hashmap para 10 elementos, tiene factor de carga de 0.5 y es de tipo linear probing (todos los hashes en el reto fueron hechos con el mismo factor de carga y tipo de hash).

```
def mega_load(control,memory,tamanio):
    """
    Carga los datos del reto
    """
    time_memory = start_gathering_data(memory)

    data_structs = control['model']
    dianfiles = cf.data_dir + f'DIAN/Salida_agregados_renta_juridicos_AG-{tamanio}.csv'
    input_file = csv.DictReader(open(dianfiles, encoding='utf-8'))

    for data in input_file:
        model.add_todo(data_structs, data['Año'], data)

    size = model.sort_all_info(control["model"])
    answer = (data_structs,size)
    return stop_gathering_data(answer,memory,time_memory[0],time_memory[1])
```

Esta función en el controlador, lee el archivo csv y por cada dato invoca la función add_todo del modelo.

```
def add_todo(catalog,year,data):

    year = ajustar_llave(year)
    mega_hash = catalog["data"]
    exist_map = mp.contains(mega_hash,year)

    if exist_map == False:
        mp.put(mega_hash,year,new_hash(5))

    add_year(mega_hash,year,data)

    pass
```

Revisa si el año del dato está en en datastrucs, si no lo está lo inserta como una llave y su valor es un hashmap para 5 elementos. Luego invoca la función add_year por cada dato en el csv.

```python
def add_year(mega_hash, year, data):

    hash_year = get_value(mega_hash,year)
    existyear = mp.contains(hash_year,'info')

    if existyear == False:
        mp.put(hash_year,'info',new_info())
        mp.put(hash_year,'sector',new_hash(30))
        mp.put(hash_year,"iterator sector",new_iterator())
        mp.put(hash_year,'subsector',new_hash(30))
        mp.put(hash_year,"iterator subsector",new_iterator())


    add_info(hash_year, data)
    add_sector(hash_year, data)
    add_iterator_sector(hash_year, ajustar_llave(data['Código sector económico']))
    add_subsector(hash_year, data)
    add_iterator_subsector(hash_year, ajustar_llave(data['Código subsector económico']))
    pass
```

add_year agrega la información que va dentro del hash de cada año. Primero revisa si la llave info está dentro del hash del año. Si no está, crea los 5 elementos que debe tener cada hash de los años. Info es un array list, sector es un hashmap con 30 elementos. Iterator sector, que es una lista sencillamente encadenada. Subsector es un hashmap con 30 elementos. Iterator subsector, que es una lista sencillamente encadenada. Después de eso, se usan las funciones add. Add info es para agregar el dato entrante a la la lista en la llave info del hash del año.

```
def add_sector(hash_year, data):

    hash_sector = get_value(hash_year,'sector')

    sector_unico = ajustar_llave(data['Código sector económico'])
    existsector = mp.contains(hash_sector,sector_unico)

    if existsector == False:
        mp.put(hash_sector,sector_unico,new_hash(4))
        hash_num_sector = get_value(hash_sector, sector_unico)
        mp.put(hash_num_sector,'info',new_info())
        mp.put(hash_num_sector,'subsector',new_hash(30))
        mp.put(hash_num_sector,"iterator subsector",new_iterator())
        mp.put(hash_num_sector,'combined',new_info())
    else:
        hash_num_sector = get_value(hash_sector, sector_unico)

    add_info(hash_num_sector, data)
    add_subsector(hash_num_sector, data)
    add_iterator_subsector(hash_num_sector, ajustar_llave(data['Código subsector económico']))
    add_combined(hash_num_sector, data)

    pass
```

Add sector revisa si ya existe el sector de un dato en el hash de la llave sector, del hash del año. Si no existe en el hash de la llave sector, pone una pareja llave, valor. La llave es el numero del sector, y el valor es unn hash para 4 elementos. Luego, crea la llave info con valor de una lista. Subsector, con valor de un hash para 30 elementos. Iterator subsector, una lista. Y combined, otra lista. Finalmente las funciones de añadir. Add info es para agregar el dato entrante a la la lista en la llave info del hash del año.

```
def add_subsector(hash_num_sector, data):

    hash_subsector = get_value(hash_num_sector,'subsector')

    subsector_unico = ajustar_llave(data['Código subsector económico'])
    existsubsector = mp.contains(hash_subsector,subsector_unico)

    if existsubsector == False:
        mp.put(hash_subsector,subsector_unico,new_hash(2))
        hash_num_subsector = get_value(hash_subsector, subsector_unico)
        mp.put(hash_num_subsector,'info',new_info())
        mp.put(hash_num_subsector,'combined',new_info())
    else:
        hash_num_subsector = get_value(hash_subsector, subsector_unico)

    add_info(hash_num_subsector, data)
    add_combined(hash_num_subsector, data)

    pass
```

Add_subsector revisa si dentro del hash del año, en el hash de la llave sector, en el hash de la llave subsector, existe el subsector del dato. Si no existe agrega una llave valor en el hash de la llave subsector. La llave es el número del subsector y el valor un hash con espacio para 2 elementos. y a este nuevo hash le agrega la llave info, con valor de una lista. Y le añade combined, con valor de una lista. Luego, usa las funciones de añadir. Add_info es para agregar el dato entrante a la la lista en la llave info del hash del año.

```
def add_combined(hash, data):

    lista = get_value(hash,'combined')
    if lt.isEmpty(lista) == True:
        lt.addLast(lista, data)
    else:
        actual = lt.firstElement(lista)
        new_dic = sumar_diccionarios(actual, data)
        lt.changeInfo(lista, 1, new_dic)

    pass
```

Add_combined, revisa si la lista está vacía, si lo está, añade el dato entrante a la lista vacía. Si ya tiene un dato, entonces va a sumar el dato entrante y el dato que ya estaba en la lista cuando la función sumar_diccionarios. Luego cambia el dato actual de la lista, por el dato de la suma previamente hecha.

```python
def sumar_diccionarios(actual, data):

    """Sumar el dato de entrada, al dato que ya existe.

    """

    contenedor = {'Total ingresos netos': None, 'Total costos y gastos': None,'Total saldo a pagar': None,
                  'Total saldo a favor': None, 'Total retenciones': None, 'Costos y gastos nómina': None,
                  'Descuentos tributarios': None, 'Costos ganancias ocasionales' : None, "Total Impuesto a cargo":None}

    new_dic = {}

    for llave in actual:
        if llave in contenedor:
            new_dic[llave] = limpiar_datos(actual[llave]) + limpiar_datos(data[llave])
        else:
            new_dic[llave] = actual[llave]

    return new_dic
```

Recibe como parámetro dos diccionarios. El diccionario del elemento actual y del dato que está entrando. Declaramos cuales son las llaves que queremos sumar, eso está guardado en el contenedor. Creamos un nuevo diccionario vacío. Luego revisa cada llave del dato actual, si la llave está en contenedor, entonces crea una nueva llave en nuestro nuevo diccionario, la llave es la misma que fue revisada, y el valor es la suma del valor de esa llave en los dos diccionarios que entraron como parámetros. Si, la llave no está en el contenedor, entonces crea una nueva llave en el nuevo diccionario, la llave es la misma que fue revisada. Y el valor es el valor de esa misma llave en el diccionario actual. Finalmente, devuelve el nuevo diccionario.

```
def add_iterator_sector(hash, sector):

    lista = get_value(hash,"iterator sector")

    if lt.isPresent(lista,sector) == 0:
        lt.addLast(lista,sector)

    pass

def add_iterator_subsector(hash, subsector):

    lista = get_value(hash,"iterator subsector")

    if lt.isPresent(lista , subsector) == 0:
        lt.addLast(lista , subsector)

    pass
```

Revisa si en la lista de iterator subsector o sector, ya existe en subsector o el sector del dato entrante, si no existe, entonces añade a la lista el número del subsector o del sector. Y así queda una lista con todos los subsectores existentes, y una lista con todos los sectores existentes.

Descripción

Entrada	CSV con los datos de la DIAN entre 1% y 100%
Salidas	Una estructura de datos principalmente un hash, con todos los datos
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1	O(1)
Paso 2	O(n)
Paso 3	O(1)
Paso 4	O(1)
Paso 5	O(1)
Paso 6	O(1)

Paso 7	O(m) (es un for pero no de todos los datos, si no de un número pequeño de columnas)
Paso 8	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

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

Máquina 1:

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
1	1355,3	99,558
5	3554,22	118,269
10	5365,69	412,518
20	8128,65	540,24
30	10544,807	830,564
50	15043,066	1449,36
80	21672,062	2524,916

100	26072,26	3246,877

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
1	1857.527	54.668
5	5164.819	340.007
10	8095.403	805.265
20	12804.552	1806.653
30	17054.816	2589.219
50	25101.369	4602.253
80	37031.141	7726.449
100	44975.216	10611.063

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]	Consumo de datos [Kb]
1	Estructura de datos ordenada	99,558	1355,3
5	Estructura de datos ordenada	118,269	3554,22
10	Estructura de datos ordenada	412,518	5365,69
20	Estructura de datos ordenada	540,24	8128,65
30	Estructura de datos ordenada	830,564	10544,807
50	Estructura de datos ordenada	1449,36	15043,066
80	Estructura de datos ordenada	2524,916	21672,062
100	Estructura de datos ordenada	3246,877	26072,26

Máquina 2

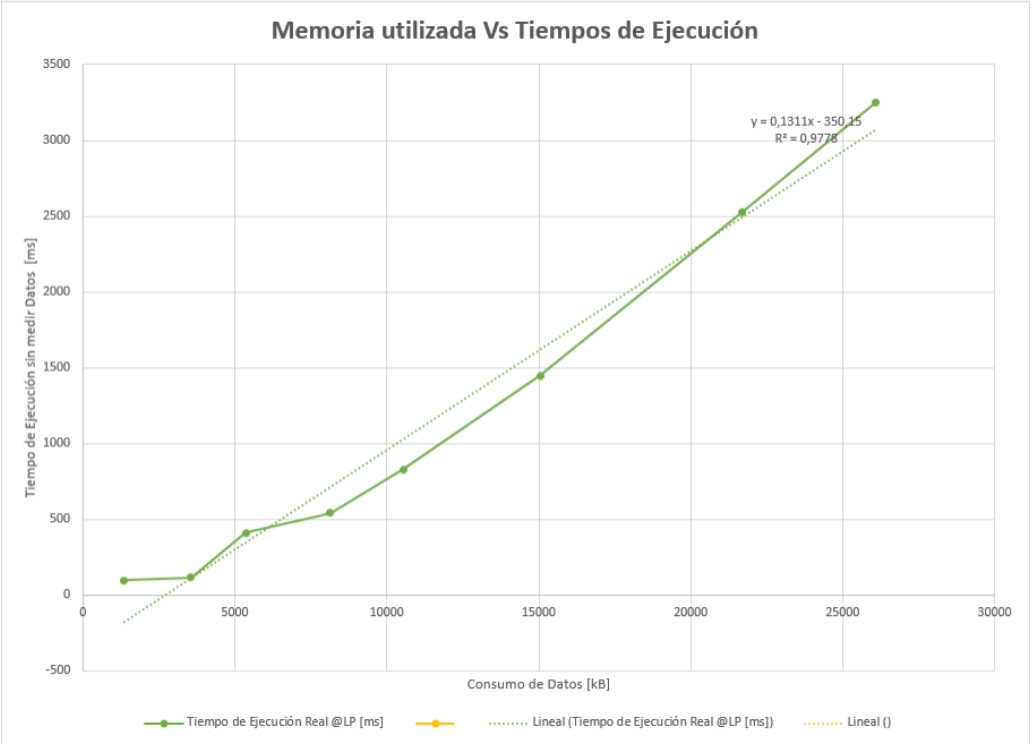
Muestra%	Salida	Tiempo de Ejecución [ms]	Consumo de datos [Kb]
1	Estructura de datos ordenada	54.668	1857.527
5	Estructura de datos ordenada	340.007	5164.819
10	Estructura de datos ordenada	805.265	8095.403
20	Estructura de datos ordenada	1806.653	12804.552
30	Estructura de datos ordenada	2589.219	17054.816

50	Estructura de datos ordenada	4602.253	25101.369
80	Estructura de datos ordenada	7726.449	37031.141
100	Estructura de datos ordenada	10611.063	44975.216

