ANÁLISIS DEL RETO

Iker Barbosa, 202424135, i.barbosac@uniandes.edu.co

Camilo Castro, 202412178, email 2

Melisa Molina, 202312232, email 3

Requerimiento 1

Descripción

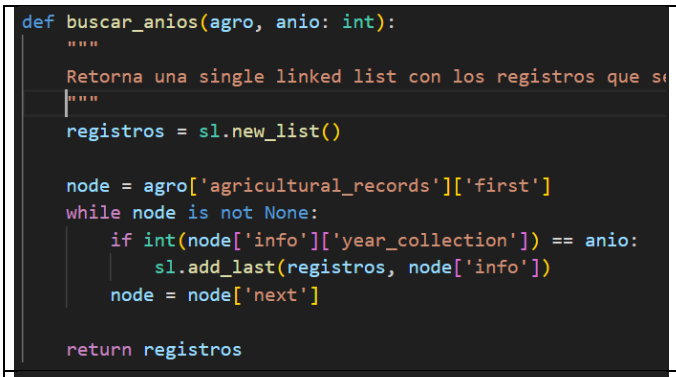
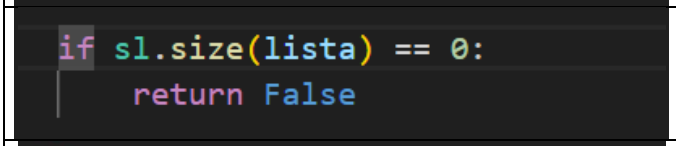
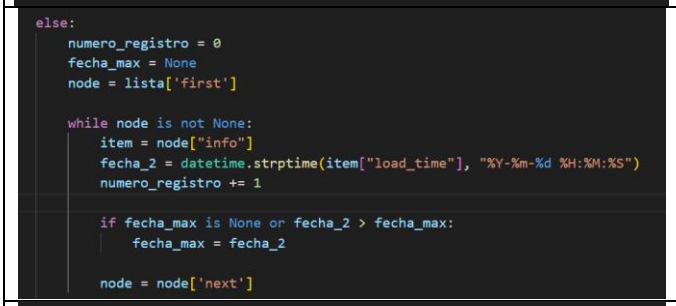
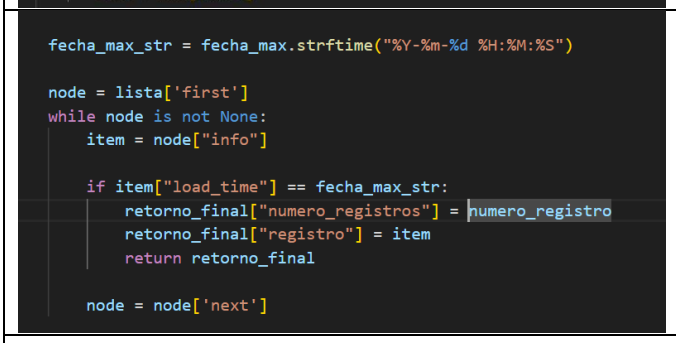
El punto del requerimiento es identificar el último registro recopilado según un año de interés. Por este motivo, usamos la librería datetime para poder comparar las fechas de la columna load_time. Primero, revisamos que la lista no esté vacía, si lo está, retorna None. En caso de que no esté vacía, recorremos la lista para hallar el último registro añadido por medio de comparaciones y guardarlo en una variable. Por último, recorremos la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. Al encontrar el primero forzamos la salida del bucle con return().

Entrada	`agro, year:str`
Salidas	`retorno_final = {"numero_registros": 0, "registro": {}}`
Implementado (Sí/No)	Implementado, Iker Barbosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

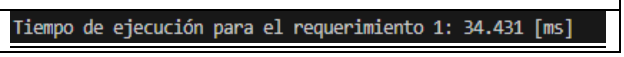
Pasos	**Complejidad**
`def req_1(agro, year: str):` `  """` `  Retorna el resultado del requerimiento 1` `  """` `  try:` `  anio = int(year)` `  except ValueError:` `  return None` `  lista = buscar_anios(agro, anio)` `  retorno_final = {"numero_registros": 0, "registro": {}}`	O(N), la función buscar_anios(), que se guarda en la variable lista, se usa para filtrar solamente los registros por el año que entra por parámetro. Para filtrar se debe recorrer toda la lista. O(1), se declara el diccionario que se va a retornar el diccionario al final.

	
	O(1), consultar el tamaño de la lista filtrada.
	O(n), recorrer la lista otra vez para hallar la fecha mas reciente.
	O(n), recorre la lista para encontrar el registro con la fecha máxima y lo añade al diccionario que retorna la función.
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el año 2024.

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (s)**
100000 datos	

200000 datos	Tiempo de ejecución para el requerimiento 1: 58.343 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 1: 91.552 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 1: 120.980 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 1: 136.652 [ms]

Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 1: 47.531 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 1: 68.652 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 1: 128.131 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 1: 144.968 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 1: 187.950 [ms]

Tablas de datos

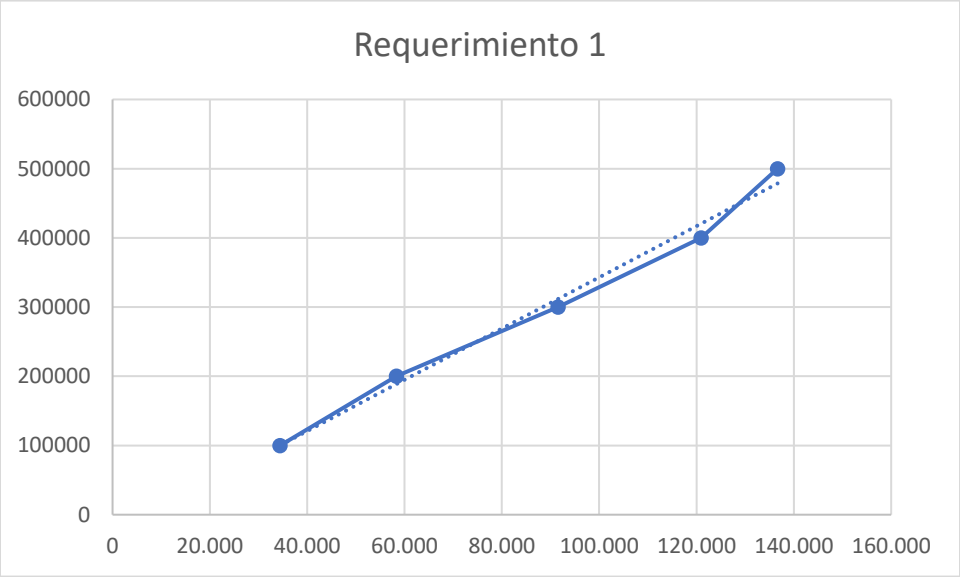
Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
100000 datos	Numero total de registros: 563 Último registro encontrado: 2024 2024-11-25 16:00:00 SURVEY WEEKLY FLORIDA CATTLE PCT	34.431
200000 datos	Numero total de registros: 1148 Último registro encontrado: 2024 2024-11-25 16:00:00 SURVEY WEEKLY FLORIDA CATTLE P	58.343
300000 datos	Numero total de registros: 1699 Último registro encontrado: 2024 2024-11-25 16:00:00 SURVEY WEEKLY FLORIDA CATTLE PCT FA	91.552
400000 datos	Numero total de registros: 2269 Último registro encontrado: 2024 2024-11-25 16:00:00 SURVEY WEEKLY FLORIDA CATTLE PCT FAI	120.980
500000 datos	Numero total de registros: 2864 Último registro encontrado: 2024 2024-11-25 16:00:00 SURVEY WEEKLY FLORIDA CATTLE F	136.652

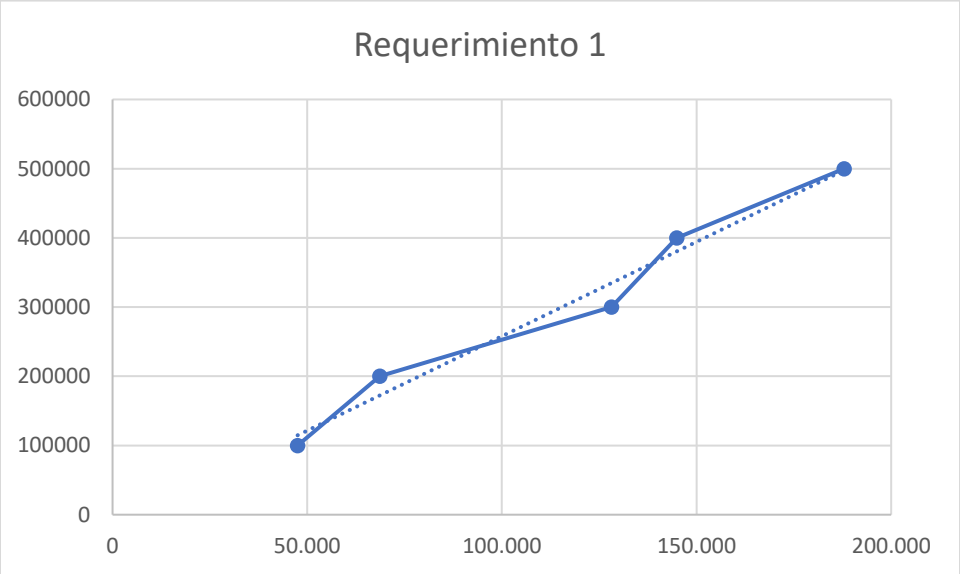
Graficas

Las gráficas con la representación de las pruebas realizadas.

