ANÁLISIS DEL RETO

Tomas Aponte, 202420148, t.aponte@uniandes.edu.co

Juan Diego García, 202423575jd.garcia12@uniandes.edu.co

Requerimiento <<1>>

```
def req_1(catalog,anno):
    """
    Retorna el resultado del requerimiento 1
    """
    # TODO: Modificar el requerimiento 1

    start_time=get_time()

    lista_fechas=al.new_list()
    info = {'source': None,
            'commodity': None,
               'unit_measurement': None,
               'state_name': None,
               'year_collection': None,
               'load_time': None,
               'value': None,}

    for i in range(catalog["year_collection"]["size"]):
        if catalog["year_collection"]["elements"][i] == str(anno):
            al.add_last(lista_fechas, catalog["load_time"]["elements"][i])
    al.merge_sort(lista_fechas, False)
    pasaron=al.size(lista_fechas)
    ultima=lista_fechas["elements"][0]

    for k in range(catalog["load_time"]["size"]):
        if (str(ultima) == catalog["load_time"]["elements"][k]) and (str(anno)==catalog["year_collection"]["elements"][k]):
            info["source"]=catalog["source"]["elements"][k]
            info["commodity"]=catalog["commodity"]["elements"][k]
            info["unit_measurement"]=catalog["unit_measurement"]["elements"][k]
            info["state_name"]=catalog["state_name"]["elements"][k]
            info["year_collection"]=catalog["year_collection"]["elements"][k]
            info["load_time"]=catalog["load_time"]["elements"][k]
            info["value"]=catalog["value"]["elements"][k]

    end_time=get_time()

    req_1_time=delta_time(start_time,end_time)

    return(info,req_1_time,pasaron)
```

Descripción

Este requerimiento se encarga de buscar y mostrar al usuario el último registro encontrado durante un año específico. Además, retorna sus características base. Lo que hace es crear un listado de fechas en donde se insertará la información de los registros que cumplan el filtro de búsqueda. Para encontrar dichos registros se ejecuta un ciclo el cual compara la información necesaria con la de todos los registros para así añadir los que cumplen el filtro. Una vez terminado el ciclo se ordena la lista de manera ascendente, de tal modo que el primer año en la lista sea el último que se encontró dentro del primer filtro. Luego se asigna la longitud de la lista ordenada para el retorno de la totalidad de registros que pasaron el filtro de búsqueda, y por último se hace un nuevo ciclo para encontrar la información del registro que se debe retornar.

Entrada	Catálogo con los datos del archivo csv, año deseado
Salidas	Información del último registro encontrado, totalidad de registros que cumplieron el filtro de búsqueda, tiempo de ejecución
Implementado (Sí/No)	Si, implementado por Juan García

Análisis de complejidad

Pasos	**Complejidad**
Asignaciones variables	O(1)
Recorrido total del catálogo (for i in range …) por filtro para insertar datos en **lista_fechas**	O(n)
Ordenamiento **lista_fechas**	O(n log n)
Nuevas variables asignadas	O(1)
Búsqueda del último registro para su retorno	O(n)
TOTAL	***O(n log n)***