Gráficas

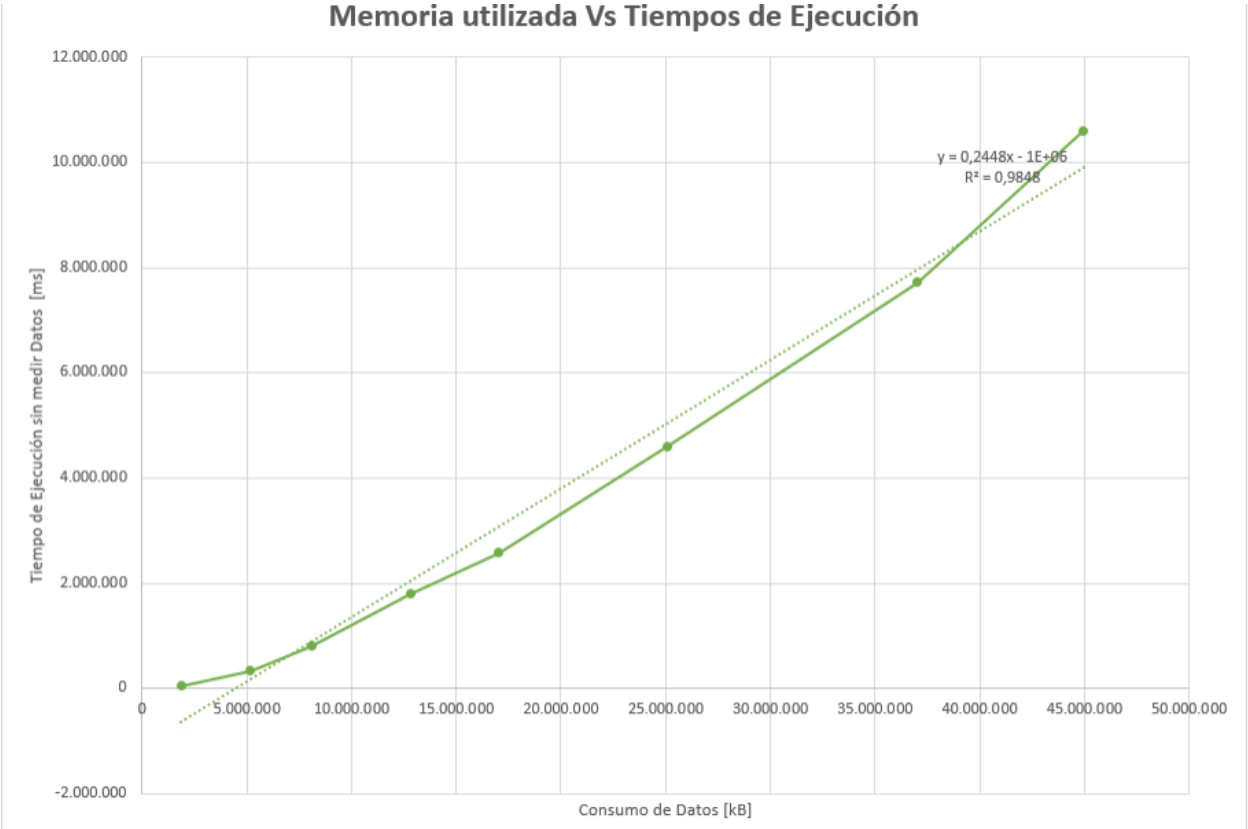
Las gráficas con la representación de las pruebas realizadas.

Máquina 1



Cantidad de archivos cargados vs Memoria utilizada

Máquina 2



Análisis

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

Teóricamente la carga de datos debería ser O(n) en general ya que a pesar de tener otro for en la función de combinar diccionarios, este no recorre toda la lista de los elementos sino que cuando es necesario recorrer unas pocas columnas para sumar dos diccionarios. Es decir que la complejidad que prevalece sigue siendo O(N). En la primera gráfica de Tiempo y Memoria ejecutada se ve una regresión lineal fuerte lo cual muestra que entre mayor cantidad de datos cargados más memoria y más tiempo se demora en ejecutar lo cual es un proceso lógico. Este proceso sucede con un factor de carga óptimo para un hash probing que es con 0.5. Finalmente cabe recalcar que a pesar de ser una carga de datos de baja complejidad por el hecho de ser un mapa no ordenado, este ocupa bastante espacio ya que son varios hash dentro de hash. Este tiempo si fue mayor al del Reto 1 pero porque la estructura fue mucho más compleja de crear sin embargo esto ayudará a que los requerimientos sean más eficientes. La complejidad sin embargo fue la misma.

Requerimiento <<1>>

Descripción

```
def req_1(data_structs,anio,sector):
    """
    Función que soluciona el requerimiento 1
    """
    return mayor_elemento(get_info(get_sector_con_anio(data_structs,anio,sector)),"Total saldo a pagar")
```

Este requerimiento comienza obteniendo el hash que pertenecen a un sector y año en específico, después como segundo paso, get_info obtiene la llave de “info” que es una lista con todos los diccionarios que perteneces a ese hash el cual sería el del año y sector proporcionados. Finalmente mayor_elemento obtiene el diccionario con mayor “Total saldo a pagar” como se muestra a continuación.

```
def mayor_elemento(lista,criterio):

    mayor_valor = limpiar_datos(lt.firstElement(lista)[criterio])
    mayor = lt.firstElement(lista)

    for data in lt.iterator(lista):

        if limpiar_datos(data[criterio]) >= mayor_valor:
            mayor = data
            mayor_valor = limpiar_datos(data[criterio])

    return mayor
```

Mayor elemento obtiene el mayor elemento de una lista bajo un criterio que es una llave específica del diccionario como lo es “Total saldo a pagar” y devuelve el diccionario con mayor total saldo a pagar en este caso.

Entrada	Se solicita un año entre el 2012 y el 2021, y el número de un sector
Salidas	La actividad económica con mayor saldo a pagar en el año y sector solicitado

Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(n), n siendo el número de elementos que tienen el año y sector solicitado
TOTAL	***O(n)***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0,189
5	0,259
10	0,312
20	0,355

30	0,411
50	0,546
80	0,832
100	1,080

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0.209
5	0.255
10	0.258
20	0.355
30	0.466
50	0.704
80	0.764
100	0.964

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]
1	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0,189
5	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0,259
10	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0,312
20	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0,355
30	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0,411
50	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0,546
80	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	2524,916
100	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	3246,877

Máquina 2

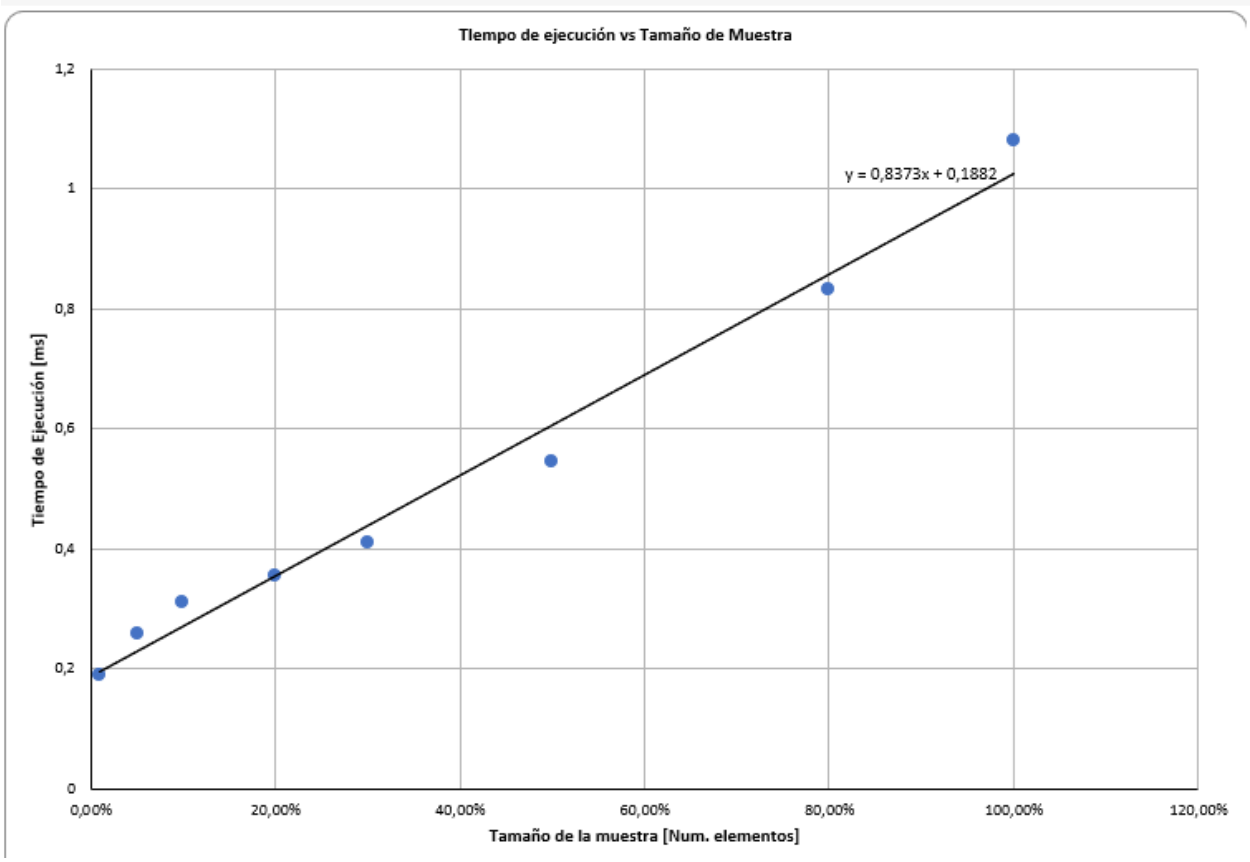
Muestra%	Salida	Tiempo de Ejecución [ms]
1	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.209

5	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.255
10	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.258
20	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.355
30	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.466
50	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.704
80	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.764
100	actividad económica con mayor saldo a pagar para el sector 1 en el año 2016	0.964

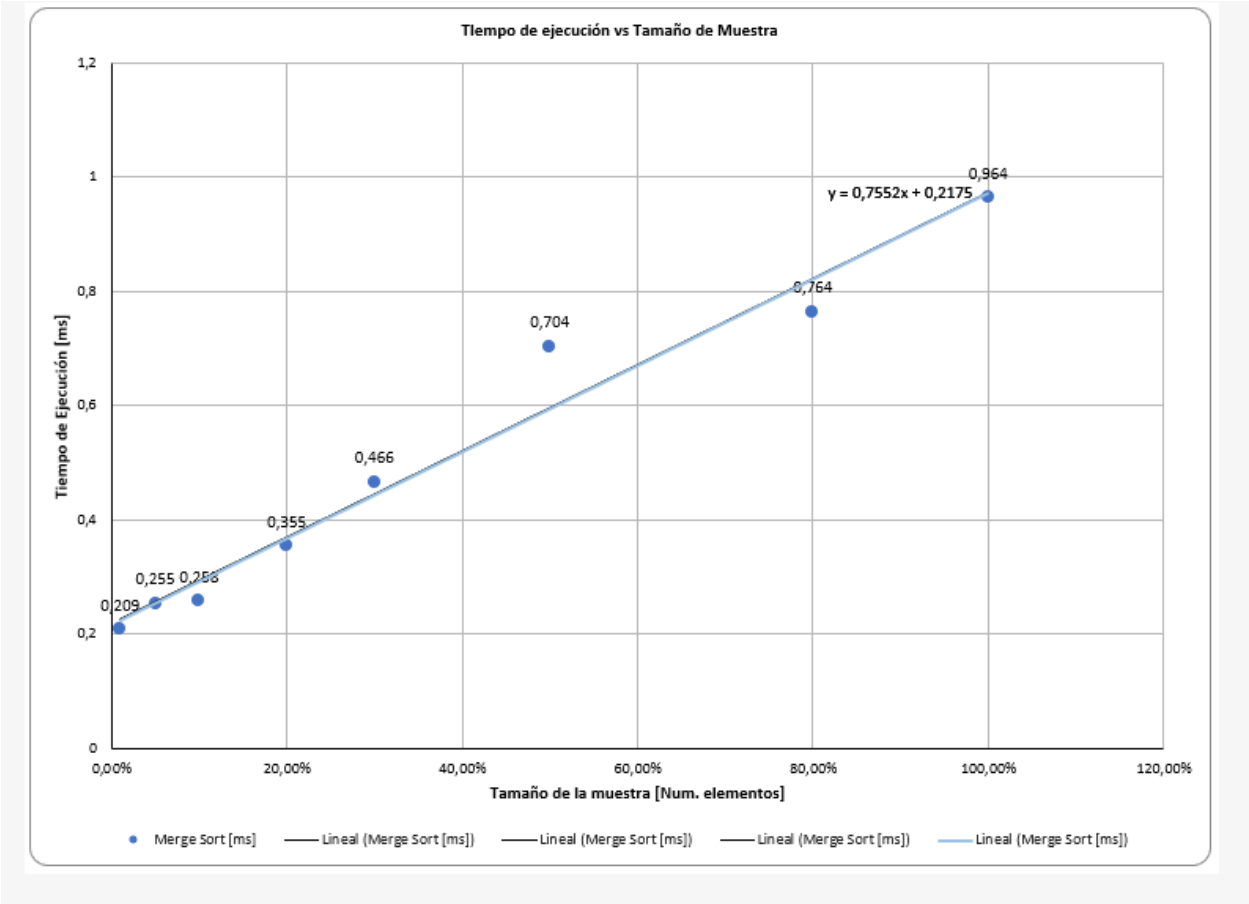
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1


Tlempo de ejecución vs Tamaño de Muestra
y = 0,8373x + 0,1882
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
1,2
1
0,8
0,6
0,4
0,2
0
0,00%
20,00%
40,00%
60,00%
80,00%
100,00%
120,00%

Máquina 2



Análisis

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

Teóricamente el requerimiento tiene complejidad O(N) aunque cabe mencionar que en este caso n no es la cantidad total de datos cargados si no los datos pertenecientes a la llave obtenida en el requerimiento por lo que el tiempo de ejecución es menor en comparación a si se buscara en todos los datos cargados. Así mismo se puede ver un regresión lineal fuerte en las gráficas que validan esta complejidad. La complejidad fue menor al Reto No 1 ya que se pasó de O(N^2) a O (N) y consecuentemente los tiempos de ejecución fueron mucho menor.