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10



Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11



Análisis

Al tratarse de recorridos de lista completos para realizar comparaciones, por temas de velocidad decidimos usar Single Linked List, la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar para recorrer la lista a su totalidad, ya que debemos comparar todos los elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

Requerimiento 2

Descripción

El punto del requerimiento es identificar el último registro cargado dado un departamento de interés. Similar al requerimiento anterior, usamos la librería datetime para poder comparar las fechas de la columna load_time y así identificar el último registro cargado. En este caso, simplemente recorremos la lista y con un condicional verificamos los datos correspondan al departamento que recibimos por parámetro y guardamos las fechas en variables para compararlas. Por último, recorremos la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. Al encontrar el primero forzamos la salida del bucle con return().

Entrada	`(agro, departamento:str):`

Salidas	`retorno_final = {"numero_registros": 0, "registro": {}}`
Implementado (Sí/No)	Implementado, Iker Barbosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
`fecha_1 = datetime.strptime("2011-01-01 00:00:00", "%Y-%m-%d %H:%M:%S")` `retorno_final = {"numero_registros": 0, "registro": {}}` `node = agro["agricultural_records"]["first"]` `numero_registro = 0` `while node is not None:` `  item = node["info"]` `  if item["state_name"] == departamento:` `  fecha_2 = datetime.strptime(item["load_time"], "%Y-%m-%d %H:%M:%S")` `  if fecha_2 > fecha_1:` `  fecha_1 = fecha_2` `  numero_registro += 1` `  node = node["next"]` `fecha_min = fecha_1.strftime("%Y-%m-%d %H:%M:%S")`	O(n), recorrer la lista para hallar el último registro añadido dependiendo del departamento pasado por parámetro.
`node = agro['agricultural_records']['first']` `while node is not None:` `  item = node["info"]` `  if item["load_time"] == fecha_min and item["state_name"] == departamento:` `  retorno_final["numero_registros"] = numero_registro` `  retorno_final["registro"] = item` `  return retorno_final` `  node = node['next']`	O(n), recorrer la lista para hallar el último registro añadido en el departamento pasado por parámetro.
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el estado de CALIFORNIA.

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (s)**
100000 datos	`Tiempo de ejecución para el requerimiento 2: 29.489 [ms]`
200000 datos	`Tiempo de ejecución para el requerimiento 2: 86.088 [ms]`
300000 datos	`Tiempo de ejecución para el requerimiento 2: 120.451 [ms]`
400000 datos	`Tiempo de ejecución para el requerimiento 2: 145.154 [ms]`
500000 datos	`Tiempo de ejecución para el requerimiento 2: 166.005 [ms]`

Procesadores	12th Gen Intel(R) Core(TM) i5-12450H
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	Tiempo (s)
100000 datos	Tiempo de ejecución para el requerimiento 2: 74.627 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 2: 119.370 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 2: 203.774 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 2: 254.043 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 2: 265.767 [ms]

Tablas de datos

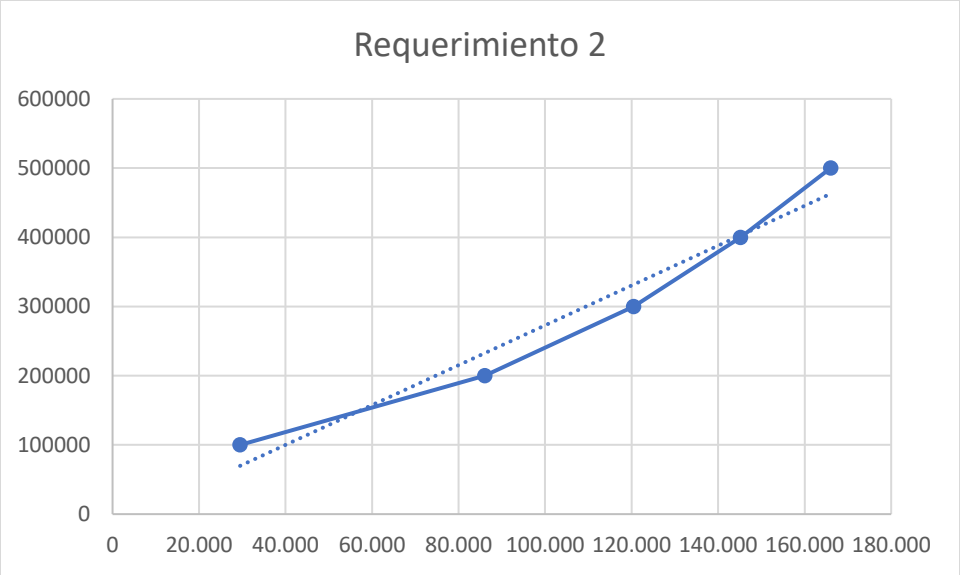
Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
100000 datos	Numero total de registros: 2155 Último registro encontrado: 2022 2024-11-07 12:00:00 CENSUS ANNUAL CALIFORNIA CATTLE OPE	29.489
200000 datos	Numero total de registros: 4331 Último registro encontrado: 2024 2024-11-21 15:00:00 SURVEY MONTHLY CALIFORNIA SHEEP LB,	86.088
300000 datos	Numero total de registros: 6586 Último registro encontrado: 2024 2024-11-21 15:00:00 SURVEY MONTHLY CALIFORNIA SHEEP LE	120.451
400000 datos	Numero total de registros: 8764 Último registro encontrado: 2024 2024-11-21 15:00:00 SURVEY MONTHLY CALIFORNIA SHEEP LB	145.154
500000 datos	Numero total de registros: 10984 Último registro encontrado: 2024 2024-11-21 15:00:00 SURVEY MONTHLY CALIFORNIA SHEEP LB	166.005

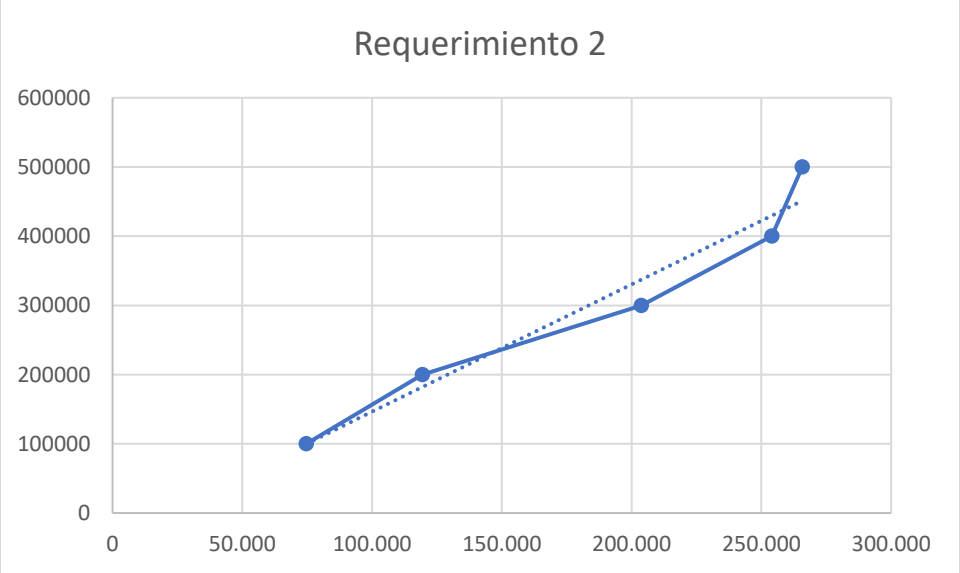
Graficas

Las gráficas con la representación de las pruebas realizadas.