Pruebas Realizadas

Año de interés: 2010

Procesador: Ryzen 7 7730U

RAM: 16GB

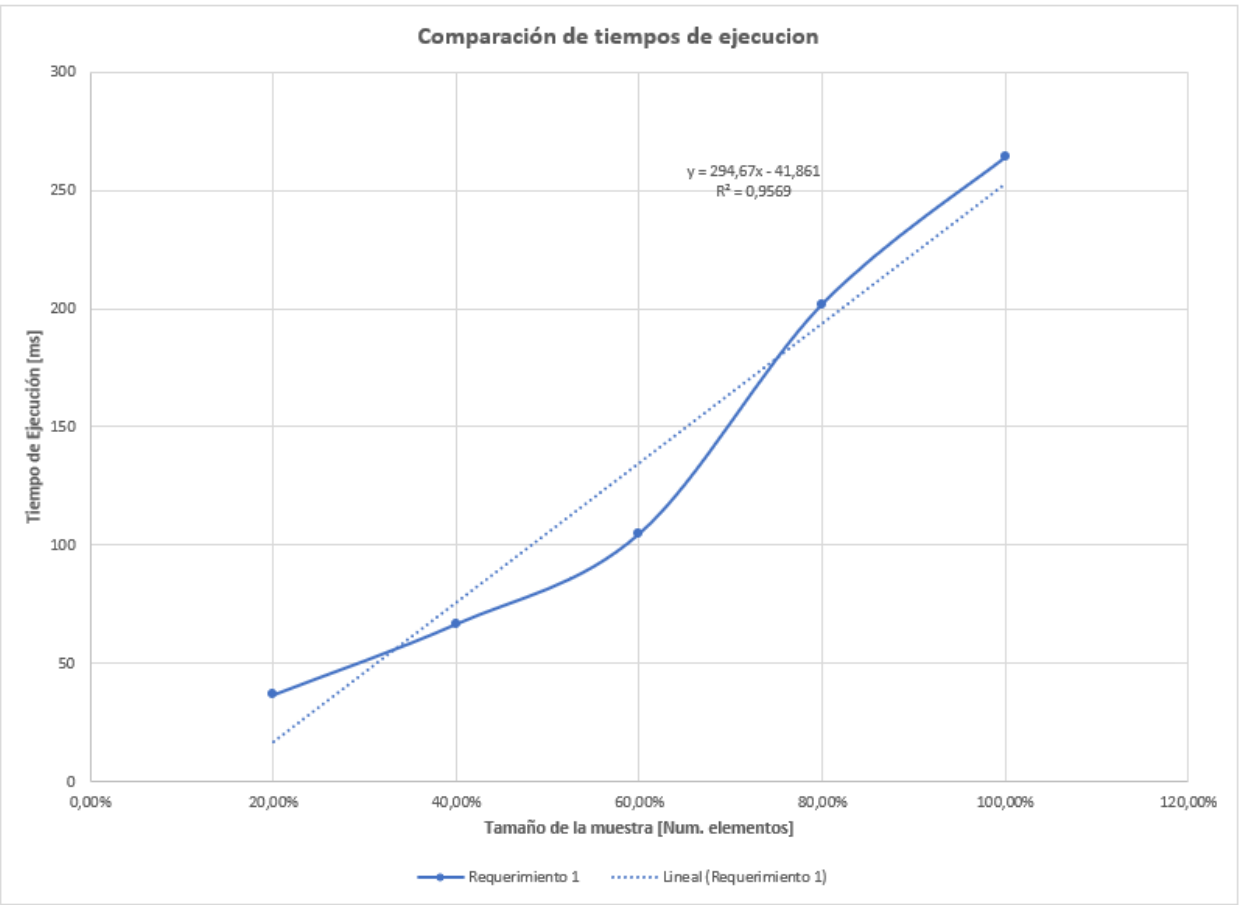
Sistema Operativo: Windows 11

Entrada	**Tiempo (s)**
20%	**36,9**
40%	**66,83**
60%	**104,99**
80%	**201,98**
100%	**263,99**

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 1
20,00%	100000,00	**36,9**
40,00%	200000,00	**66,83**
60,00%	300000,00	**104,99**
80,00%	400000,00	**201,98**
100,00%	500000,00	**263,99**

Gráficas



Análisis

La gráfica nos permite evidenciar como la función no es del todo lineal, ya que los datos no se acomodan del todo bien con la línea de tendencia. Sin embargo, tampoco se puede evidenciar un crecimiento cuadrático, ya que las curvas presentes dentro de la gráfica no se logran acomodar a una parábola. De ahí que entonces la gráfica se parezca más a una de tipo linearítmico. Esto se puede evidenciar en cómo la gráfica aumenta repentinamente entre el 60% y el 80% de los datos, similar a como lo hace una función de ese tipo. El uso de funciones de orden para los datos son las causantes a tiempos mas elevados que en las funciones del RETO 1.

Requerimiento <<3>>

```python
def req_3(catalog, dep, year0, year):
    # TODO: Modificar el requerimiento 3
    start_time = get_time()
    if year<year0:
        return "Por favor introduzca un intervalo de tiempo valido..."
    lista_info = al.new_list()
    alt_info = al.new_list()
    ordenada = al.new_list()
    survey = 0
    census = 0
    info = {'source': None,
            'commodity': None,
              'unit_measurement': None,
              'year_collection': None,
              'freq_collection': None,
              'load_time': None}

    for i in range(catalog["state_name"]["size"]):
        if catalog["state_name"]["elements"][i] == str(dep) and year0 <= catalog["year_collection"]["elements"][i] <= year:
            if catalog["load_time"]["elements"][i] not in ordenada["elements"]:
                al.add_last(ordenada, catalog["load_time"]["elements"][i])
    al.merge_sort(ordenada, False)
    for j in range(ordenada["size"] - 1):
        for i in range(catalog["load_time"]["size"] - 1):
            if catalog["load_time"]["elements"][i] == ordenada["elements"][j]:
                if catalog["state_name"]["elements"][i] == str(dep) and year0 <= catalog["year_collection"]["elements"][i] <= year:
                    info = {
                    "source": catalog["source"]["elements"][i],
                    "commodity": catalog["commodity"]["elements"][i],
                    "unit_measurement": catalog["unit_measurement"]["elements"][i],
                    "year_collection": catalog["year_collection"]["elements"][i],
                    "freq_collection": catalog["freq_collection"]["elements"][i],
                    "load_time": catalog["load_time"]["elements"][i]
                    }
                    al.add_last(lista_info, info)
                    if catalog["source"]["elements"][i] == "CENSUS":
                        census += 1
                    if catalog["source"]["elements"][i] == "SURVEY":
                        survey += 1
    tamaño = al.size(lista_info)
```

Descripción

Este requerimiento se encarga de recopilar todos los registros con su respectiva información en una lista ordenada, dentro de un margen de tiempo para un departamento especificado por el usuario, indicando también cuántos de los orígenes de los registros son de tipo "*survey*" o *"census*". Si esta lista supera los 20 elementos, se retorna solo los primeros y últimos 5. Además, si el margen de tiempo no es coherente, se le indica al usuario que intente la búsqueda con un margen de tiempo adecuado.