Requerimiento <<2>>

Descripción

```
"""
def req_2(data_structs,anio,sector):
    """
    Función que soluciona el requerimiento 2
    """
    return mayor_elemento(get_info(get_sector_con_anio(data_structs,anio,sector)),"Total saldo a favor")
```

Este requerimiento comienza obteniendo el hash que pertenecen a un sector y año en específico, después como segundo paso, get_info obtiene la llave de “info” que es una lista con todos los diccionarios que perteneces a ese hash el cual sería el del año y sector proporcionados. Finalmente mayor_elemento obtiene el diccionario con mayor “Total saldo a favor” como se muestra a continuación.

```
def mayor_elemento(lista,criterio):

    mayor_valor = limpiar_datos(lt.firstElement(lista)[criterio])
    mayor = lt.firstElement(lista)

    for data in lt.iterator(lista):

        if limpiar_datos(data[criterio]) >= mayor_valor:
            mayor = data
            mayor_valor = limpiar_datos(data[criterio])

    return mayor
```

Mayor elemento obtiene el mayor elemento de una lista bajo un criterio que es una llave específica del diccionario como lo es “Total saldo a favor” y devuelve el diccionario con mayor total saldo a favor en este caso.

Entrada	Se solicita un año entre el 2012 y el 2021, y el número de un sector
Salidas	La actividad económica con mayor saldo a favor en el año y sector solicitado
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(n), m siendo el número de elementos que tienen el año y sector solicitado
TOTAL	***O(n)***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Tiempo de Ejecución Real @LP [ms]
1	0,217
5	0,330
10	0,513
20	0,821
30	0,848
50	1,276

80	1,744
100	2,304

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Tiempo de Ejecución Real @LP [ms]
1	0,238
5	0,387
10	0,570
20	1,121
30	0,848
50	1,439
80	2,692
100	3,340

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]

1	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0,217
5	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0,330
10	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0,513
20	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0,821
30	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0,848
50	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	1,276
80	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	1,744
100	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	2,304

Máquina 2

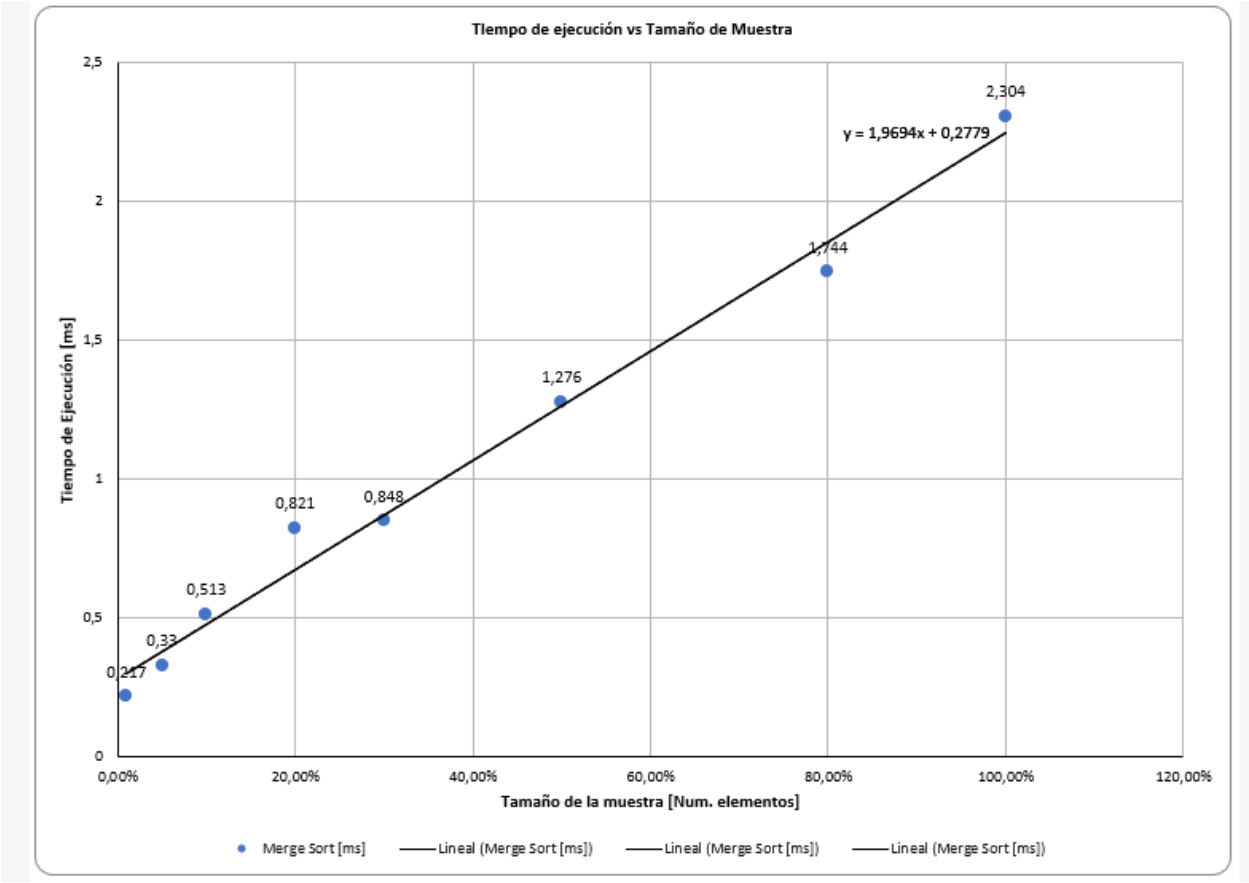
Muestra%	Salida	Tiempo de Ejecución [ms]
1	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0.238
5	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0.387
10	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0.570
20	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	1.121

30	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	0,848
50	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	1.439
80	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	2.692
100	actividad económica con mayor saldo a favor para el sector 3 en el año 2021	3.340

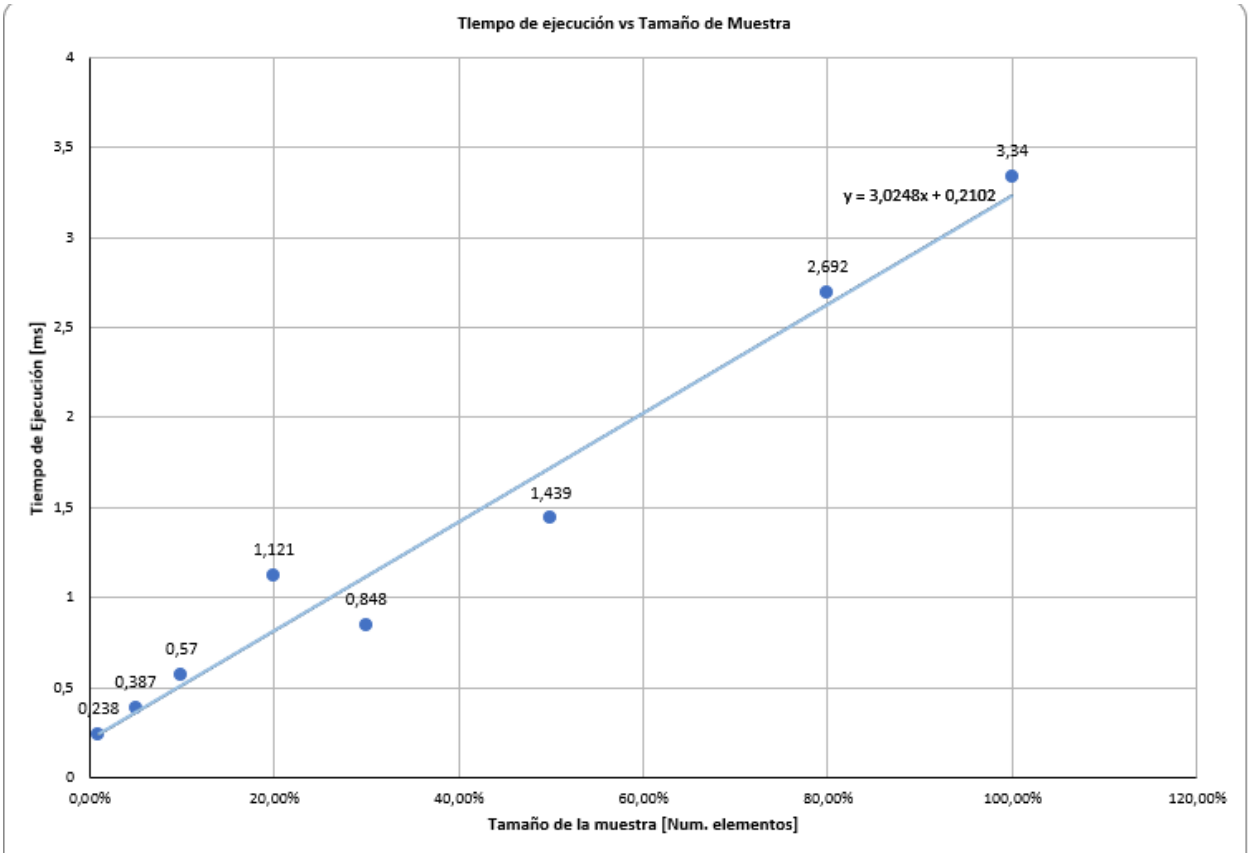
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1



Máquina 2



Análisis

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

Teóricamente el requerimiento tiene complejidad O(N) aunque cabe mencionar que en este caso n no es la cantidad total de datos cargados si no los datos pertenecientes a la llave obtenida en el requerimiento por lo que el tiempo de ejecución es menor en comparación a si se buscara en todos los datos cargados. Así mismo se puede ver un regresión lineal fuerte en las gráficas que validan esta complejidad. La complejidad fue menor al Reto No 1 ya que se pasó de O(N^2) a O (N) y consecuentemente los tiempos de ejecución fueron mucho menor.

Requerimiento <<3>>

Descripción

```
def req_3(data_structs,anio):
    """
    Función que soluciona el requerimiento 3
    """
    lista_total_combined = lista_subsectores_combined(data_structs,anio,get_subsector_iterator(get_value(data_structs["data"],anio)))
    menor_retenciones = menor_elemento(lista_total_combined,"Total retenciones")

    return menor_retenciones, SortLista(get_info(get_subsector_con_anio(data_structs,anio, menor_retenciones["Código subsector económico"])),cmp_menor_Total_retenciones)
```

```
def lista_subsectores_combined(data_structs,anio,iterator):

    lista = lt.newList("SINGLE_LINKED")

    for subsector in lt.iterator(iterator):
        lt.addLast(lista, lt.firstElement(get_combined(get_subsector_con_anio(data_structs,anio,subsector))))

    return lista
```

```
def menor_elemento(lista,criterio):

    menor_valor = limpiar_datos(lt.firstElement(lista)[criterio])
    menor = lt.firstElement(lista)

    for data in lt.iterator(lista):

        if limpiar_datos(data[criterio]) <= menor_valor:
            menor = data
            menor_valor = limpiar_datos(data[criterio])

    return menor
```

Este requerimiento comienza obteniendo el hash del año entrante. Como segundo paso, está get subsector iterator, que nos da la lista de los subsectores que hay en ese año. Luego, lista subsectores combined crea una lista en la que se añade cada valor de la llave combined de cada subsector que está en el hash de la llave subsector en el hash del año entrante.

Menor elemento obtiene el menor elemento de una lista bajo un criterio que es una llave específica del diccionario como lo es “Total saldo a favor” y devuelve el diccionario con menor total retenciones en este caso.

Luego de encontrar ese combined de menor cantidad de total retenciones. Vamos a usar get_subsector_con_anio para sacar el subsector que tuvo la menor cantidad de total retenciones, y luego get info, para sacar la lista de elementos de ese subsector. Después, sorteamos esa lista por el total de retenciones, para que queden en orden de menor a mayor por ese criterio. Finalmente, devuelve el diccionario combined, y la lista ya sorteada de elementos del subsector con menos total de retenciones.