Procesadores	AMD Ryzen 7 5700G
Memoria RAM	22 GB
Sistema Operativo	Windows 10



Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11



Análisis

Al tratarse de recorridos de lista completos para realizar comparaciones, por temas de velocidad decidimos usar Single Linked List, la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar para recorrer la lista a su totalidad, ya que debemos comparar todos los elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Al inicio tiene una pendiente más inclinada, pero a medida que tenemos más datos, adopta su forma lineal y una pendiente más fija. Por lo que cumple con el comportamiento lineal esperado.

Requerimiento 3

Descripción

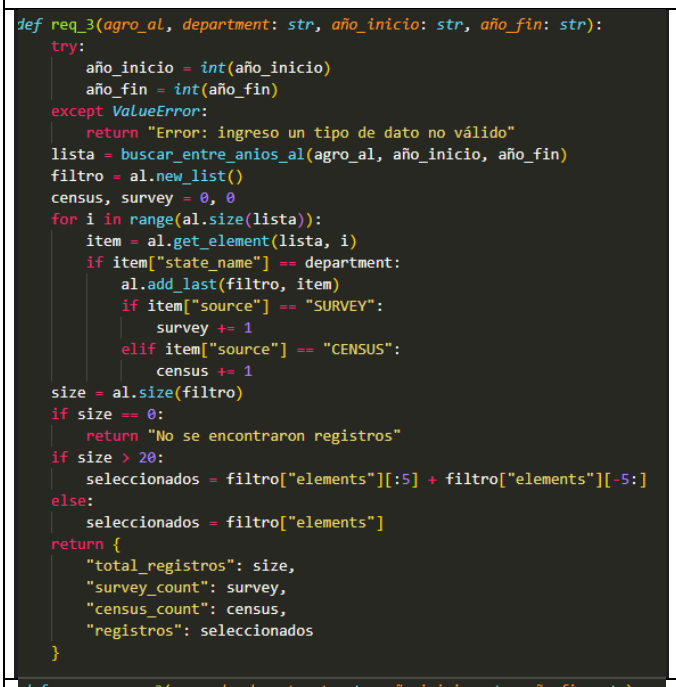
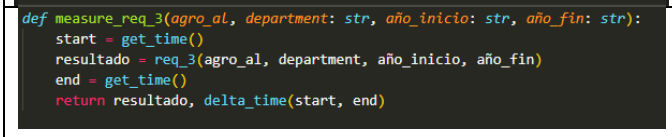
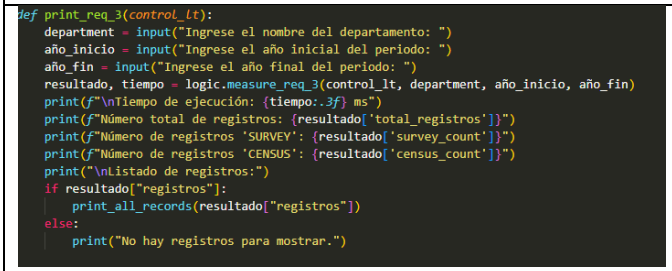
El punto del requerimiento es consultar los registros recopilados para un departamento de interés dado un rango de años de recopilación. Por este motivo, se usa una función auxiliar para obtener un **ArrayList** filtrado por años de carga. Desde ahí, se recorre el **ArrayList** más pequeño y se filtra por el nombre del departamento. En caso de que el número de registros sea menor o igual a 20, se devuelve la lista completa; en caso contrario, se devuelven únicamente los primeros 5 y los últimos 5 registros. Además, se lleva un conteo de registros según su fuente de recopilación (SURVEY o CENSUS).

Entrada	`agro_al, department: str, año_inicio: str, año_fin: str`
Salidas	A) `return { "total_registros": size, "survey_count": survey, "census_count": census, "registros": seleccionados }` B) `return "Error: ingreso un tipo de dato no válido"`
	Implementado, Camilo Castro

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
	Esta función tiene una complejidad de O(N), donde N es la cantidad total de registros en agro_al["agricultural_records"]. La razón es que la función recorre toda la lista de registros una sola vez, verificando si cada uno se encuentra dentro del rango de años especificado y agregándolo a la lista de resultados en caso afirmativo. Como cada una de estas operaciones es de tiempo constante O(1) por cada elemento, la