Entrada	Catálogo con los datos del archivo csv, el departamento deseado y el intervalo de tiempo deseado
Salidas	Lista con los datos de los registros encontrados, total de datos de tipo "*survey*", total de datos de tipo *"census*", total de datos que pasaron el filtro de búsqueda y el tiempo de ejecución del requisito en milisegundos
Implementado (Sí/No)	Si, implementado por Juan García

Análisis de complejidad

Pasos	**Complejidad**
Validación del intervalo (if year<year0)	O(1)
Asignación de variables	O(1)
Recorrido total del catálogo (for i in range …)	O(n)
Filtros de la primera búsqueda	O(n)
Ordenamiento **ordenada**	O(n log n)
Búsqueda coincidencias **ordenada** y catálogo	$O(n^2)$
Asignación info	O(1)

Añadir a **lista_info**	O(n)
Verificación del tamaño	O(1)
Posible agregado a **alt_list**	O(n)
TOTAL	***O(n^2)***

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).

Año de inicio: 2010

Año de fin: 2018

Departamento: SOUTH CAROLINA

Procesador: Ryzen 7 7730U

RAM: 16GB

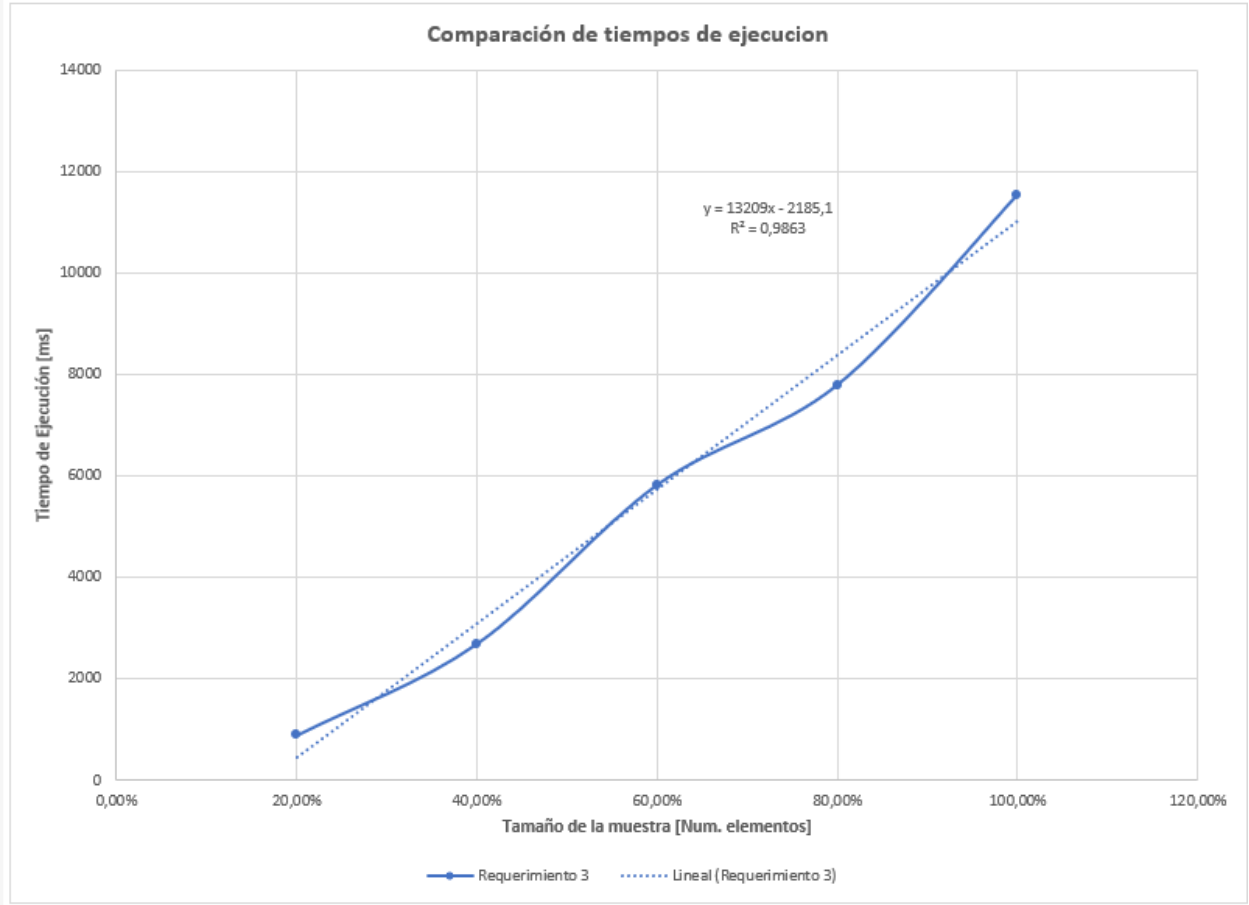
Sistema Operativo: Windows 11

Entrada	**Tiempo (s)**
20%	**878,07**
40%	**2684,37**
60%	**5816,75**
80%	**7785,89**
100%	**11536,24**