Entrada	Se solicita un año entre el 2012 y el 2021
Salidas	El subsector que tuvo el menor total de retenciones, con sus datos sumados. Las tres actividades económicas que menos aportaron y las tres actividades económicas que más aportaron al valor total de retenciones del subsector
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(s) s es el número de subsectores en el año solicitado.
Paso 4	O(s)
Paso 5	O(1)
Paso 6	O(1)
Paso 7	O(nlog(n)) n es el número de elementos del subsector con menor total retenciones del año solicitado
TOTAL	***O(nlog(n))***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0,576
5	0,654
10	0,719
20	0,900
30	0,981
50	1,136
80	1,375
100	1,578

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Tiempo de Ejecución Real @LP [ms]
1	0.357
5	0.563
10	0.704
20	0.826
30	0.881
50	1.038
80	1.086
100	1.121

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el subsector económico con el menor total de retenciones para el año 2013	0,576
5	el subsector económico con el menor total de retenciones para el año 2013	0,654
10	el subsector económico con el menor total de retenciones para el año 2013	0,719
20	el subsector económico con el menor total de retenciones para el año 2013	0,900
30	el subsector económico con el menor total de retenciones para el año 2013	0,981

50	el subsector económico con el menor total de retenciones para el año 2013	1,136
80	el subsector económico con el menor total de retenciones para el año 2013	1,375
100	el subsector económico con el menor total de retenciones para el año 2013	1,578

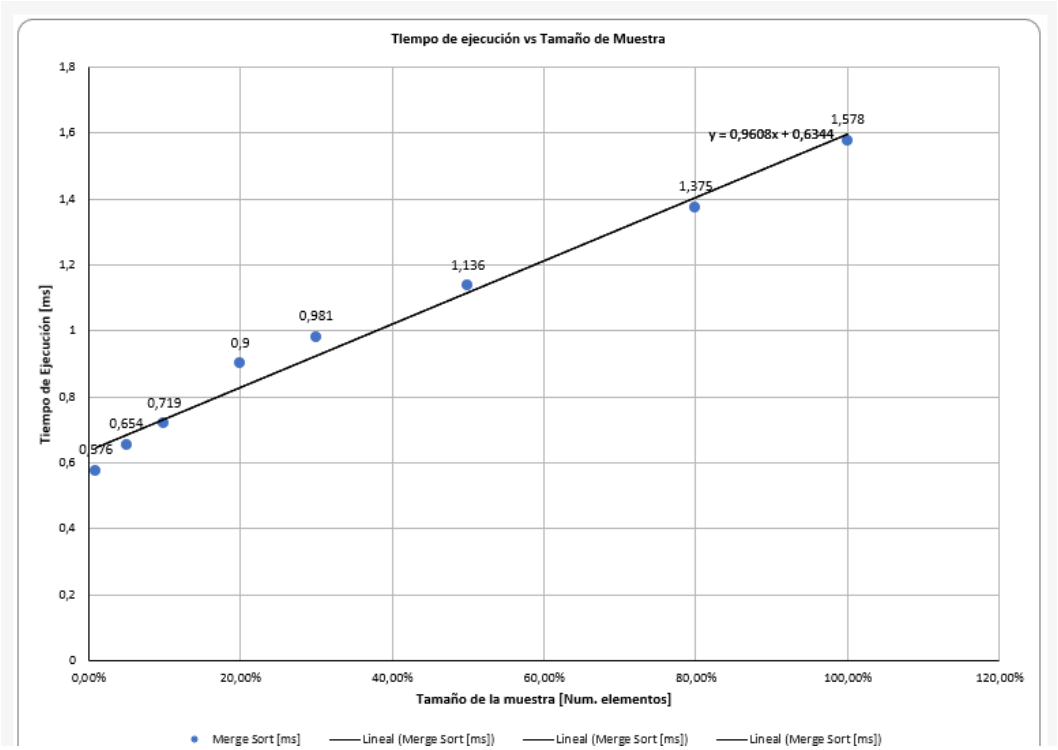
Máquina 2

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el subsector económico con el menor total de retenciones para el año 2013	0,357
5	el subsector económico con el menor total de retenciones para el año 2013	0,563
10	el subsector económico con el menor total de retenciones para el año 2013	0,704
20	el subsector económico con el menor total de retenciones para el año 2013	0,826
30	el subsector económico con el menor total de retenciones para el año 2013	0,881
50	el subsector económico con el menor total de retenciones para el año 2013	1,038
80	el subsector económico con el menor total de retenciones para el año 2013	1,086
100	el subsector económico con el menor total de retenciones para el año 2013	1,121

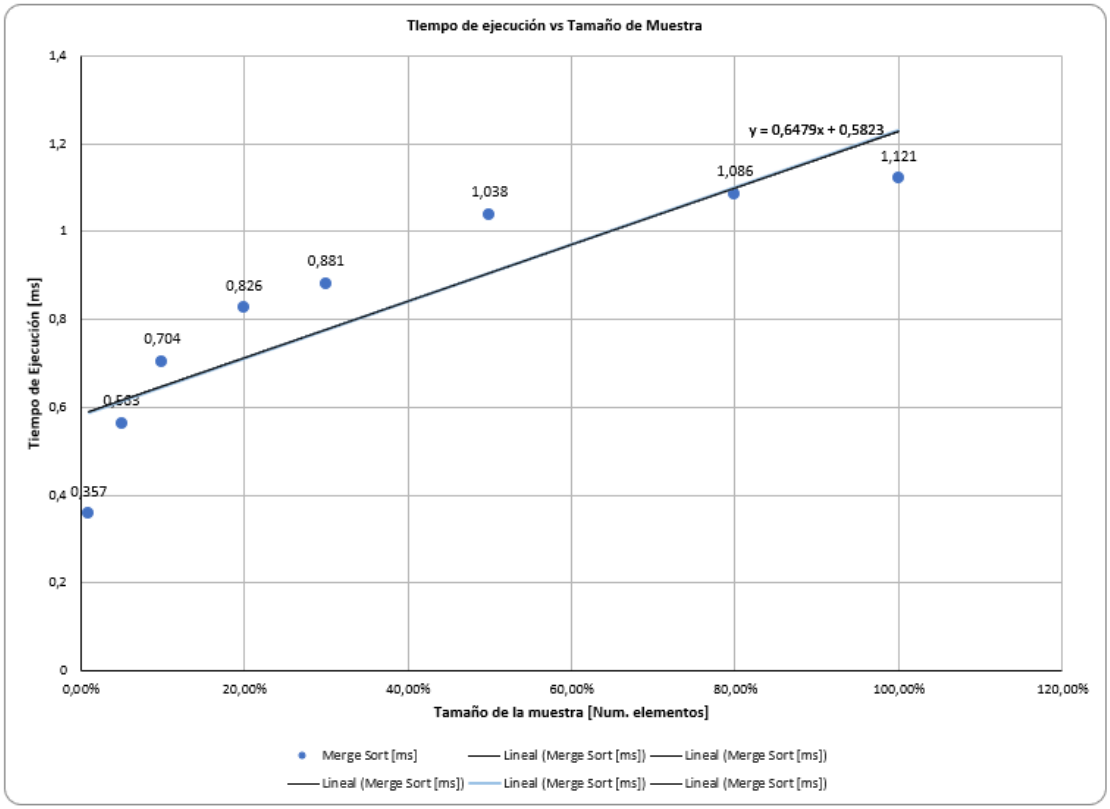
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1


Tiempo de ejecución vs Tamaño de Muestra
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
y = 0,9608x + 0,6344
0,576
0,654
0,719
0,9
0,981
1,136
1,375
1,578
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
0,00%
20,00%
40,00%
60,00%
80,00%
100,00%
120,00%
Merge Sort [ms]
Lineal (Merge Sort [ms])
Lineal (Merge Sort [ms])
Lineal (Merge Sort [ms])

Máquina 2



Análisis

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

En la teoría la complejidad del requerimiento sería N Log N, puesto que la operación de mayor complejidad es organizar la lista por total retenciones con el algoritmo mergesort que tiene la complejidad n log n. Esto se debe a que las otras operaciones del requerimiento es un recorrido de un for para buscar el menor elemento con retenciones que sería O(N). Es importante mencionar que aunque en la gráfica muestre una función lineal, el algoritmo si se comporta como n log n pero en la gráfica no se encontró una opción para representar una ecuación de tal complejidad., pero en este caso también cabe recalcar que n no es toda la cantidad de datos, ya que se está mirando u organizando una lista de un subsector en específico. La complejidad fue menor al Reto No 1 ya que se pasó de O(N^2) a O (N Log N) y consecuentemente los tiempos de ejecución fueron mucho menor.

Requerimiento <<4>>

Descripción

```
def req_4(data_structs, anio):
    """
    Función que soluciona el requerimiento 4
    """
    lista_total_combined = lista_subsectores_combined(data_structs,anio,get_subsector_iterator(get_value(data_structs["data"],anio)))
    mayor_costos_gastos_nomina = mayor_elemento(lista_total_combined,"Costos y gastos nómina")

    return mayor_costos_gastos_nomina, SortLista(get_info(get_subsector_con_anio(data_structs,anio,mayor_costos_gastos_nomina["Código subsector económico"])),cmp_menor_costos_gastos_nomi
```

```
def lista_subsectores_combined(data_structs,anio,iterator):

    lista = lt.newList("SINGLE_LINKED")

    for subsector in lt.iterator(iterator):
        lt.addLast(lista, lt.firstElement(get_combined(get_subsector_con_anio(data_structs,anio,subsector))))

    return lista
```

```
def mayor_elemento(lista,criterio):

    mayor_valor = limpiar_datos(lt.firstElement(lista)[criterio])
    mayor = lt.firstElement(lista)

    for data in lt.iterator(lista):

        if limpiar_datos(data[criterio]) >= mayor_valor:
            mayor = data
            mayor_valor = limpiar_datos(data[criterio])

    return mayor
```

Este requerimiento comienza obteniendo el hash del año entrante. Como segundo paso, está get subsector iterator, que nos da la lista de los subsectores que hay en ese año. Luego, lista subsectores combined crea una lista en la que se añade cada valor de la llave combined de cada subsector que está en el hash de la llave subsector en el hash del año entrante.

Mayor elemento obtiene el mayor elemento de una lista bajo un criterio que es una llave específica del diccionario como lo es “Total saldo a pagar” y devuelve el diccionario con mayor costos y gastos nómina en este caso.

Luego de encontrar ese combined de mayor cantidad de total retenciones. Vamos a usar get_subsector_con_anio para sacar el subsector que tuvo la mayor cantidad de costos y gastos nóminas, y luego get info, para sacar la lista de elementos de ese subsector. Después, sorteamos esa lista por los costos y gastos de nómina, para que queden en orden de menor a mayor por ese criterio. Finalmente, devuelve el diccionario combined, y la lista ya sorteada de elementos del subsector con mayor costos y gastos nómina.

Entrada	Se solicita un año entre el 2012 y el 2021, y el número de un sector
Salidas	El subsector que tuvo el mayor costos y gastos nómina, con sus datos sumados. Las tres actividades económicas que menos aportaron y las tres actividades económicas que más aportaron al valor costo y gastos nómina del subsector
Implementado (Sí/No)	Si está implementado y fue hecho por Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(s) s es el número de subsectores en el año solicitado.
Paso 4	O(s)
Paso 5	O(1)
Paso 6	O(1)
Paso 7	O(mlog(m)) m es el número de elementos del subsector con mayor costos y gastos nómina del año solicitado
TOTAL	***O(mlog(m))***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0,517
5	0,875
10	1,573
20	1,988
30	5,520
50	8,580
80	10,818
100	12,130

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Tiempo de Ejecución Real @LP [ms]
1	0.445
5	0.909
10	1.639
20	2.446
30	7.428
50	11.557
80	19.982
100	24.022

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el subsector económico con los mayores costos y gastos de nómina para el año 2021	0,517
5	el subsector económico con los mayores costos y gastos de nómina para el año 2021	0,875
10	el subsector económico con los mayores costos y gastos de nómina para el año 2021	1,573
20	el subsector económico con los mayores costos y gastos de nómina para el año 2021	1,988
30	el subsector económico con los mayores costos y gastos de nómina para el año 2021	5,520

50	el subsector económico con los mayores costos y gastos de nómina para el año 2021	8,580
80	el subsector económico con los mayores costos y gastos de nómina para el año 2021	10,818
100	el subsector económico con los mayores costos y gastos de nómina para el año 2021	12,130