	complejidad general queda determinada por el tamaño de la lista original, es decir, **O(N)**.
 ```def req_3(agro_al, department: str, año_inicio: str, año_fin: str): try: año_inicio = int(año_inicio) año_fin = int(año_fin) except ValueError: return "Error: ingreso un tipo de dato no válido" lista = buscar_entre_anios_al(agro_al, año_inicio, año_fin) filtro = al.new_list() census, survey = 0, 0 for i in range(al.size(lista)): item = al.get_element(lista, i) if item["state_name"] == department: al.add_last(filtro, item) if item["source"] == "SURVEY": survey += 1 elif item["source"] == "CENSUS": census += 1 size = al.size(filtro) if size == 0: return "No se encontraron registros" if size > 20: seleccionados = filtro["elements"][:5] + filtro["elements"][-5:] else: seleccionados = filtro["elements"] return { "total_registros": size, "survey_count": survey, "census_count": census, "registros": seleccionados }```	La función req_3 tiene una complejidad de O(N + M). Primero, llama a buscar_entre_anios_al, que tiene una complejidad O(N) al recorrer todos los registros. Luego, itera sobre la lista filtrada (lista), de tamaño M, para aplicar un segundo filtro por departamento y contar los tipos de fuente, lo que introduce un término O(M). Finalmente, selecciona los primeros y últimos 5 registros si hay más de 20, lo que es una operación O(1). Como el comportamiento dominante es la iteración sobre los registros originales y la lista filtrada, la complejidad total es O(N + M).. Dado que es formato O(), entonces es **O(N)**
 ```def measure_req_3(agro_al, department: str, año_inicio: str, año_fin: str): start = get_time() resultado = req_3(agro_al, department, año_inicio, año_fin) end = get_time() return resultado, delta_time(start, end)```	La función measure_req_3 tiene una complejidad de O(N + M), ya que simplemente mide el tiempo de ejecución de la función req_3 sin realizar cálculos adicionales significativos. Como req_3 es la que ejecuta la lógica de filtrado y selección de registros, su complejidad domina la función measure_req_3. Por lo tanto, la complejidad de measure_req_3 es equivalente a la de req_3, es decir, O(N + M). Dado que es formato O(), entonces es **O(N)**
 ```def print_req_3(control_lt): department = input("Ingrese el nombre del departamento: ") año_inicio = input("Ingrese el año inicial del periodo: ") año_fin = input("Ingrese el año final del periodo: ") resultado, tiempo = logic.measure_req_3(control_lt, department, año_inicio, año_fin) print(f"\nTiempo de ejecución: {tiempo:.3f} ms") print(f"Número total de registros: {resultado['total_registros']}") print(f"Número de registros 'SURVEY': {resultado['survey_count']}") print(f"Número de registros 'CENSUS': {resultado['census_count']}") print("\nListado de registros:") if resultado["registros"]: print_all_records(resultado["registros"]) else: print("No hay registros para mostrar.")```	La función print_req_3 tiene una complejidad de O(N + M), ya que su ejecución está dominada por la llamada a measure_req_3, que a su vez ejecuta req_3. Además, imprime los registros seleccionados en un bucle de O(1), ya que el número máximo de registros mostrados es constante (hasta 10 registros). Como la función req_3 tiene una complejidad O(N + M) y print_req_3 no introduce iteraciones adicionales significativas, su complejidad total también es O(N + M).y retornamos el Array List. Dado que es formato O(), entonces es **O(N)**
TOTAL	***O(N)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el tipo de producto US TOTAL, entre 2000 y 2001.

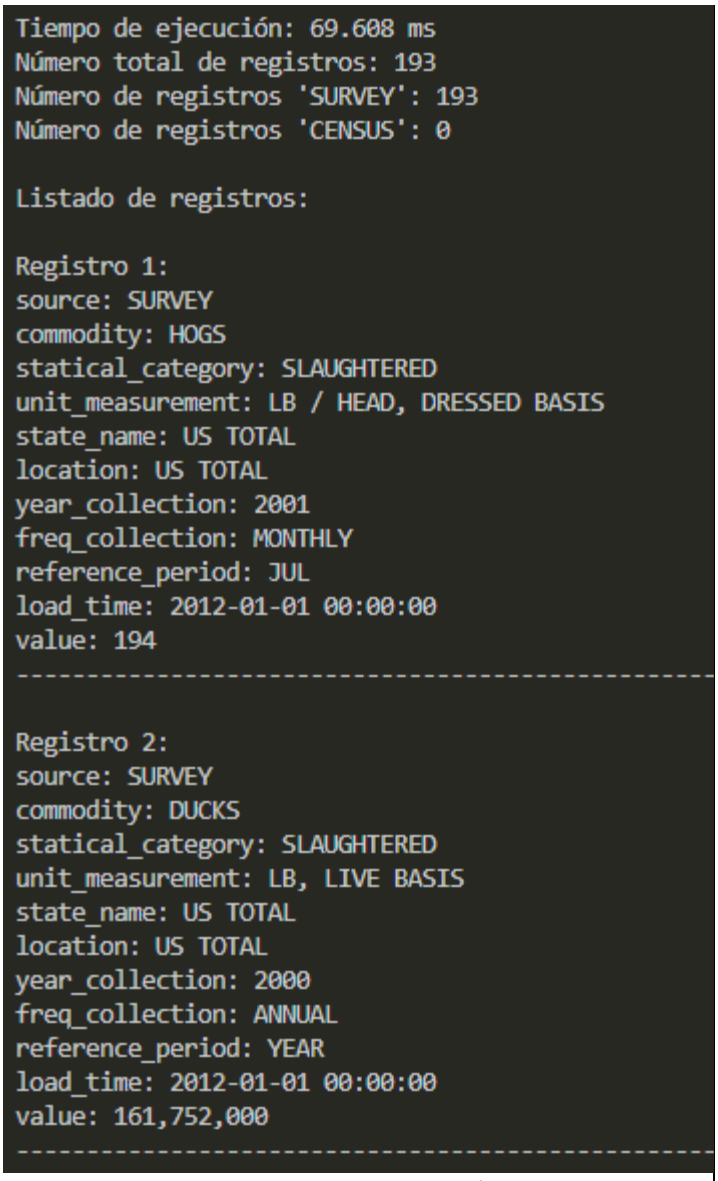
Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	8,00 GB
Sistema Operativo	Windows 10 Home Single Language

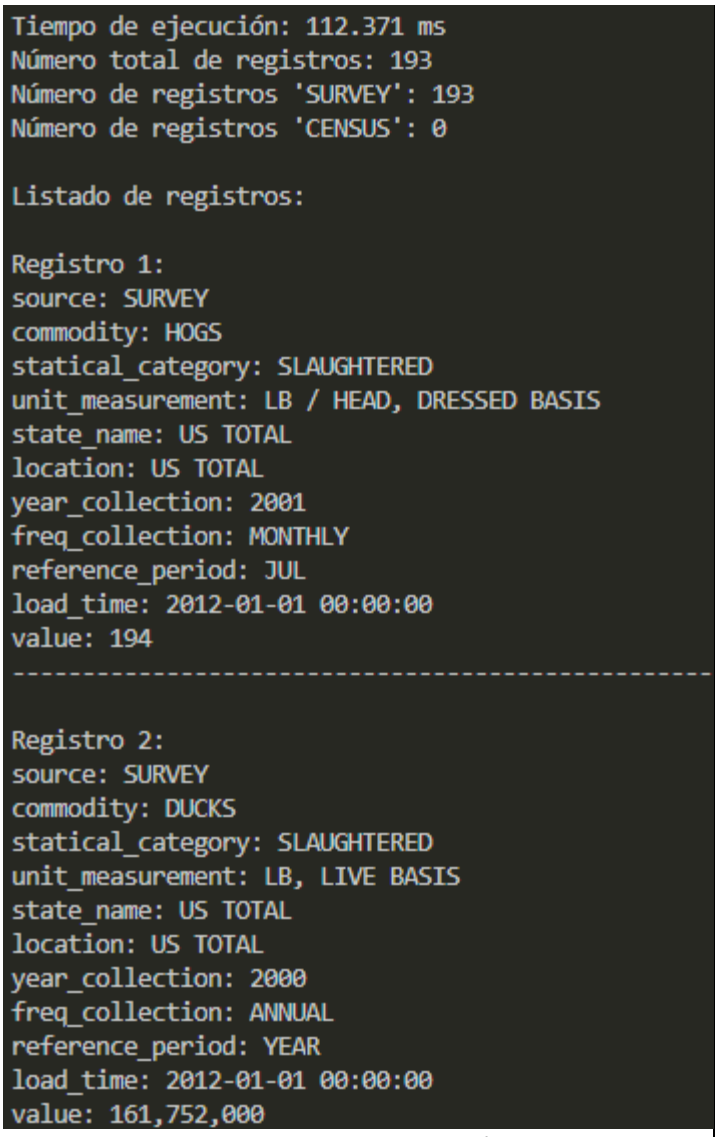
Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución: 69.608 ms
200000 datos	Tiempo de ejecución: 112.371 ms
300000 datos	Tiempo de ejecución: 181.784 ms
400000 datos	Tiempo de ejecución: 237.469 ms
500000 datos	Tiempo de ejecución: 314.861 ms

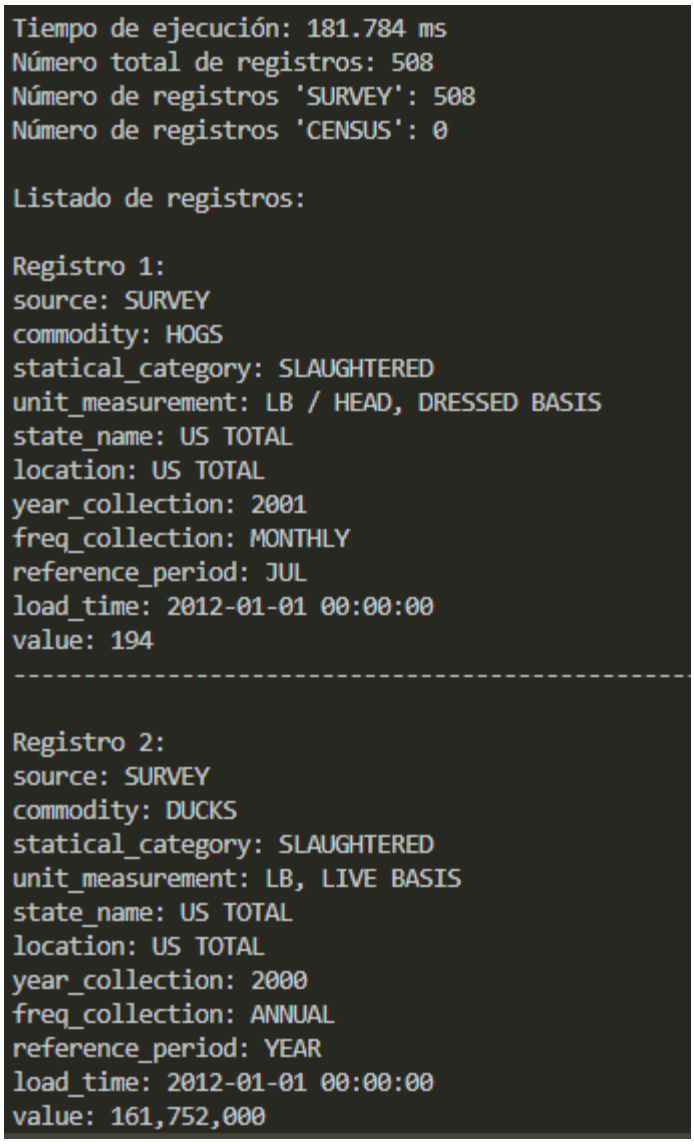
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

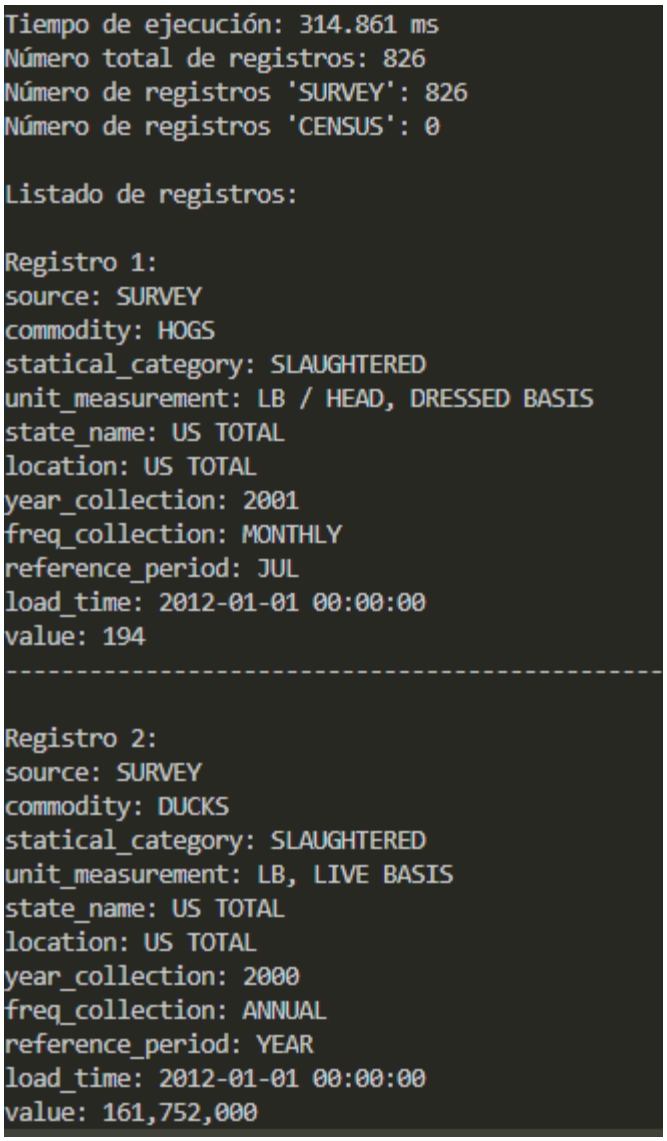
Muestra	**Salida**	**Tiempo (ms)**

100000 datos	 Y asi con 8 registros más	69.608 ms

200000 datos	 Y asi con 8 resgistros más	112.371 ms

300000 datos	 Y asi con 8 resgistros más	181.784 ms