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 3
20,00%	100000,00	**878,07**
40,00%	200000,00	**2684,37**
60,00%	300000,00	**5816,75**
80,00%	400000,00	**7785,89**
100,00%	500000,00	**11536,24**

Graficas



Análisis

La presencia de curvas alrededor de toda la gráfica nos puede dar pistas de que la función se logra acomodar con una función cuadrática. Sin embargo, también es importante mencionar como los datos se logran acomodar a un ajuste lineal. Quizá debido a un reducido tamaño dentro de la lista de ordenadas, a comparación de la totalidad del catalogo, hacen que el ciclo a pesar de que tienda a ser cuadrático, no logra siempre un aumento de este tipo, pues la variabilidad de los registros que cumplen el filtro pueden afectar en la tendencia de la curva. De ahí que un buen término medio sea una función linearítmica, pues esta es mayor que una lineal, pero menor que una cuadrática, y está presente dentro de la función con el ordenamiento de tipo merge_sort. El uso de tanto ciclos anidados como funciones para orden como merge causan tiempos mas elevados para este requisite en comparacion al RETO 1.

Requerimiento <<4>>

```python
def req_4(catalog, product, ai, af):
    """
    Retorna el resultado del requerimiento 4
    """
    start_time = get_time()

    if af < ai:
        return "Ingrese un intervalo de tiempo válido"

    filtro = al.new_list()
    ans = al.new_list()
    registros = 0
    survey = 0
    census = 0

    for i in range(catalog["commodity"]["size"]):
        if (catalog["commodity"]["elements"][i] == product and ai <= catalog["year_collection"]["elements"][i] <= af):
            year = catalog["year_collection"]["elements"][i]
            if year not in filtro["elements"]:
                al.add_last(filtro, year)

    al.merge_sort(filtro, False)

    for year in filtro["elements"]:
        for i in range(catalog["commodity"]["size"]):
            if (catalog["commodity"]["elements"][i] == product and ai <= catalog["year_collection"]["elements"][i] <= af and catalog["year_collecti
                info = {
                    "source": catalog["source"]["elements"][i],
                    "year_collection": catalog["year_collection"]["elements"][i],
                    "load_time": catalog["load_time"]["elements"][i],
                    "freq_collection": catalog["freq_collection"]["elements"][i],
                    "state_name": catalog["state_name"]["elements"][i],
                    "unit_measurement": catalog["unit_measurement"]["elements"][i]
                }

                al.add_last(ans, info)
```

Descripción

Este requerimiento se encarga de buscar los registros dentro del catálogo que tienen un producto y tiempo de recolección específico dado por el usuario. Primero descarta intervalos de tiempo inválidos, luego crea las listas y contadores que serán empleados, recorre la lista por primera vez para guardar y organizar los años validos en una lista, se recorre la lista una segunda vez para cada elemento de la lista ordenada y se encuentran y guardan los registros que coinciden ya en orden, también incrementan los contadores en el proceso, se comprueba si la lista deba ser acortada y retorna el resultado.

Entrada	Catálogo, producto, año inicial, año final
Salidas	Listado de registros, cantidad de tipo census, cantidad de tipo survey, cantidad de registros, tiempo de ejecución.
Implementado (Sí/No)	Si, Tomas Aponte

Análisis de complejidad

Pasos	**Complejidad**
Se crean contadores y listas requeridas	O(1)
Se recorre la lista para años validos	O(n)
Se añaden los años validos a una lista	O(1)
Se organiza la lista	O(nlog(n))
Se recorre el catálogo para cada año valido	O(n**2)
Se añaden lo registros validos a una lista	O(1)
Incrementan los contadores necesarios	O(1)

Se acorta la lista si es necesario	O(1)
Se retornan los resultados	O(1)
TOTAL	O(n**2)