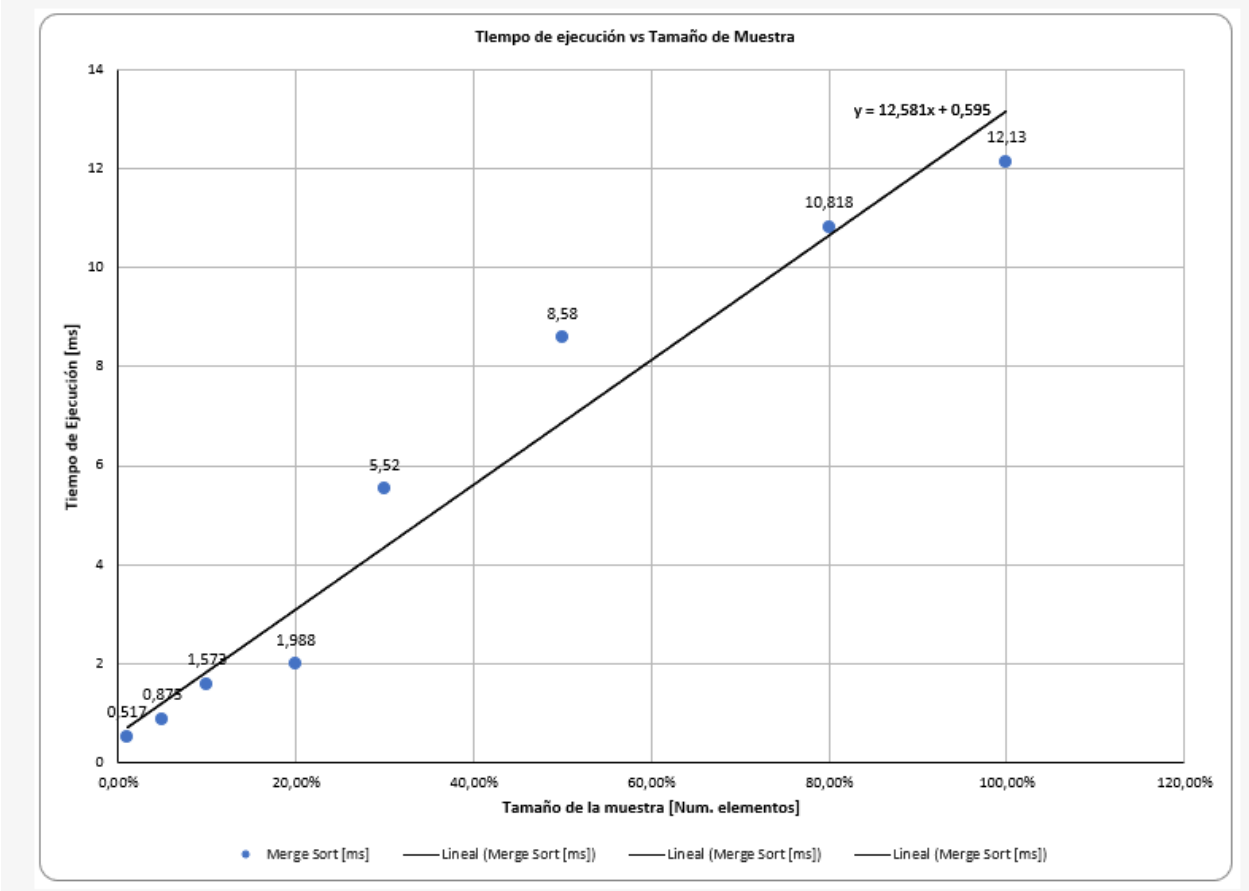
Máquina 2

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el subsector económico con los mayores costos y gastos de nómina para el año 2021	0,445
5	el subsector económico con los mayores costos y gastos de nómina para el año 2021	0,909
10	el subsector económico con los mayores costos y gastos de nómina para el año 2021	1,639
20	el subsector económico con los mayores costos y gastos de nómina para el año 2021	2,446
30	el subsector económico con los mayores costos y gastos de nómina para el año 2021	7,428
50	el subsector económico con los mayores costos y gastos de nómina para el año 2021	11,557
80	el subsector económico con los mayores costos y gastos de nómina para el año 2021	19,982
100	el subsector económico con los mayores costos y gastos de nómina para el año 2021	24,022

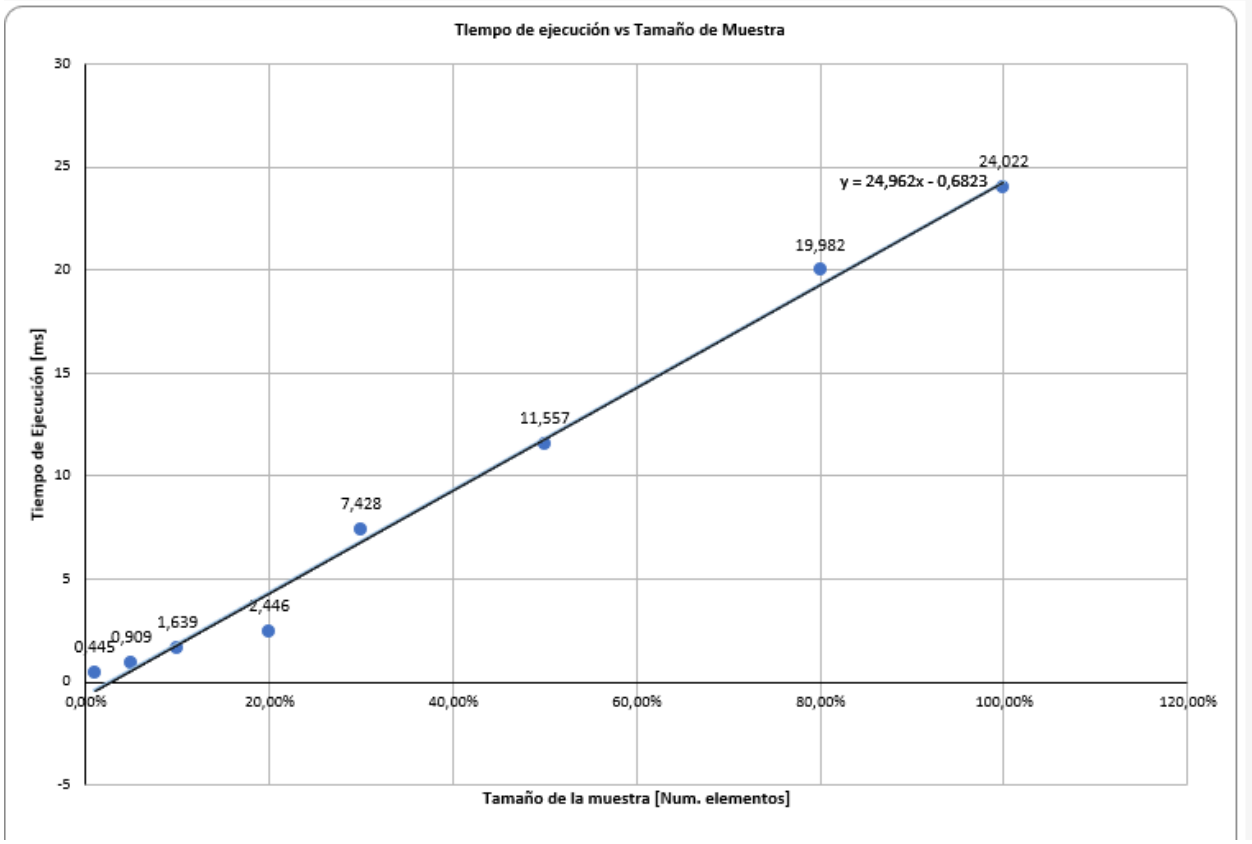
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1



Máquina 2



Análisis

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

En la teoría la complejidad del requerimiento sería N Log N, puesto que la operación de mayor complejidad es organizar la lista por total retenciones con el algoritmo mergesort que tiene la complejidad n log n. Esto se debe a que las otras operaciones del requerimiento es un recorrido de un for para buscar el mayor elemento con el criterio de costos y gastos que sería O(N). Es importante mencionar que aunque en la gráfica muestre una función lineal, el algoritmo si se comporta como n log n pero en la gráfica no se encontró una opción para representar una ecuación de tal complejidad., pero en este caso también cabe recalcar que n no es toda la cantidad de datos, ya que se está mirando u organizando una lista de un subsector en específico. La complejidad fue menor al Reto No 1 ya que se pasó de O(N^2) a O (N Log N) y consecuentemente los tiempos de ejecución fueron mucho menor.

Requerimiento <<5>>

Descripción

```
def req_5(data_structs,anio):
    """
    Función que soluciona el requerimiento 5
    """
    lista_total_combined = lista_subsectores_combined(data_structs,anio,get_subsector_iterator(get_value(data_structs["data"],anio)))
    mayor_descuentos = mayor_elemento(lista_total_combined,"Descuentos tributarios")

    return mayor_descuentos,  SortLista(get_info(get_subsector_con_anio(data_structs,anio,mayor_descuentos["Código subsector económico"])),cmp_menor_descuentos_tributarios)
```

```
def lista_subsectores_combined(data_structs,anio,iterator):

    lista = lt.newList("SINGLE_LINKED")

    for subsector in lt.iterator(iterator):
        lt.addLast(lista, lt.firstElement(get_combined(get_subsector_con_anio(data_structs,anio,subsector))))

    return lista
```

```
def mayor_elemento(lista,criterio):

    mayor_valor = limpiar_datos(lt.firstElement(lista)[criterio])
    mayor = lt.firstElement(lista)

    for data in lt.iterator(lista):

        if limpiar_datos(data[criterio]) >= mayor_valor:
            mayor = data
            mayor_valor = limpiar_datos(data[criterio])

    return mayor
```

Este requerimiento comienza obteniendo el hash del año entrante. Como segundo paso, está get subsector iterator, que nos da la lista de los subsectores que hay en ese año. Luego, lista subsectores combined crea una lista en la que se añade cada valor de la llave combined de cada subsector que está en el hash de la llave subsector en el hash del año entrante.

Mayor elemento obtiene el mayor elemento de una lista bajo un criterio que es una llave específica del diccionario como lo es “Descuentos Tributarios” y devuelve el diccionario con mayor descuentos tributarios en este caso.

Luego de encontrar ese combined de mayor cantidad de descuentos tributarios. Vamos a usar get_subsector_con_anio para sacar el subsector que tuvo la mayor cantidad de descuentos tributarios, y luego get info, para sacar la lista de elementos de ese subsector. Después, sorteamos esa lista por los descuentos tributarios, para que queden en orden de menor a mayor por ese criterio. Finalmente, devuelve el diccionario combined, y la lista ya sorteada de elementos del subsector con mayor descuento tributarios.

Entrada	Se solicita un año entre el 2012 y el 2021, y el número de un sector
Salidas	El subsector que tuvo el mayor descuento tributario, con sus datos sumados. Las tres actividades económicas que menos aportaron y las tres actividades económicas que más aportaron al valor descuentos tributarios del subsector
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(s) s es el número de subsectores en el año solicitado.
Paso 4	O(s)
Paso 5	O(1)
Paso 6	O(1)
Paso 7	O(mlog(m)) m es el número de elementos del subsector con mayor descuentos tributarios del año solicitado
TOTAL	***O(mlog(m))***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos

Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0,520
5	0,798
10	1,168
20	1,823
30	5,017
50	7,509
80	6,345
100	10,067

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0.442

5	0.817
10	1.716
20	2.390
30	6.974
50	10.653
80	3.569
100	22.530

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el subsector económico con los mayores descuentos tributarios para el año 2021	0,520
5	el subsector económico con los mayores descuentos tributarios para el año 2021	0,798
10	el subsector económico con los mayores descuentos tributarios para el año 2021	1,168
20	el subsector económico con los mayores descuentos tributarios para el año 2021	1,823
30	el subsector económico con los mayores descuentos tributarios para el año 2021	5,017
50	el subsector económico con los mayores descuentos tributarios para el año 2021	7,509
80	el subsector económico con los mayores descuentos tributarios para el año 2021	6,345

100	el subsector económico con los mayores descuentos tributarios para el año 2021	10,067

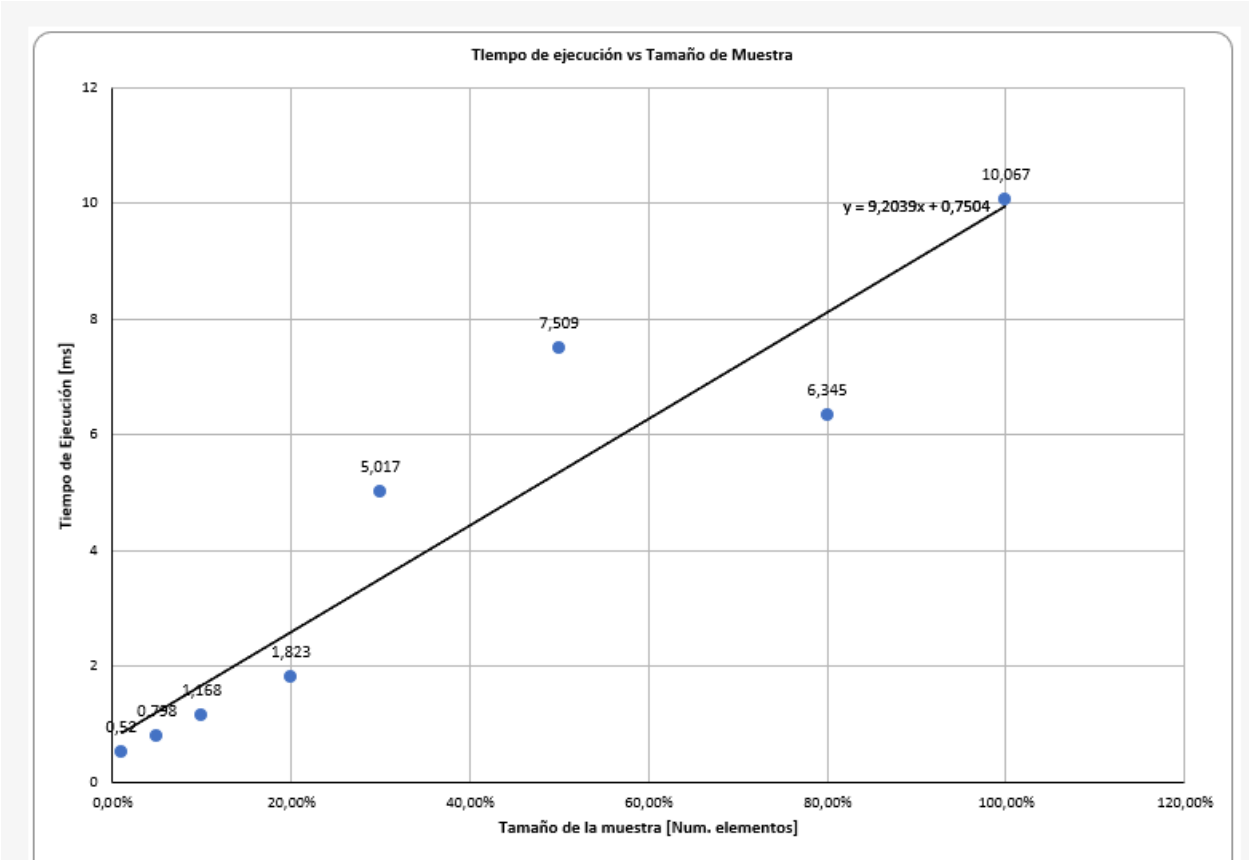
Máquina 2

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el subsector económico con los mayores descuentos tributarios para el año 2021	0,442
5	el subsector económico con los mayores descuentos tributarios para el año 2021	0,817
10	el subsector económico con los mayores descuentos tributarios para el año 2021	1,716
20	el subsector económico con los mayores descuentos tributarios para el año 2021	2,390
30	el subsector económico con los mayores descuentos tributarios para el año 2021	6,974
50	el subsector económico con los mayores descuentos tributarios para el año 2021	10,653
80	el subsector económico con los mayores descuentos tributarios para el año 2021	13,569
100	el subsector económico con los mayores descuentos tributarios para el año 2021	22,530