400000 datos	Tiempo de ejecución: 237.469 ms Número total de registros: 664 Número de registros 'SURVEY': 664 Número de registros 'CENSUS': 0 Listado de registros: Registro 1: source: SURVEY commodity: HOGS statical_category: SLAUGHTERED unit_measurement: LB / HEAD, DRESSED BASIS state_name: US TOTAL location: US TOTAL year_collection: 2001 freq_collection: MONTHLY reference_period: JUL load_time: 2012-01-01 00:00:00 value: 194 ---------- Registro 2: source: SURVEY commodity: DUCKS statical_category: SLAUGHTERED unit_measurement: LB, LIVE BASIS state_name: US TOTAL location: US TOTAL year_collection: 2000 freq_collection: ANNUAL reference_period: YEAR load_time: 2012-01-01 00:00:00 value: 161,752,000	237.469 ms

500000 datos	 Y asi con 8 registros más	314.861 ms

Graficas

Las gráfica con la representación de las pruebas realizadas.

Análisis

Para el procesamiento de los datos, decidimos utilizar un Array List, ya que permite acceder rápidamente a los elementos mediante get_element(), lo que tiene una complejidad de O(1). La implementación del requerimiento tiene un orden de complejidad O(n) en su mayoría, debido a que debemos recorrer el Array List una vez para filtrar los registros entre los años indicados, y luego recorrer nuevamente la lista filtrada para seleccionar aquellos que cumplen con el criterio del departamento.

Las operaciones de acceso a elementos (get_element()), inserción al final (add_last()) y conteo (size()) son todas O(1), lo que hace que las operaciones dentro del ciclo mantengan su eficiencia. Sin embargo, como el recorrido del Array List es lineal y se repite varias veces, el orden de complejidad total del requerimiento sigue siendo O(n).

Este comportamiento se puede evidenciar experimentalmente en una gráfica de tiempo de ejecución, donde se observa que el crecimiento es lineal con respecto a la cantidad de registros analizados.

Requerimiento 4

Descripción

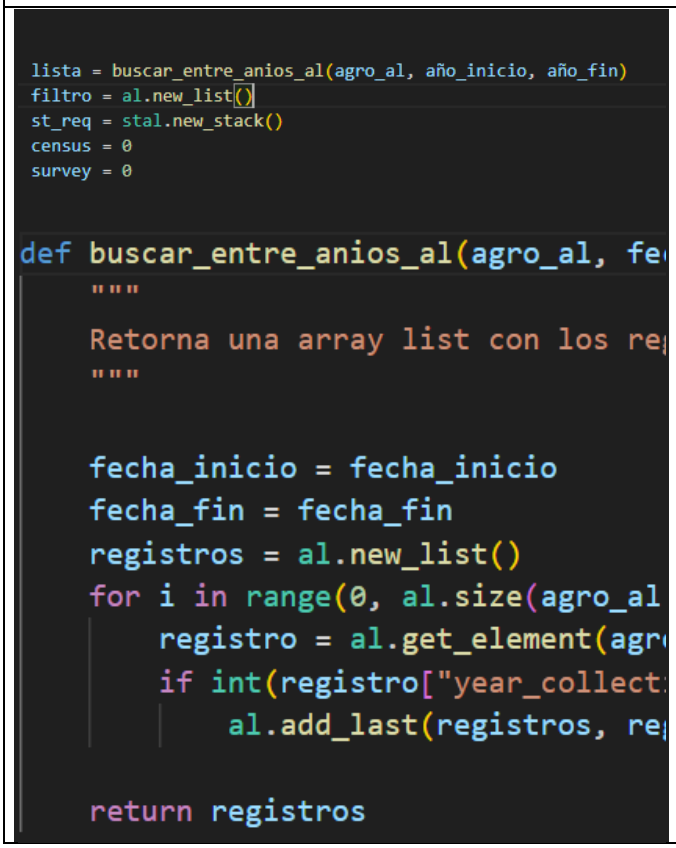
El punto del requerimiento es consultar los registros recopilados para un tipo de producto de interés dado un rango de años de recopilación dado. Por este motivo, use una función auxiliar para que me dé

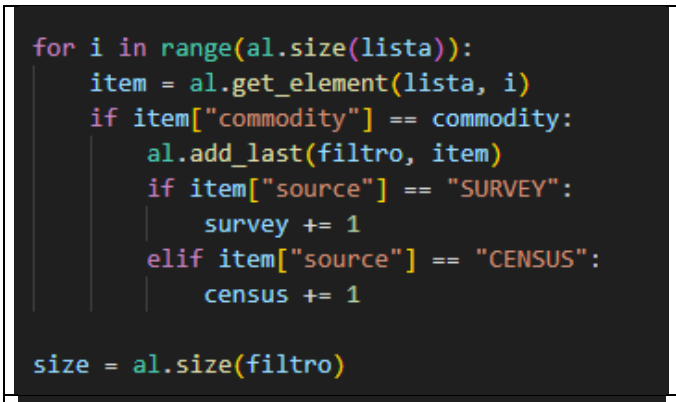
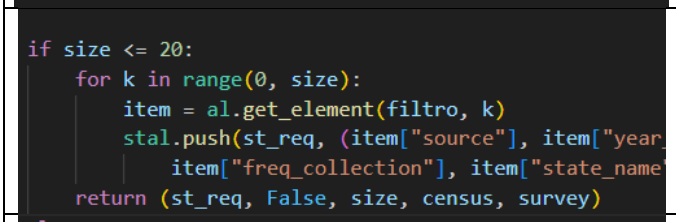
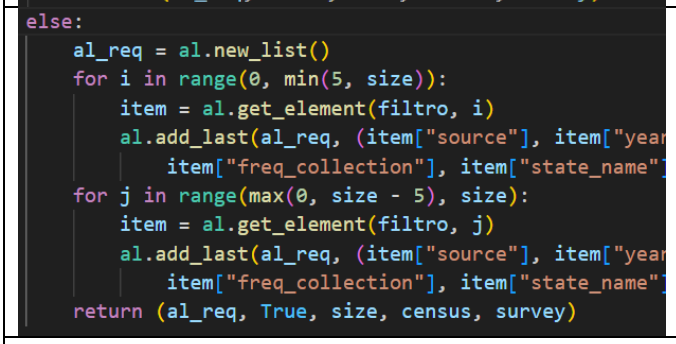
un Array List filtrado entre años de carga. Desde acá es más fácil recorrer de nuevo un Array List más pequeño y filtrarlo por tipo de producto. En caso de que el número de registros sea menor o igual a 20 devuelvo una Pila, en caso contrario, devuelvo un Array List con acceso a elementos.

Entrada	agro_al, commodity:str, año_inicio:str, año_fin:str)
Salidas	C) Tupla con pila, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. D) Tupla con Array List, boolean, size, numero de registros tipo celsus, numero de registros tipo survey.
	Implementado, Iker Barbosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
	O(n), la función auxiliar buscar_entre_anios_al() es una función que recorre todo el Array List para filtrar entre los años recibidos por parametro.

	O(n), recorrer la lista filtrada entre años para hallar los registros que coincidan con el tipo de producto que se recibió por parámetro.
	O(n), si el tamaño de lista es menor o igual a 20, recorremos la lista para añadir sus elementos a una pila y retornarla.
	O(n), en caso de que el tamaño de la lista sea mayor a 20. Creamos una nueva Array List con new_list(), e iteramos la lista para añadir los primeros 5 elementos y los últimos 5 con get.element() y retornamos el Array List.
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el tipo de producto EGGS, entre 1980 y 2024.

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 4: 72.495 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 4: 142.293 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 4: 249.642 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 4: 293.690 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 4: 359.728 [ms]

Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 4: 107.659 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 4: 156.351 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 4: 239.947 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 4: 335.712 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 4: 465.873 [ms]