Pruebas Realizadas

Año inicio: 2010

Año final: 2020

Producto: CATTLE

Procesador: Ryzen 7 7730U

RAM: 16GB

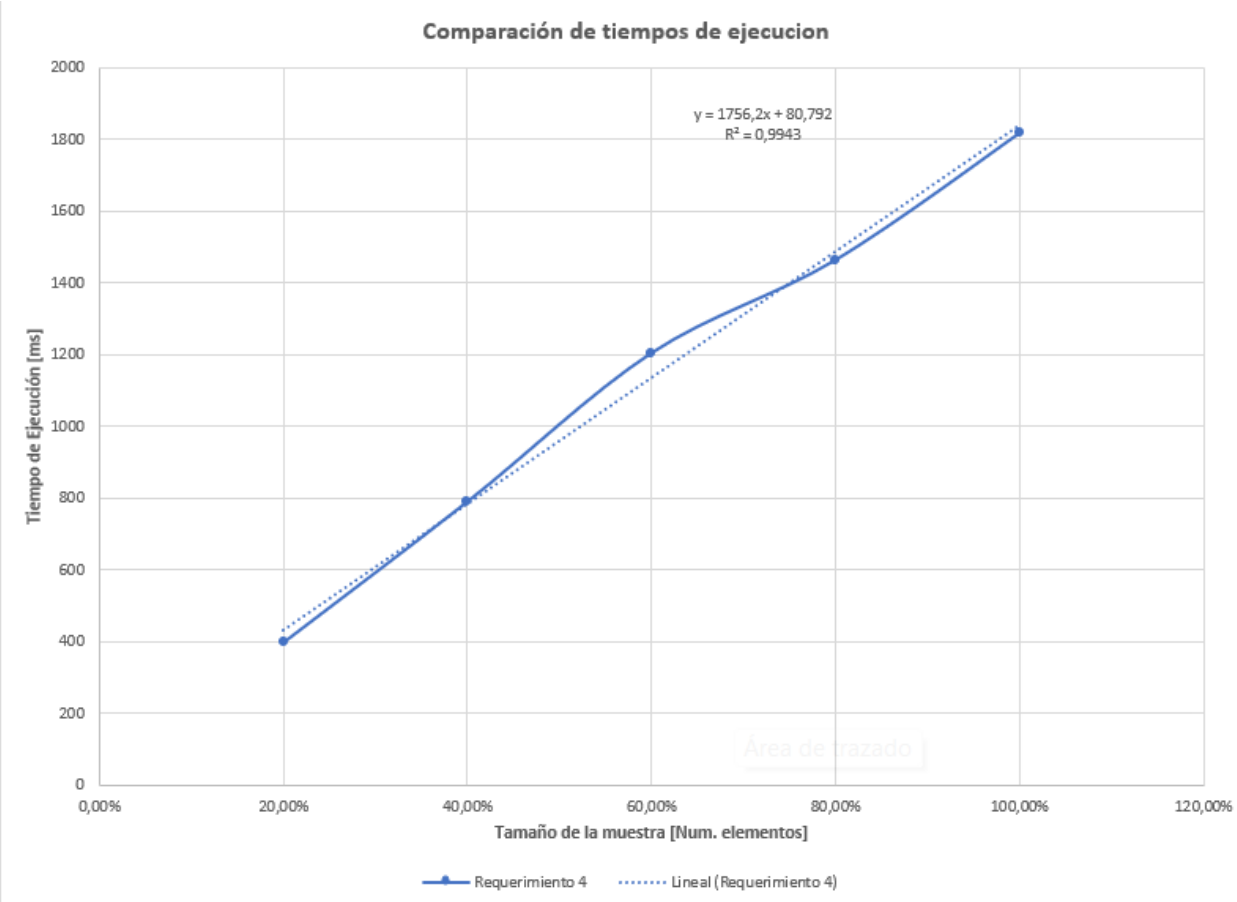
Sistema Operativo: Windows 11

Entrada	**Tiempo (s)**
20%	**397,45**
40%	**790,13**
60%	**1204,44**
80%	**1463,61**
100%	**1816,9**

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 4
20,00%	100000,00	**397,45**
40,00%	200000,00	**790,13**
60,00%	300000,00	**1204,44**
80,00%	400000,00	**1463,61**
100,00%	500000,00	**1816,9**

Graficas



Análisis

La grafica no es del todo coherente con el análisis (el cual propone una complejidad O(n**2)) ya que la relación entre el tamaño de la muestra y el tiempo de ejecución demuestra un comportamiento cercano al lineal. Esto se debe a que, aunque en la implementación del código hay un ciclo doble y un ordenamiento, al utilizar los mismos datos es posible que la estructura de estos sea favorable y no represente el peor caso posible. Sin embargo, es posible notar dos curvas leves en su estructura que muy probablemente se deben a los pasos de mayor complejidad en el algoritmo. Gracias a la implementación de ciclos anidados y funciones de orden el tiempo de ejecución se ve afectado negativamente en este requisito frente al del RETO 1

Requerimiento <<6>>

```python
def req_6(catalog, dep, ai, af):
    """
    Retorna el resultado del requerimiento 6
    """
    start_time = get_time()
    organized = al.new_list()
    ans = al.new_list()
    registros = 0
    survey = 0
    census = 0

    for i in range(catalog["load_time"]["size"]):
        load_time = catalog["load_time"]["elements"][i][:10]
        if (catalog["state_name"]["elements"][i] == dep and ai <= load_time <= af):
            if load_time not in organized["elements"]:
                al.add_last(organized, load_time)

    al.merge_sort(organized, False)

    for load_time in organized["elements"]:
        for i in range(catalog["load_time"]["size"]):
            current_time = catalog["load_time"]["elements"][i][:10]
            if (catalog["state_name"]["elements"][i] == dep and ai <= current_time <= af and current_time == load_time):
                info = {
                    "source": catalog["source"]["elements"][i],
                    "year_collection": catalog["year_collection"]["elements"][i],
                    "load_time": catalog["load_time"]["elements"][i],
                    "freq_collection": catalog["freq_collection"]["elements"][i],
                    "state_name": catalog["state_name"]["elements"][i],
                    "unit_measurement": catalog["unit_measurement"]["elements"][i],
                    "commodity": catalog["commodity"]["elements"][i]
                }
                al.add_last(ans, info)

                if catalog["source"]["elements"][i] == "SURVEY":
                    survey += 1
```