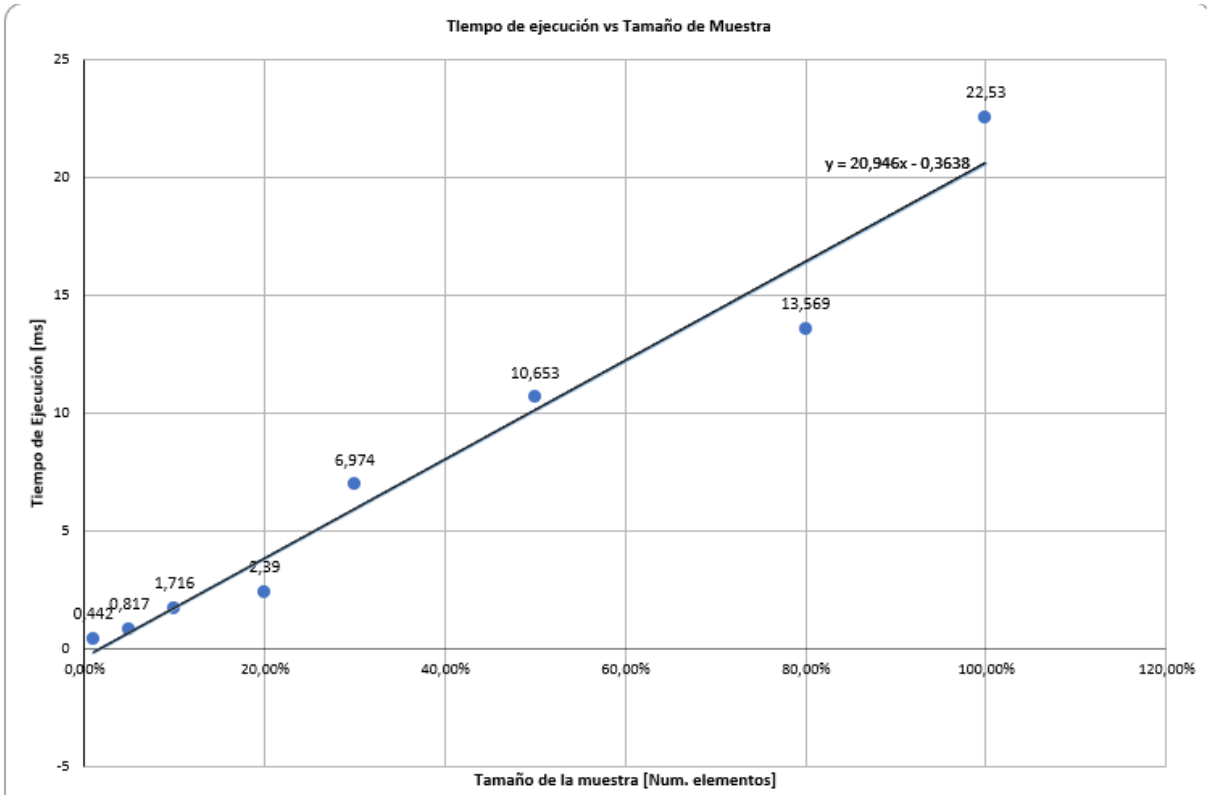
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1


Tlempo de ejecución vs Tamaño de Muestra
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
y = 9,2039x + 0,7504
10,067
7,509
6,345
5,017
1,823
1,168
0,798
0,52
0
2
4
6
8
10
12
0,00%
20,00%
40,00%
60,00%
80,00%
100,00%
120,00%

Máquina 2



Análisis

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

En la teoría la complejidad del requerimiento sería N Log N, puesto que la operación de mayor complejidad es organizar la lista por total retenciones con el algoritmo mergesort que tiene la complejidad n log n. Esto se debe a que las otras operaciones del requerimiento es un recorrido de un for para buscar el mayor elemento con mayores descuentos tributarios que sería O(N). Es importante mencionar que aunque en la gráfica muestre una función lineal, el algoritmo si se comporta como n log n pero en la gráfica no se encontró una opción para representar una ecuación de tal complejidad., pero en este caso también cabe recalcar que n no es toda la cantidad de datos, ya que se está mirando u organizando una lista de un subsector en específico. La complejidad fue menor al Reto No 1 ya que se pasó de O(N^2) a O (N Log N) y consecuentemente los tiempos de ejecución fueron mucho menor.

Requerimiento <<6>>

Descripción

```
def req_6(data_structs,anio):
    """
    Función que soluciona el requerimiento 6
    """

    lista_sectores = lista_sectores_combined(data_structs,anio,get_sector_iterator(get_value(data_structs["data"],anio)))
    sector_mayor_ingresos_netos =  mayor_elemento(lista_sectores,"Total ingresos netos")

    lista_subsectores = lista_subsectores_del_sector_combined(get_subsector_iterator(get_sector_con_anio(data_structs,anio,sector_mayor_ingresos_netos["Código sector económico"])),ge
    subsector_mayor = mayor_elemento(lista_subsectores,"Total ingresos netos")
    subsector_menor = menor_elemento(lista_subsectores, "Total ingresos netos")

    actividad_mayor_mayor = mayor_elemento(get_info( get_subsector_especifico(get_sector_con_anio(data_structs,anio,sector_mayor_ingresos_netos["Código sector económico"]), subsector
    actividad_menor_mayor = menor_elemento(get_info( get_subsector_especifico(get_sector_con_anio(data_structs,anio,sector_mayor_ingresos_netos["Código sector económico"]), subsector
    actividad_mayor_menor = mayor_elemento(get_info( get_subsector_especifico(get_sector_con_anio(data_structs,anio,sector_mayor_ingresos_netos["Código sector económico"]), subsector
    actividad_menor_menor = menor_elemento(get_info( get_subsector_especifico(get_sector_con_anio(data_structs,anio,sector_mayor_ingresos_netos["Código sector económico"]), subsector


    subsector_mayor["Actividad económica que más aportó"] = actividad_mayor_mayor
    subsector_mayor["Actividad económica que menos aportó"] = actividad_menor_mayor
    subsector_menor["Actividad económica que más aportó"] = actividad_mayor_menor
    subsector_menor["Actividad económica que menos aportó"] = actividad_menor_menor

    return sector_mayor_ingresos_netos,subsector_mayor,subsector_menor

"""
```

```
nico"])),get_sector_con_anio(data_structs,anio,sector_mayor_ingresos_netos["Código sector económico"]))



 subsector_mayor["Código subsector económico"])),"Total ingresos netos")
 subsector_mayor["Código subsector económico"])),"Total ingresos netos")
 subsector_menor["Código subsector económico"])),"Total ingresos netos")
 subsector_menor["Código subsector económico"])),"Total ingresos netos")
```

La primera línea crea una lista con la llave “combined” de cada sector del año solicitado, es decir esta llave tiene como valor una lista donde cada elemento es el diccionario de los distintos sectores con las columnas sumadas. Esto se hace para posteriormente encontrar cual es el mayor elemento de esta lista que corresponde al diccionario del sector con el mayo total de ingresos netos.

Después se hace básicamente el mismo proceso que el anterior pero con los subsectores. Puesto que al obtener el sector mayor, se accede al hash de ese sector y se revisa su llave de subsectores. Todos los diferentes subsectores se añaden a una lista que se llama lista subsectores, para luego poder determinar cual es el mayor y menor elemento de esa lista que corresponden al mayor y menor subsector del sector mayor encontrado.

Como paso final, se realiza el mismo proceso pero con las actividades económicas del hash del menor y mayor subsector. Es decir que hay 4 actividades económicas distintas que se guardan en el diccionario del subsector mayor y menor que sería el mayor y menor actividad económica del subsector mayor y la mayor y menor actividad del subsector menor. De esta manera se cumpliría el requerimiento con una complejidad de o(n) puesto que la única operación que utiliza un for es mayor y menor elemento, el resto son operaciones de o(1) porque se obtienen de un get del mapa no ordenado.

Entrada	Se solicita un año entre el 2012 y el 2021,
Salidas	el sector económico con el mayor total de ingresos netos para un año específico
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(m), m siendo el número de elementos que tienen el año y sector solicitado
Paso 4	o(m), todas las operaciones de mayor y menor elemento asociadas a este paso son o(m)
TOTAL	***O(m)***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	1,032

5	1,356
10	1,748
20	2,001
30	2,019
50	2,298
80	2,534
100	2,608

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	1,286
5	1,353
10	1,825
20	2,156
30	2,268
50	2,398
80	2,537

100	2,948

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	Salida	Tiempo de Ejecución [ms]
1	el sector económico con el mayor total de ingresos netos para el año 2021	1,032
5	el sector económico con el mayor total de ingresos netos para el año 2021	1,356
10	el sector económico con el mayor total de ingresos netos para el año 2021	1,748
20	el sector económico con el mayor total de ingresos netos para el año 2021	2,001
30	el sector económico con el mayor total de ingresos netos para el año 2021	2,019
50	el sector económico con el mayor total de ingresos netos para el año 2021	2,298
80	el sector económico con el mayor total de ingresos netos para el año 2021	2,534
100	el sector económico con el mayor total de ingresos netos para el año 2021	2,608

Máquina 2

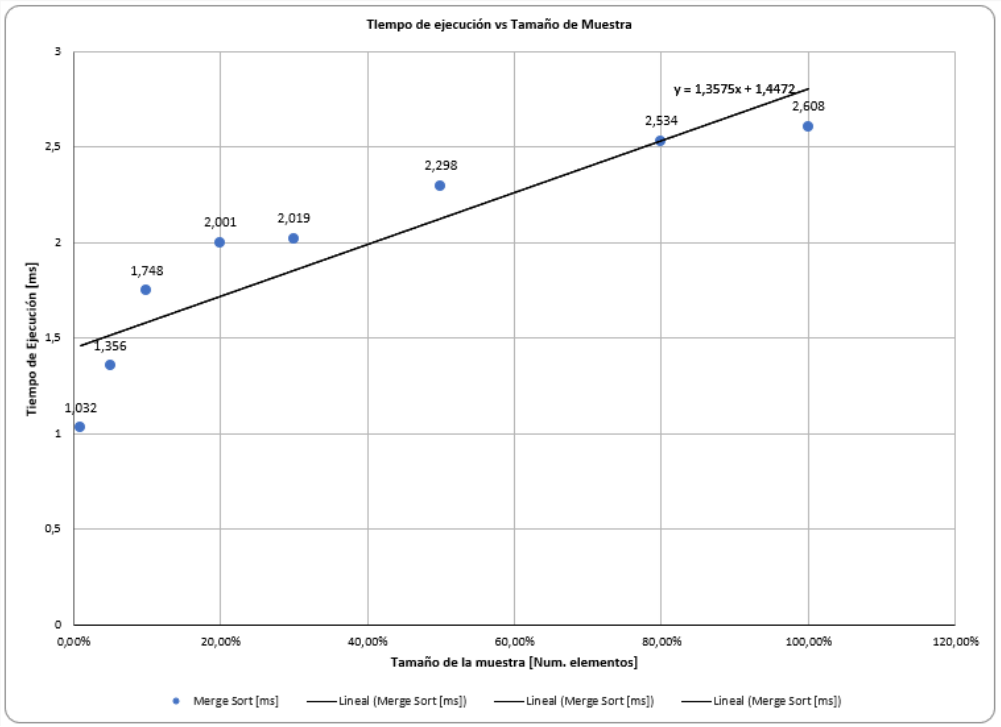
Muestra%	Salida	Tiempo de Ejecución [ms]
1	el sector económico con el mayor total de ingresos netos para el año 2021	1,286

5	el sector económico con el mayor total de ingresos netos para el año 2021	1,353
10	el sector económico con el mayor total de ingresos netos para el año 2021	1,825
20	el sector económico con el mayor total de ingresos netos para el año 2021	2.156
30	el sector económico con el mayor total de ingresos netos para el año 2021	2,268
50	el sector económico con el mayor total de ingresos netos para el año 2021	2.398
80	el sector económico con el mayor total de ingresos netos para el año 2021	2.537
100	el sector económico con el mayor total de ingresos netos para el año 2021	2.948