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

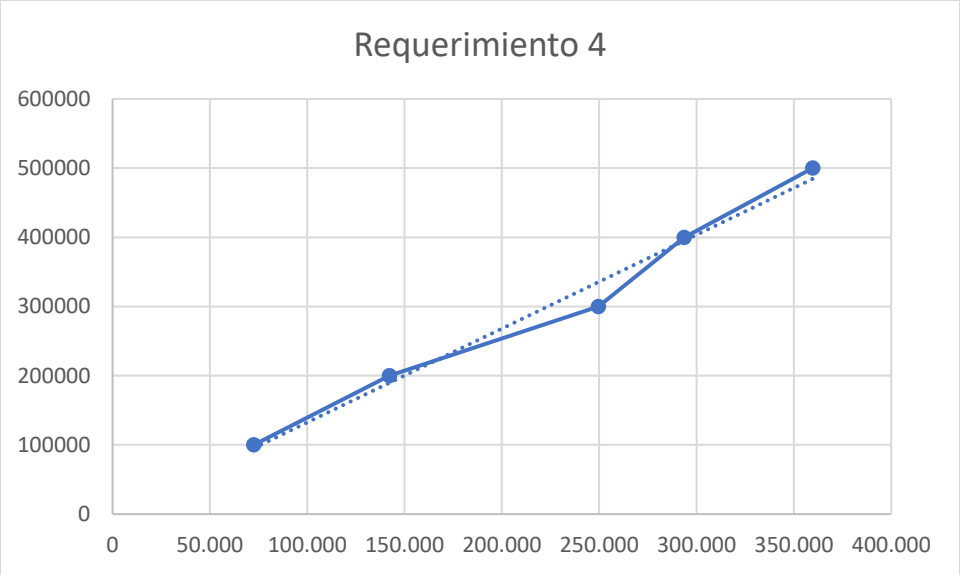
Muestra	**Salida**	**Tiempo (ms)**
100000 datos	('SURVEY', '2005', '2014-03-24', 'POINT IN TIME', 'US TOTAL', 'LB', 'EGG ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'IOWA', 'EGGS', 'EGGS') ('SURVEY', '2020', '2024-03-05', 'MONTHLY', 'MINNESOTA', 'DOZEN', 'EGGS' ('SURVEY', '2014', '2019-06-25', 'MONTHLY', 'ILLINOIS', 'EGGS', 'EGGS') ('SURVEY', '2012', '2014-09-22', 'MONTHLY', 'NEW YORK', 'EGGS', 'EGGS') ('SURVEY', '2018', '2024-03-05', 'ANNUAL', 'OHIO', 'DOZEN', 'EGGS') ('SURVEY', '1991', '2012-01-01', 'ANNUAL', 'ALABAMA', 'EGGS', 'EGGS') ('CENSUS', '2015', '2015-01-31', 'ANNUAL', 'WASHINGTON', '$', 'EGGS') ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'VIRGINIA', 'EGGS', 'EGGS') ('SURVEY', '2021', '2024-03-05', 'MONTHLY', 'OKLAHOMA', 'DOZEN', 'EGGS') Total de registros: 582 Total de registros con tipo de fuente/origen “CENSUS”: 86 Total de registros con tipo de fuente/origen “SURVEY”: 496	72.495
200000 datos	('SURVEY', '2005', '2014-03-24', 'POINT IN TIME', 'US TOTAL', ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'IOWA', 'EGGS', ' ('SURVEY', '2020', '2024-03-05', 'MONTHLY', 'MINNESOTA', 'DOZ ('SURVEY', '2014', '2019-06-25', 'MONTHLY', 'ILLINOIS', 'EGGS ('SURVEY', '2012', '2014-09-22', 'MONTHLY', 'NEW YORK', 'EGGS ('SURVEY', '2012', '2014-09-22', 'ANNUAL', 'MASSACHUSETTS', ' ('SURVEY', '1980', '2012-01-01', 'ANNUAL', 'TEXAS', 'EGGS', ' ('SURVEY', '2016', '2019-06-25', 'MONTHLY', 'MINNESOTA', 'EGG ('SURVEY', '1980', '2012-01-01', 'ANNUAL', 'TEXAS', 'EGGS', ' ('SURVEY', '2022', '2024-02-20', 'MONTHLY', 'MISSOURI', 'DOZE Total de registros: 1127 Total de registros con tipo de fuente/origen “CENSUS”: 144 Total de registros con tipo de fuente/origen “SURVEY”: 983	142.293

300000 datos	('SURVEY', '2005', '2014-03-24', 'POINT IN TIME', 'US TOTAL', ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'IOWA', 'EGGS', ' ('SURVEY', '2020', '2024-03-05', 'MONTHLY', 'MINNESOTA', 'DOZ ('SURVEY', '2014', '2019-06-25', 'MONTHLY', 'ILLINOIS', 'EGGS ('SURVEY', '2012', '2014-09-22', 'MONTHLY', 'NEW YORK', 'EGGS ('SURVEY', '2004', '2012-01-01', 'MONTHLY', 'GEORGIA', '$ / D ('SURVEY', '1982', '2012-01-01', 'ANNUAL', 'WISCONSIN', 'EGGS ('SURVEY', '2023', '2024-02-20', 'MONTHLY', 'OTHER STATES', ' ('SURVEY', '2018', '2024-03-05', 'MONTHLY', 'ARKANSAS', 'EGGS ('SURVEY', '2020', '2024-03-05', 'MONTHLY', 'MISSISSIPPI', 'D Total de registros: 1700 Total de registros con tipo de fuente/origen "CENSUS": 224 Total de registros con tipo de fuente/origen "SURVEY": 1476	249.642
400000 datos	('SURVEY', '2005', '2014-03-24', 'POINT IN TIME', 'US TOTAL', ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'IOWA', 'EGGS', ' ('SURVEY', '2020', '2024-03-05', 'MONTHLY', 'MINNESOTA', 'DOZ ('SURVEY', '2014', '2019-06-25', 'MONTHLY', 'ILLINOIS', 'EGGS ('SURVEY', '2012', '2014-09-22', 'MONTHLY', 'NEW YORK', 'EGGS ('CENSUS', '2018', '2018-02-01', 'ANNUAL', 'AMERICAN SAMOA', ('SURVEY', '2019', '2024-03-05', 'MONTHLY', 'VIRGINIA', 'DOZE ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'ILLINOIS', 'EGGS ('SURVEY', '2022', '2024-02-20', 'MONTHLY', 'US TOTAL', 'DOZE ('SURVEY', '2019', '2024-03-05', 'MONTHLY', 'COLORADO', 'EGGS Total de registros: 2307 Total de registros con tipo de fuente/origen "CENSUS": 305 Total de registros con tipo de fuente/origen "SURVEY": 2002	293.690
500000 datos	('SURVEY', '2005', '2014-03-24', 'POINT IN TIME', 'US TOTAL', ('SURVEY', '2008', '2014-09-04', 'MONTHLY', 'IOWA', 'EGGS', 'E ('SURVEY', '2020', '2024-03-05', 'MONTHLY', 'MINNESOTA', 'DOZE ('SURVEY', '2014', '2019-06-25', 'MONTHLY', 'ILLINOIS', 'EGGS' ('SURVEY', '2012', '2014-09-22', 'MONTHLY', 'NEW YORK', 'EGGS' ('SURVEY', '2017', '2019-06-25', 'MONTHLY', 'MISSISSIPPI', 'EG ('SURVEY', '2005', '2012-01-01', 'MONTHLY', 'MISSISSIPPI', '$ ('SURVEY', '2012', '2014-09-22', 'MONTHLY', 'MAINE', 'EGGS', ' ('SURVEY', '2018', '2020-02-26', 'MONTHLY', 'US TOTAL', 'LB', ('SURVEY', '2000', '2012-01-01', 'ANNUAL', 'ALABAMA', 'EGGS', Total de registros: 2900 Total de registros con tipo de fuente/origen "CENSUS": 386 Total de registros con tipo de fuente/origen "SURVEY": 2514	359.728

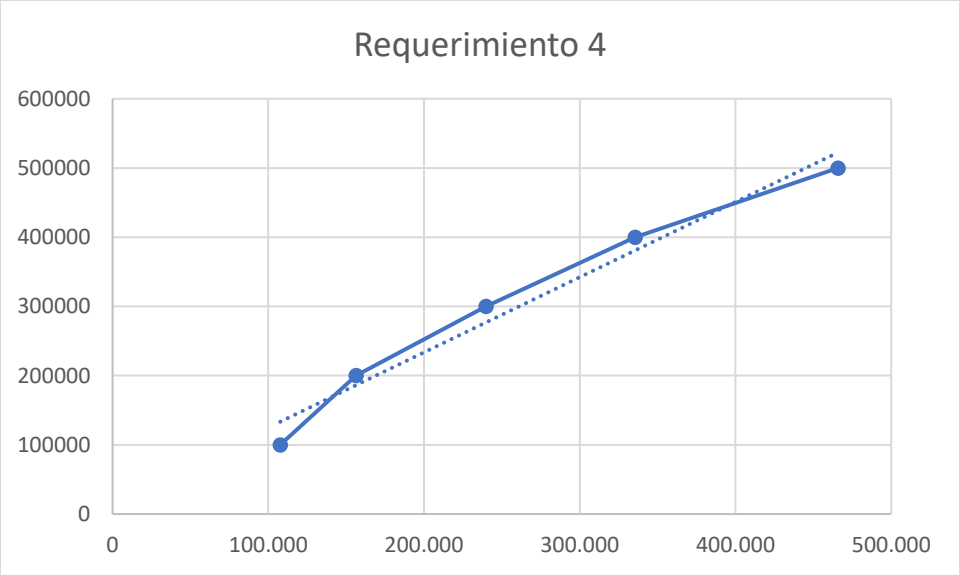
Graficas

Las gráficas con la representación de las pruebas realizadas.

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10



Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11



Análisis

Por temas de retorno decidimos usar Array List, ya que nos facilita el acceso a elementos con get_element() y es de complejidad O(1), la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar los Array List varias veces poder realizar filtros, añadir elementos y obtener elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por poco es una recta completa, indicando su crecimiento lineal.

Requerimiento 6

Descripción

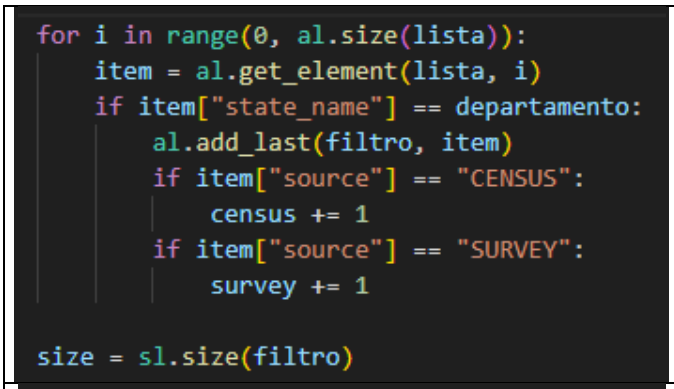
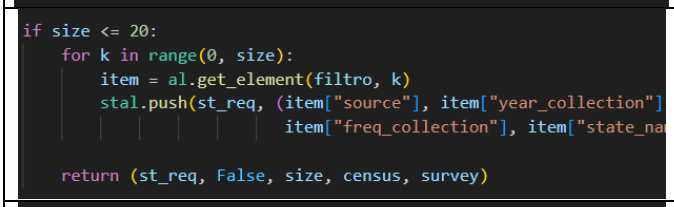
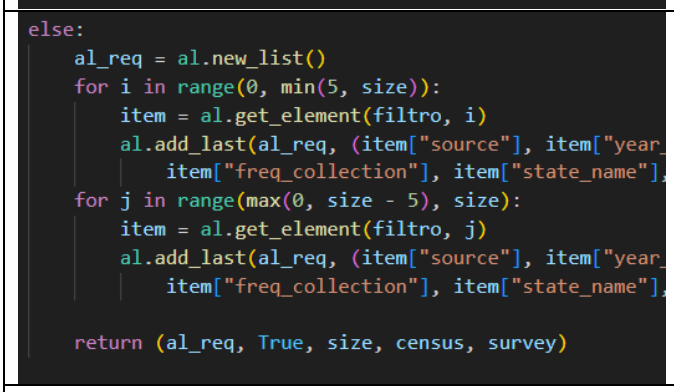
El punto del requerimiento es conocer la estadística del departamento de mi interés para un rango de fechas de carga de los registros dado. Por este motivo, use una función auxiliar para que me dé un Array List filtrado entre fechas de carga que funciona con la librería datetime. Desde acá es más fácil recorrer de nuevo un Array List más pequeño y filtrarlo por departamento. En caso de que el número de registros sea menor o igual a 20 devuelvo una Pila, en caso contrario, devuelvo un Array List con acceso a elementos.

Entrada	`agro_al, departamento:str, fecha_inicial:str, fecha_final:str)`
Salidas	A) Tupla con pila, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. B) Tupla con Array List, boolean, size, numero de registros tipo celsus, numero de registros tipo survey.
	Implementado, Iker Barbosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
`filtro = al.new_list()` `st_req = stal.new_stack()` `census = 0` `survey = 0` `lista = buscar_entre_fechas_al(agro_al, fecha_inicial, fecha_fina` `def buscar_entre_fechas_al(agro_al, fecha_inicia` `try:` `fecha_inicio = datetime.strptime(fecha_i` `fecha_fin = datetime.strptime(fecha_fina` `except ValueError:` `return("Error: tipo de dato no válido")` `registros = al.new_list()` `for i in range(0, al.size(agro_al["agricultu` `registro = al.get_element(agro_al["agric` `fecha_load = datetime.strptime(registro[` `if fecha_inicio <= fecha_load <= fecha_f` `al.add_last(registros, registro)` `return registros`	O(n), la función auxiliar buscar_entre_fechas_al() es una función que recorre todo el Array List para filtrar entre fechas de carga recibidas por parámetro.

	O(n), recorrer la lista filtrada entre fechas de carga para hallar los registros que coincidan con el tipo de departamento que se recibió por parámetro.
	O(n), si el tamaño de lista es menor o igual a 20, recorremos la lista para añadir sus elementos a una pila y retornarla.
	O(n), en caso de que el tamaño de la lista sea mayor a 20. Creamos una nueva Array List con new_list(), e iteramos la lista para añadir los primeros 5 elementos y los últimos 5 con get.element() y retornamos el Array List.
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos, el estado de CALIFORNIA, la fecha de carga 2012-01-01 00:00:00 y la fecha de carga 2024-11-07 12:00:00 .

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 6: 571.472 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 6: 1167.862 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 6: 1743.198 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 6: 2334.033 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 6: 2891.148 [ms]

Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 6: 1733.116 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 6: 2089.620 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 6: 2867.754 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 6: 4201.119 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 6: 5515.121 [ms]

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

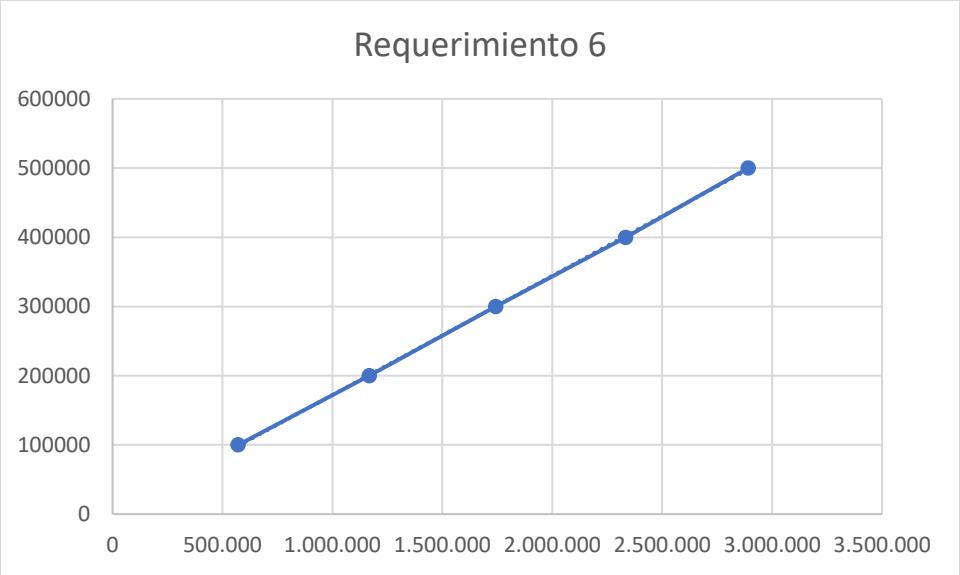
Muestra	**Salida**	**Tiempo (ms)**
100000 datos	('CENSUS', '2012', '2012-12-31', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('SURVEY', '1929', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('SURVEY', '2003', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'LB', ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERAT ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERAT ('SURVEY', '1999', '2012-01-01', 'WEEKLY', 'CALIFORNIA', 'HEAD', ('CENSUS', '2002', '2012-01-01', 'POINT IN TIME', 'CALIFORNIA', ('CENSUS', '1997', '2012-01-01', 'ANNUAL', 'CALIFORNIA', '$', 'M ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERAT ('CENSUS', '2007', '2012-01-01', 'POINT IN TIME', 'CALIFORNIA', Total de registros: 2155 Total de registros con tipo de fuente/origen "CENSUS": 1360 Total de registros con tipo de fuente/origen "SURVEY": 795	571.472