Descripción

Este requerimiento esta encargado de buscar los registros en el catálogo que cumplan con un criterio de búsqueda de tiempo de subida y departamento. Primero crea las listas y contadores necesarias para su ejecución, lego da un recorrido al catalogo para extraer los tiempos de subida validos en una lista diferente, se organiza la lista y se recorre el catalogo para cada uno de los tiempos de carga almacenados. Los registros que coincidan con estas fechas de carga y departamentos se agregan a la lista de resultado. Al añadir un registro a la lista incrementan los contadores. Se comprueba si hay que acortar la lista y se retorna el resultado.

Entrada	Catalogo, departamento, fecha inicio, fecha fin
Salidas	Listado de registros, cantidad de tipo census, cantidad de tipo survey, cantidad de registros, tiempo de ejecución.
Implementado (Sí/No)	Si, Tomas Aponte

Análisis de complejidad

Pasos	**Complejidad**
Crea listas y contadores necesarios	O(1)
Se recorre el catálogo para tiempos de carga validos	O(n)
Se añaden los tiempos a una lista	O(1)
Se organiza la lista	O(nlog(n)
Se recorre el catalogo para cada tiempo de carga	O(n**2)

Se añaden los registros validos a una lista	O(1)
Incrementan los contadores necesarios	O(1)
Se acorta la lista si es necesario	O(1)
Se retornan los resultados	O(1)
TOTAL	O(n**2)