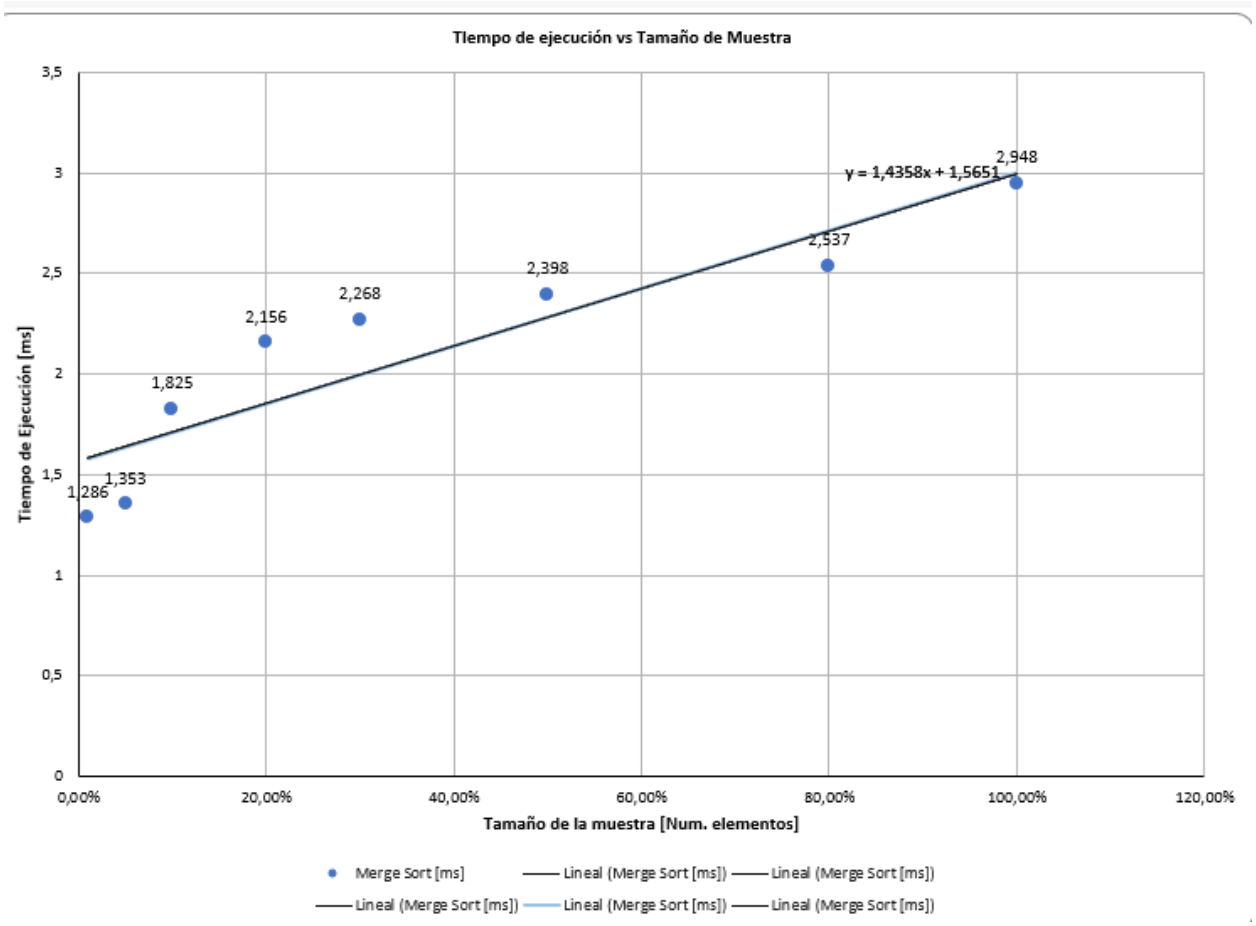
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1:


Tlempo de ejecución vs Tamaño de Muestra
y = 1,3575x + 1,4472
2,608
2,534
2,298
2,001
2,019
1,748
1,356
1,032
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
Merge Sort [ms]
Lineal (Merge Sort [ms])
Lineal (Merge Sort [ms])
Lineal (Merge Sort [ms])

Máquina 2



Análisis

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

En la teoría la complejidad de este requerimiento sería O(N) puesto que básicamente todas las operaciones que se realizan son obtener valores de un hash lo cual tiene complejidad O(1) y después de obtener esos elementos o listas, se hace un recorrido sobre ese dato para obtener el ayor o menor de un dato con la función mayor _elemento y menor elemento explicadas previamente en este informe cuando se utilizaron en los primeros dos requerimientos y la complejidad de esta búsqueda sería de O(N). Adicionalmente se puede ver en la gráfica que la línea de tendencia tiende a una lineal confirmando y verificando que la complejidad si sería de O(n). La complejidad fue menor al Reto No 1 ya que se pasó de O(N^3) a O (N) y consecuentemente los tiempos de ejecución fueron mucho menor, es decir la mejoría con hash fue muy amplia en especial en este requerimiento.

Requerimiento <<7>>

Descripción

```
def req_7(data_structs,anio,subsector,top):
    """
    Función que soluciona el requerimiento 7
    """
    top = int(top)
    lista_info_organizada = SortLista(get_info(get_subsector_con_anio(data_structs,anio,subsector)),cmp_menor_Total_costos_gastos)

    if lt.size(lista_info_organizada) < top:
        return lista_info_organizada
    else:
        return lt.subList(lista_info_organizada,1,top)
"""
```

Este requerimiento tiene 3 pasos, en primera instancia se obtiene el hash de un subsector específico de un año específico a través de las funciones previamente explicadas. Después de obtener este hash se obtiene la lista con la información de este subsector en específico y se sortea con una cmp de Total costos y gastos organizada de mayor a menor. Finalmente el if y else verifican que el top (n) es posible, en otras palabras si se desea el top 9 pero solo hay 2 datos solo se devuelven esos dos datos, en el caso de que si se pueda hacer el top se hace una sublista de la lista obtenida.

Entrada	Se solicita un año entre el 2012 y el 2021, el número (n) de actividades económicas, y el código de subsector económico
Salidas	el TOP (N) de las actividades económicas con el menor total de costos y gastos para un subsector y un año específicos
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(m log m) donde m es la cantidad de elementos de la lista obtenida de ese subsector
Paso 3	O(1)
TOTAL	***O(m log m)***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Tiempo de Ejecución Real @LP [ms]
1	0,085
5	0,090
10	0,257
20	0,494
30	0,588
50	1,869
80	3,637
100	5,600

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	0,065
5	0,076
10	0,210
20	0,520
30	0,878
50	2,665
80	5,803
100	6,423

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	**Salida**	**Tiempo de Ejecución [ms]**
1	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,085

5	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,090
10	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,257
20	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,494
30	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,588
50	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	1,869
80	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	3,637
100	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	5,600

Máquina 2

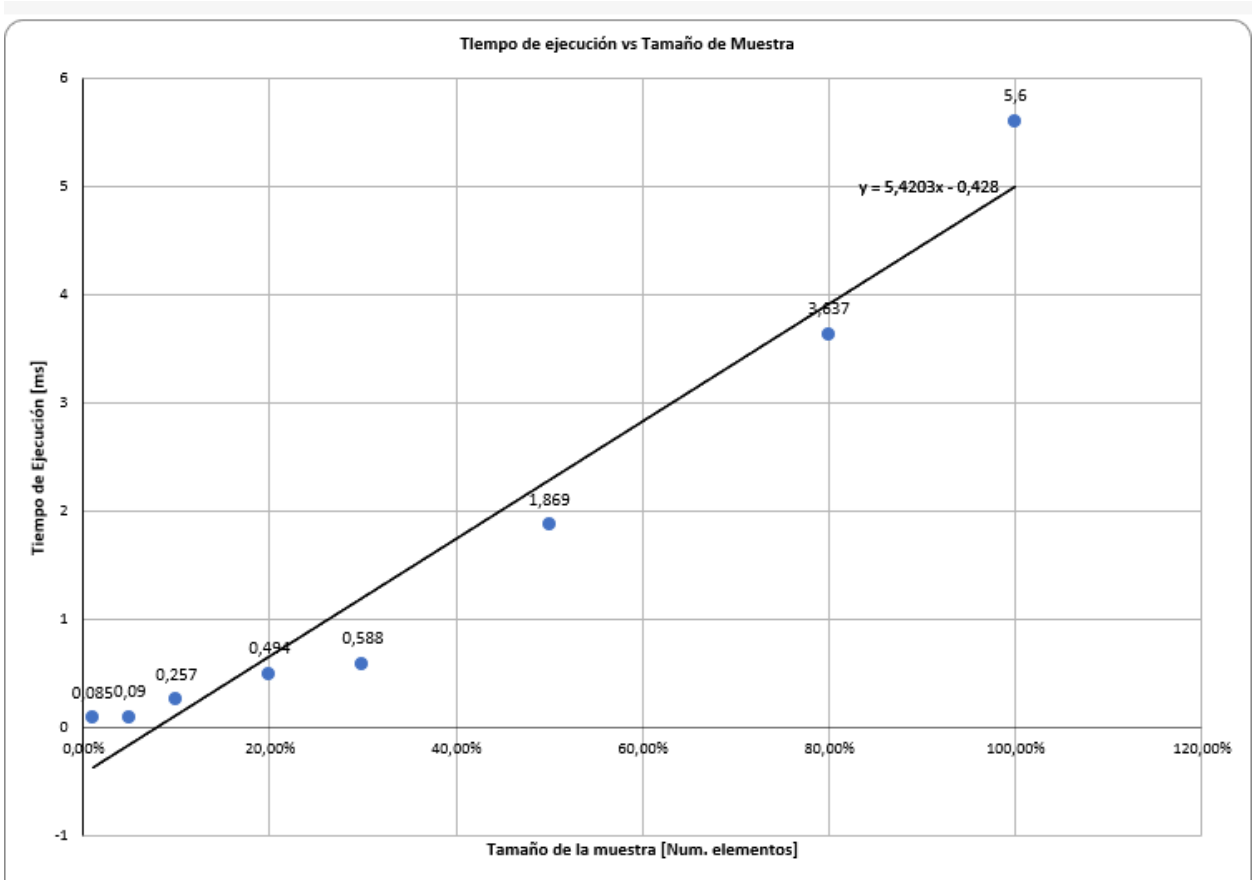
1	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,065
5	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,076
10	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,210

20	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,520
30	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	0,878
50	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	2,665
80	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	5,803
100	el TOP (9) de las actividades económicas con el menor total de costos y gastos para el subsector 11 en el año 2020	6,423

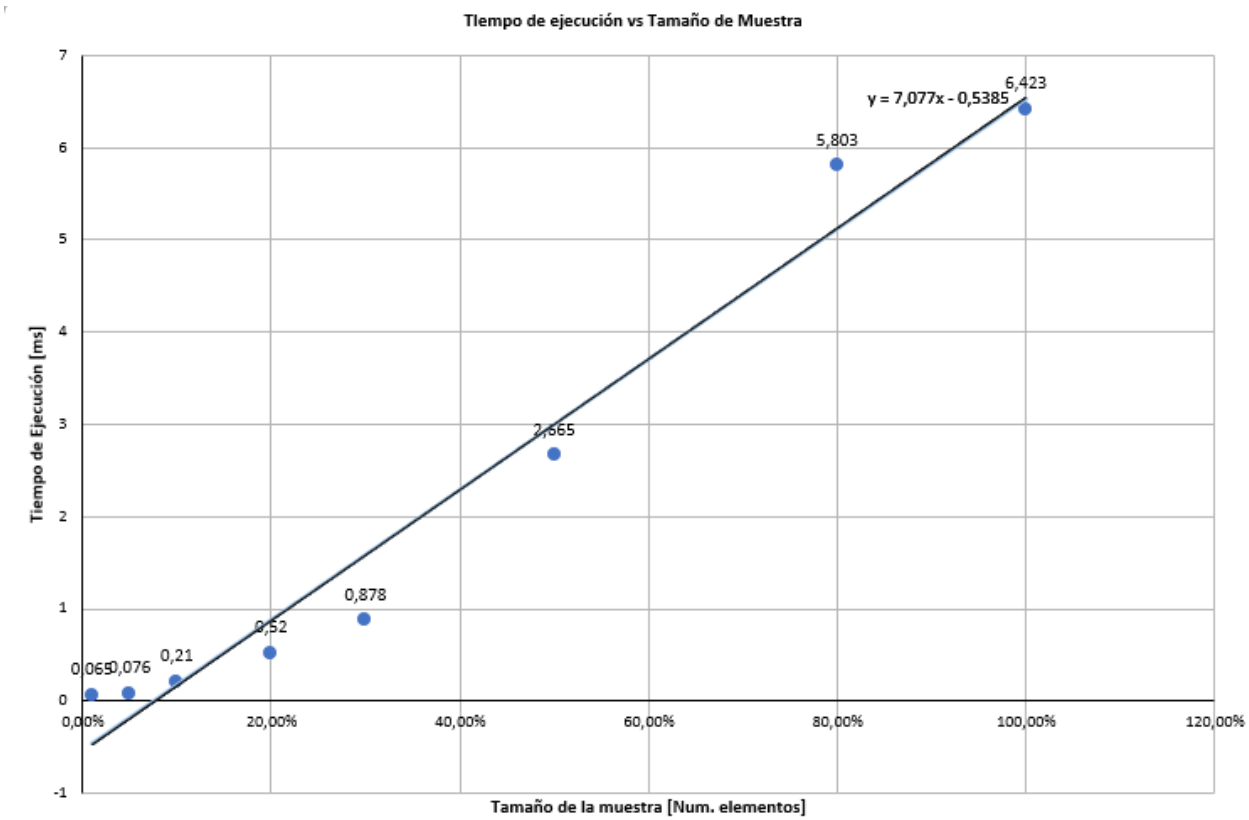
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1:


Tlempo de ejecución vs Tamaño de Muestra
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
y = 5,4203x - 0,428
5,6
3,637
1,869
0,588
0,494
0,257
0,09
0,085
6
5
4
3
2
1
0
-1
0,00%
20,00%
40,00%
60,00%
80,00%
100,00%
120,00%

Máquina 2



Análisis

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

En la teoría la complejidad de este requerimiento sería O(N Log N) puesto que solo hay tres tipos de operaciones en este requerimiento que sería obtener la información de un hashmap que corresponde a O(1), crear una sublista, que al ser un ADT de disclib implementado su complejidad va a ser cercano a O(1) ya que incluso en el peor caso que se tenga que hacer una sublista de todos los elementos, el if y else se encargan de solo retornar la información completa como O(1), y la última operación es sortear una lista con un merge sort que tendría complejidad O(N log N). Además en la gráfica se ve un aumento lineal pronunciado que es más parecida a la complejidad de N log N puesto que no se encontró como representar una fórmula de esta complejidad. La complejidad fue menor al Reto No 1 ya que se pasó de O(N^2) a O (N Log N) y consecuentemente los tiempos de ejecución fueron mucho menor.