Pruebas Realizadas

Fecha inicial: 2010-05-02

Fecha final: 2017-09-20

Departamento de interés: SOUTH CAROLINA

Procesador: Ryzen 7 7730U

RAM: 16GB

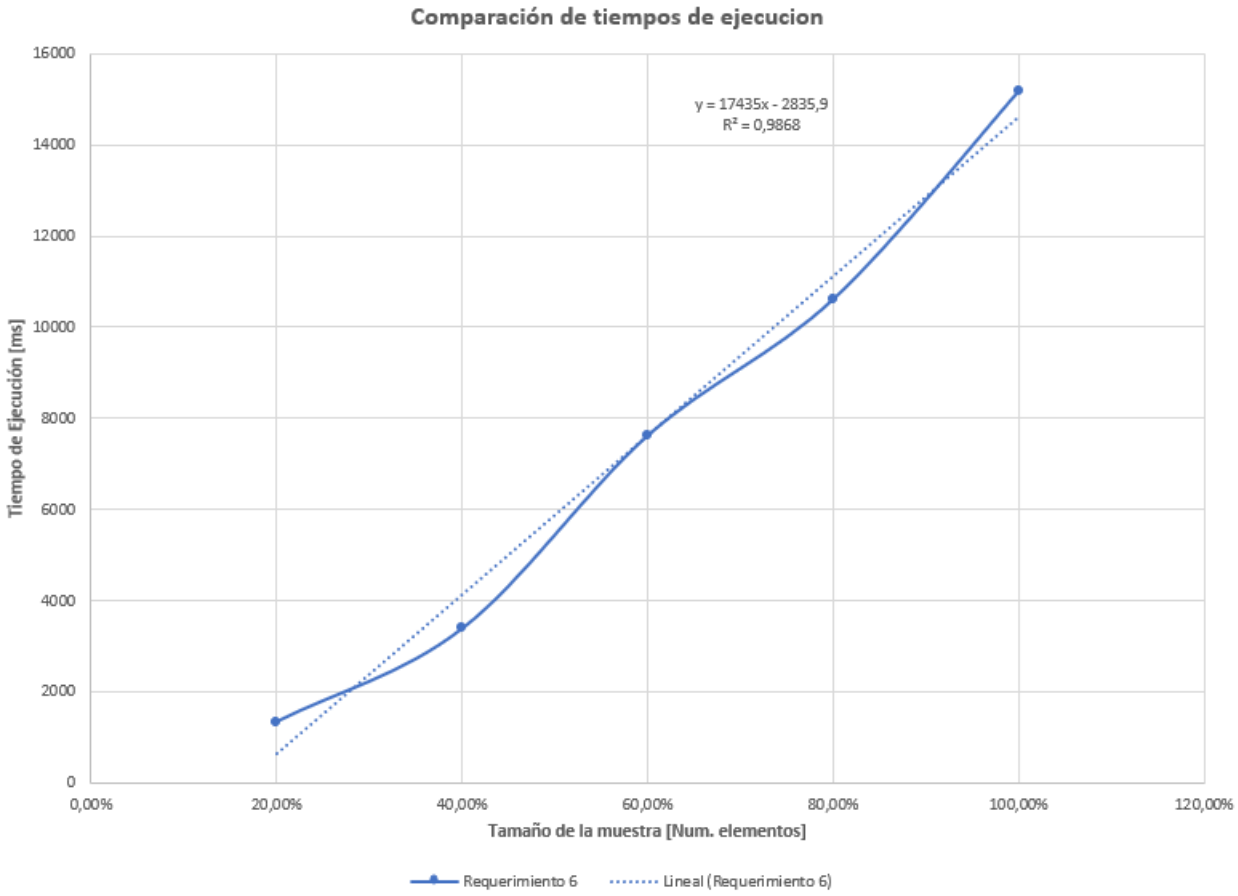
Sistema Operativo: Windows 11

Entrada	**Tiempo (s)**
20%	1344,64
40%	3386,45
60%	7617,07
80%	10606,88
100%	15169,02

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 6
20,00%	100000,00	**1344,64**
40,00%	200000,00	**3386,45**
60,00%	300000,00	**7617,07**
80,00%	400000,00	**10606,88**
100,00%	500000,00	**15169,02**

Graficas



Análisis

La relación entre el tamaño de la muestra y el tiempo de ejecución muestra un comportamiento irregular, lo cual coincide con lo planteado en el análisis de complejidad. Esto tiene sentido gracias a que dentro de la función hay un ordenamiento de los datos de complejidad O(nlog(n)) seguido de un ciclo doble de complejidad O(n**2) lo cual causa las curvaturas que se presencian en la gráfica. Al utilizar funciones de orden y ciclos anidados este requerimiento toma mas tiempo comparado al del RETO 1.

Requerimiento <<7>>

```python
def req_7(catalog, dep, year0, year, ord):
    start_time = get_time()
    if year<year0:
        return "Por favor introduzca un intervalo de tiempo valido..."
    if ord == "ASCENDENTE":
        ord = True
    else:
        ord = False
    filtro = 0
    t_filtro = 0
    invalid = 0
    ingresos = 0
    survey = 0
    census = 0
    t_filtro = 0
    maxx = None
    minn = None
    lista_temp = al.new_list()
    ordenados = al.new_list()
    ordena2 = al.new_list()
    lista_retorno = al.new_list()
    alt_list = al.new_list()
    info = {
        "año": None,
        "ingresos": None,
        "filtro": None,
        "invalidos": None,
        "survey": None,
        "census": None,
        "tipo": None
    }
    anio0 = int(year0)
    anio = int(year)
    for i in range(anio0, anio):
        for j in range(catalog["year_collection"]["size"]-1):
            if int(catalog["year_collection"]["elements"][j]) == i and catalog["state_name"]["elements"][j] == str(dep):
```

Descripción

Esta función se encarga de retornar una lista con información pertinente para cada año dentro de un rango de tiempo especificado por el usuario, para un departamento dado y ordenado de acuerdo a las preferencias del usuario. La función, para cada año que se encuentre dentro del rango, bisca dentro del catálogo ciertos parámetros para después agregarlos a un diccionario “info” que además se agrega a una lista temporal, para después ordenar los datos de acuerdo a los ingresos que se encontraron, y si se encuentran 2 o más ingresos iguales, estos se ordenan de acuerdo a la cantidad de registros encontrados para el año dado. Posterior a esto, se acomodan los diccionarios ordenadamente en la lista de retorno, y se verifica el tamaño de la lista. Si esta supera los 15 elementos, se retornan los primeros y últimos 5.

Entrada	Catálogo con los datos del archivo csv, el departamento deseado, el intervalo de tiempo deseado y el tipo de ordenamiento de los datos
Salidas	Totalidad de registros que cumplieron el filtro, lista con la información ordenada por año y tiempo de ejecución.
Implementado (Sí/No)	Si, implementado por Juan García

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Comparaciones preliminares	O(1)
Asignación variables	O(1)
Recorrido total del catálogo por los años del parámetro	$O(n^2)$

Asignación **info**	O(1)
Añadir **info** a **lista_temp**	O(1)
Recorrido de **lista_temp** para agregar elementos a ordenar a lista **ordenados**	O(n)
Ordenamiento de **ordenados**	O(n log n)
Determinación **maxx** y **minn**	O(n)
Asignar **maxx** y **minn** a sus respectivos **info**	O(n)
Determinar si existen duplicados en **ordenados**	O(n)
Posible agregado de datos adicionales a **ordena2**	O(n)
Posible ordenamiento de **ordena2**	O(n log n)
Ciclos para añadir **info** ordenados en **lista_retorno**	$O(n^2)$
Verificación tamaño **lista_retorno**	O(1)
Posible agregado a **alt_retorno**	O(n)
TOTAL	***O(n^2)***

Pruebas Realizadas

Año de inicio: 2007

Año de fin: 2015

Departamento: SOUTH CAROLINA

Orden: ASCENDENTE

Procesador: Ryzen 7 7730U

RAM: 16GB

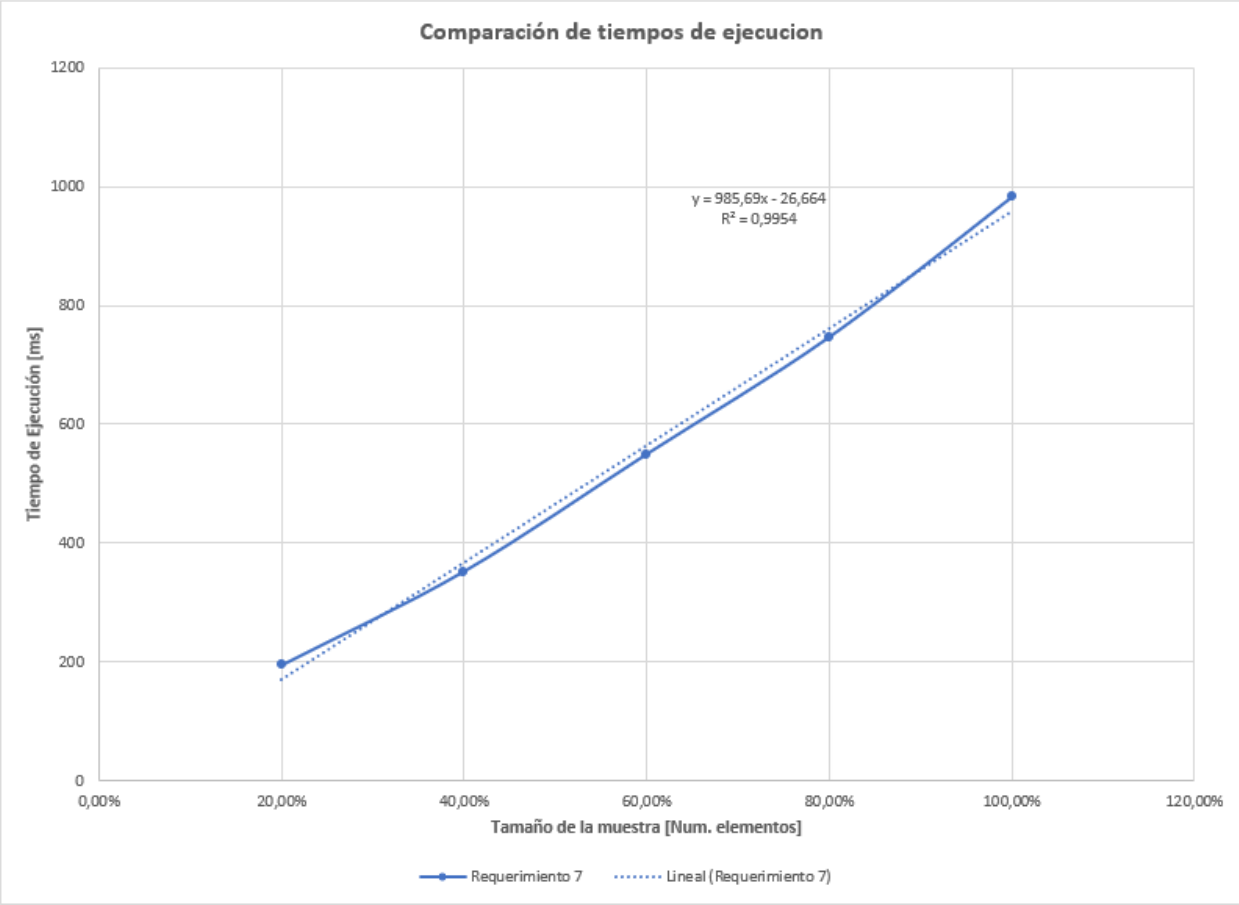
Sistema Operativo: Windows 11

Entrada	**Tiempo (s)**
20%	**193,77**
40%	**351,88**
60%	**549,46**
80%	**746,48**
100%	**982,16**

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 7
20,00%	100000,00	**193,77**
40,00%	200000,00	**351,88**
60,00%	300000,00	**549,46**
80,00%	400000,00	**746,48**
100,00%	500000,00	**982,16**

Gráficas



Análisis

Para esta función es innegable como su comportamiento es cuadrático, ya que toda la gráfica está permeada por una curva que tiende a aumentar el tiempo de ejecución. La evidencia más clara es como la gráfica cae por debajo del ajuste lineal entre el 40% y el 80% de los datos, pero ya para la totalidad de los mismos, la gráfica sobrepasa el ajuste lineal y se puede esperar que siga con este crecimiento donde existan más datos en el catálogo. Lo anterior es coherente con lo esperado habiendo hecho el análisis de la complejidad temporal, pues este quedó como $O(n^2)$. No teniamos una comparacion clara con el requerimiento 7 del RETO 1 ya que su implementacion no fue exitosa. Sin embargo, dado a que ningun requerimiento del RETO 1 se asemejaba al ordenamiento O(n**2) podriamos decir que si hubo un incremento se debe al uso de funciones para orden de datos y ciclos anidados.