Requerimiento <<8>>

Descripción

```
top = int(top)
primera_lista = lt.newList("ARRAY_LIST")
varias_tablas = lt.newList("ARRAY_LIST")
for sector in lt.iterator(get_sector_iterator(get_value(data_structs["data"],anio))):
    subsector_mayor = mayor_elemento(lista_subsectores_del_sector_combined(get_subsector_iterator(get_sector_con_anio(data_structs,anio,sector)), get_sector_con_anio(data_structs,an
    lt.addLast(primera_lista,subsector_mayor)

SortLista(primera_lista,cmp_menor_sector_economico)
```

Primeramente para este requerimiento se creó una lista vacía, y luego se obtuvo la lista que pertenecía al hash del año, donde esa lista funciona como un iterador ya que cada elemento son los sectores presentes de ese año. Después se encontraba el mayor subsector que aportó en Total Impuestos a cargo, de ese sector y se añadía a la lista. Es decir que esta primera parte retorna la lista de el subsector que más aporta por cada subsector del año que se pasa por parámetro y se sortea por sector económico de menor a mayor. Lo cual tendría una complejidad de O(N^2) dentro del for porque es un for que adentro tiene otro for (la función lista subsectores del sector combined) es un for y el sorteo es O(N log N) ya que sería un sort por merge sort.

```
for subsector in lt.iterator(primera_lista):

    lista_info_organizada = SortLista(get_info(get_subsector_con_anio(data_structs,anio,subsector["Código subsector económico"])),cmp_Total_impuestos_cargo)

    if lt.size(lista_info_organizada) < top:
        lt.addLast(varias_tablas,lista_info_organizada)
    else:
        lt.addLast(varias_tablas,lt.subList(lista_info_organizada,1,top))

return primera_lista,varias_tablas
```

La segunda parte del requerimiento sería recorrer los subsectores que más aportaron encontrados previamente, y organizar la información de todos las actividades económicas de ese subsector de acuerdo al top (n) y sortear esa lista de elementos por el Total impuestos a cargo de mayor a menor para finalmente hacer unos ifs parecidos a los del requerimiento 7 para evitar que haya un error cuando el top(n) es mayor que la cantidad de elementos que hay.

Entrada	Se solicita un año entre el 2012 y el 2021, el número (n) de actividades económicas
Salidas	el TOP (N) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para un año específico.
Implementado (Sí/No)	Si está implementado y fue hecho por Eric Alarcon y Brainer Jimenez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1	O(1)
Paso 2	O(m^2) donde m es la cantidad de elementos de la lista obtenida de ese subsector
Paso 3	O(m log m)
Paso 4	O(m^2 log m)
Paso 5	O(1)
TOTAL	***O(m^2 log m)***

Pruebas Realizadas

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

Máquina 1

Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 procesadores principales, 8 procesadores lógicos
Memoria Ram	8 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	Tiempo de Ejecución Real @LP [ms]
1	1.105
5	3,688
10	5,302
20	12,014
30	14,886
50	27,942

80	35,009
100	42,999

Máquina 2

Procesadores	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
Memoria Ram	16 GB
Sistema Operativo	Windows 11

Entrada (Tamaño archivo %)	**Tiempo de Ejecución Real @LP [ms]**
1	1,531
5	3,824
10	8,449
20	16,043
30	24,143
50	46,662
80	72,873
100	93,708

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Máquina 1

Muestra%	**Salida**	**Tiempo de Ejecución [ms]**

1	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	1,531
5	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021.	3,688
10	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	5,302
20	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021.	12,014
30	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	14,886
50	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	27,942
80	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	35,009
100	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	42,999

Máquina 2

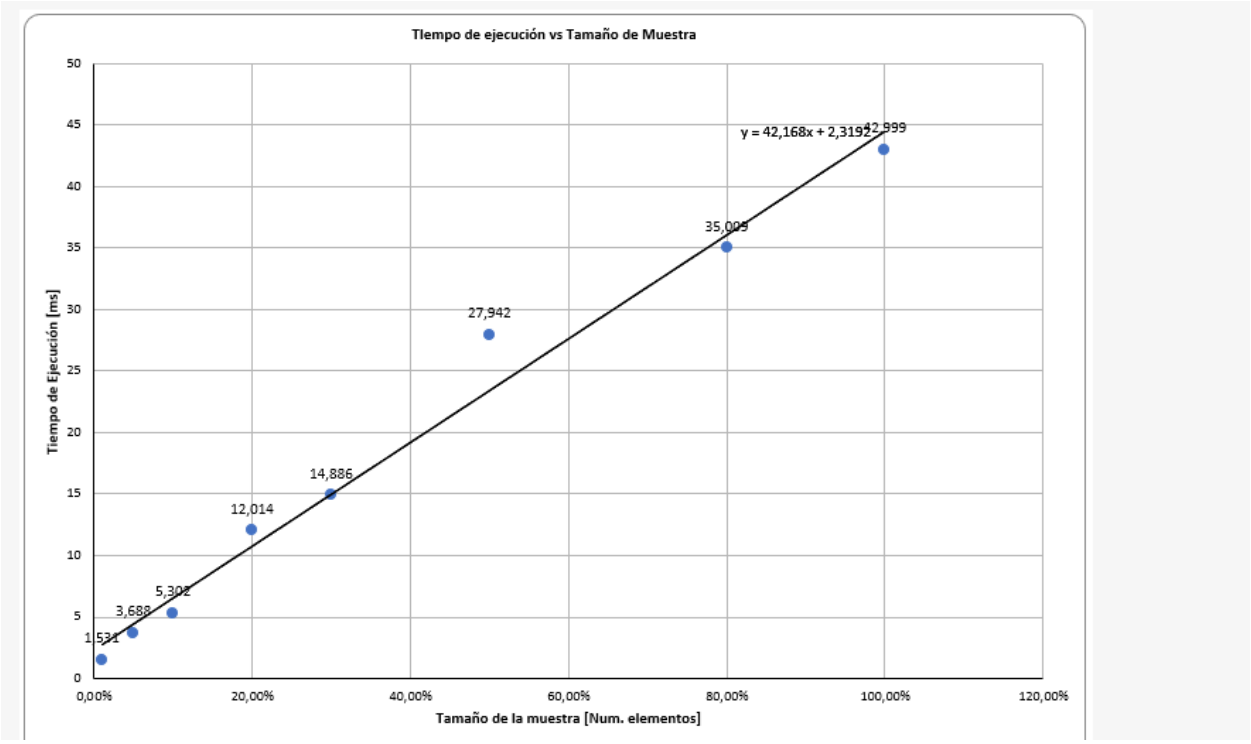
Muestra%	Salida	Tiempo de Ejecución [ms]
1	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	1,531

5	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021.	3,824
10	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	8,449
20	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021.	16,043
30	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	24,143
50	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	46,662
80	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	72,873
100	el TOP (3) de actividades económicas de cada subsector con los mayores totales de impuestos a cargo para el 2021	93,708

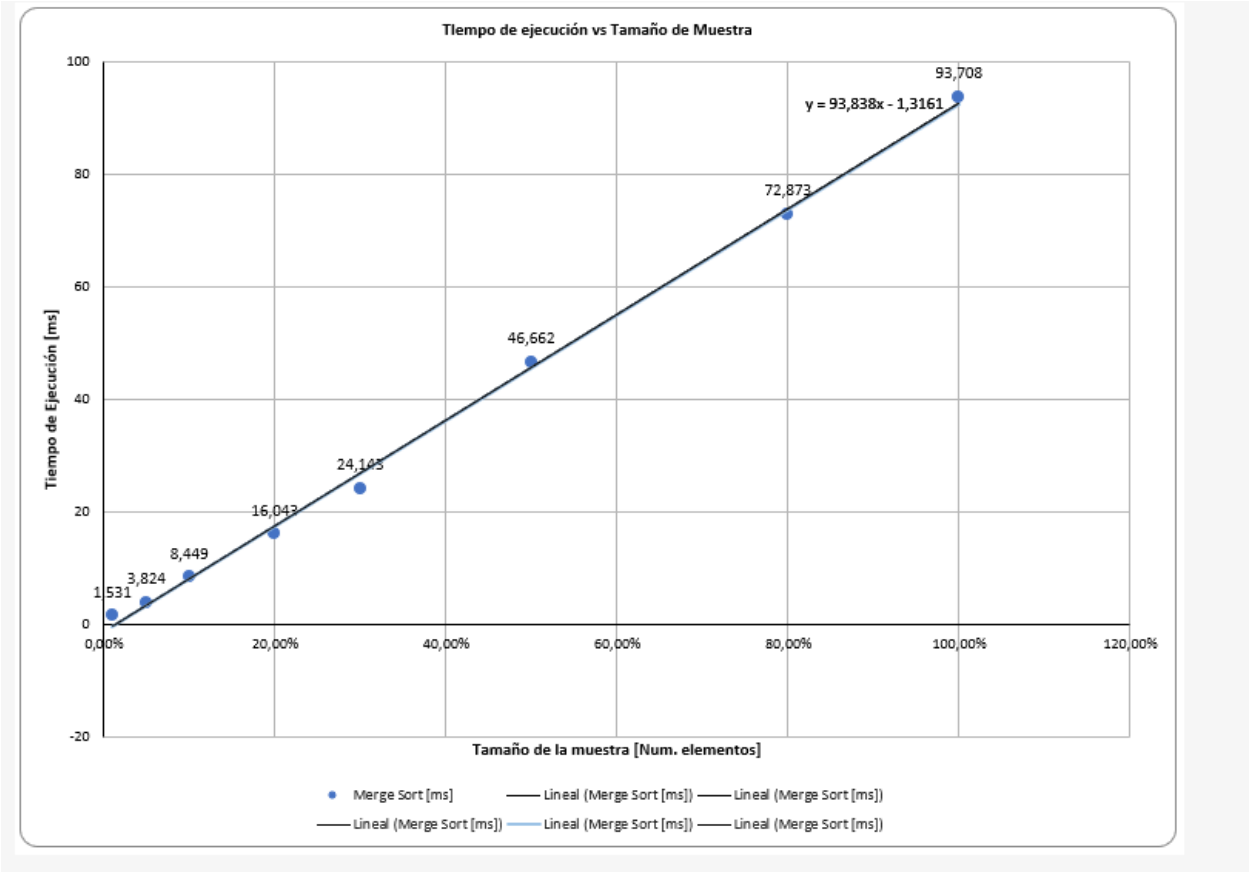
Gráficas

Las gráficas con la representación de las pruebas realizadas.

Máquina 1:


Tlempo de ejecución vs Tamaño de Muestra
Tiempo de Ejecución [ms]
Tamaño de la muestra [Num. elementos]
y = 42,168x + 2,3192
42,999
35,009
27,942
14,886
12,014
5,302
3,688
1,531
0
5
10
15
20
25
30
35
40
45
50
0,00%
20,00%
40,00%
60,00%
80,00%
100,00%
120,00%

Máquina 2



Análisis

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

En la teoría la complejidad de este requerimiento sería O(N^2 log N) porque el proceso de mayor complejidad sería recorrer un for y que dentro de este ciclo se estén organizando listas con un merge sort. No obstante, es peculiar notar que las gráficas demuestran un carácter más parecido a N log N, una explicación de esta incongruencia es que el for grande donde se sortea revisa cada sector diferente, y en el documento large no existen más de 11 subsectores distintos por lo que el for en magnitud es muy pequeño llegando a casi ser constante, por lo que la mayor complejidad podría llegar a ser la del sort. No obstante es importante recalcar que entre más datos se coloquen y si estos varían más en subsectores si se cumpliría el O(N^2 log N). No se puede comparar al Reto 1 porque en ese reto no se realizó el bono.