200000 datos	('CENSUS', '2012', '2012-12-31', 'ANNUAL', 'CALIFORNIA', 'HEAD', 'G ('SURVEY', '1929', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', 'W ('SURVEY', '2003', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'LB', 'MI ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATION ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATION ('CENSUS', '2007', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'OPERATION ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', 'E ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATION ('SURVEY', '2023', '2024-03-15', 'ANNUAL', 'CALIFORNIA', '$', 'HONE ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', '$', 'SHEE Total de registros: 4330 Total de registros con tipo de fuente/origen "CENSUS": 2771 Total de registros con tipo de fuente/origen "SURVEY": 1559	1167.862

300000 datos	('CENSUS', '2012', '2012-12-31', 'ANNUAL', 'CALIFORNIA', 'HEAD', ' ('SURVEY', '1929', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', ' ('SURVEY', '2003', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'LB', 'M ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATIO ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATIO ('SURVEY', '1992', '2012-01-01', 'POINT IN TIME', 'CALIFORNIA', 'H ('SURVEY', '1969', '2012-01-01', 'WEEKLY', 'CALIFORNIA', 'HEAD', ' ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATIO ('SURVEY', '2001', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'HEAD', ('CENSUS', '2002', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'NUMBER', Total de registros: 6585 Total de registros con tipo de fuente/origen "CENSUS": 4229 Total de registros con tipo de fuente/origen "SURVEY": 2356 Tiempo de ejecución para el requerimiento 6: 1743.198 [ms]	1743.198
400000 datos	('CENSUS', '2012', '2012-12-31', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('SURVEY', '1929', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('SURVEY', '2003', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'LB', ' ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('SURVEY', '1976', '2012-01-01', 'POINT IN TIME', 'CALIFORNIA', ' ('SURVEY', '1991', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'HEAD', ('SURVEY', '2017', '2019-06-25', 'MONTHLY', 'CALIFORNIA', 'HEAD', Total de registros: 8763 Total de registros con tipo de fuente/origen "CENSUS": 5609 Total de registros con tipo de fuente/origen "SURVEY": 3154	2334.033
500000 datos	('CENSUS', '2012', '2012-12-31', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('SURVEY', '1929', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('SURVEY', '2003', '2012-01-01', 'MONTHLY', 'CALIFORNIA', 'LB', ' ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('CENSUS', '2017', '2018-02-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('SURVEY', '1994', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'HEAD', ('CENSUS', '2012', '2012-12-31', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('CENSUS', '2007', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI ('CENSUS', '2012', '2012-12-31', 'POINT IN TIME', 'CALIFORNIA', ' ('CENSUS', '2007', '2012-01-01', 'ANNUAL', 'CALIFORNIA', 'OPERATI Total de registros: 10983 Total de registros con tipo de fuente/origen "CENSUS": 7013 Total de registros con tipo de fuente/origen "SURVEY": 3970	2891.148

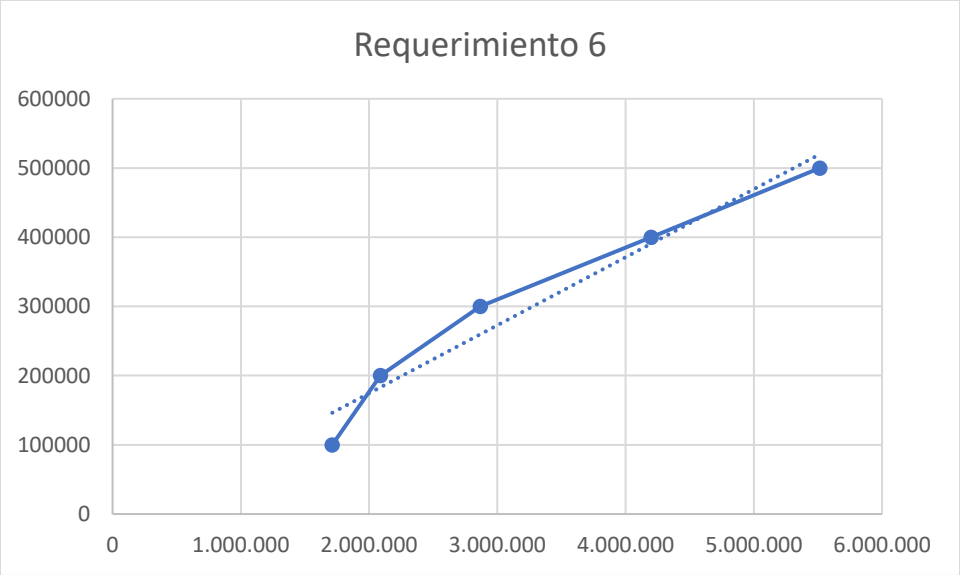
Graficas

Las gráficas con la representación de las pruebas realizadas.

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10



Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11



Análisis

Similar al requerimiento 4, decidimos usar Array List, ya que nos facilita el acceso a elementos con get_element() y es de complejidad O(1), la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar los Array List varias veces poder realizar filtros, añadir elementos y obtener elementos.

Este comportamiento se puede evidenciar perfectamente en la gráfica, especialmente en la del Ryzen 7 5700G. Una recta completa, lo que indica su crecimiento lineal.

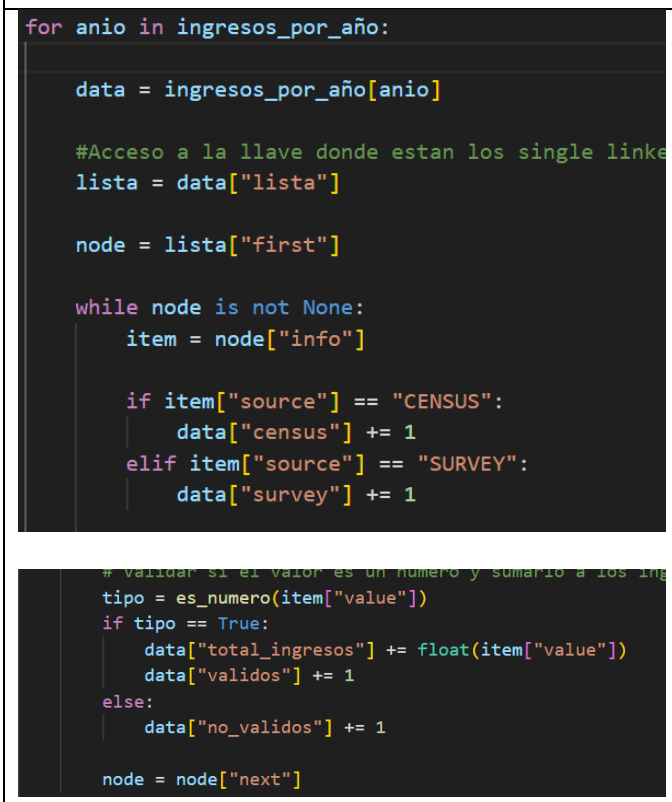
Requerimiento 7

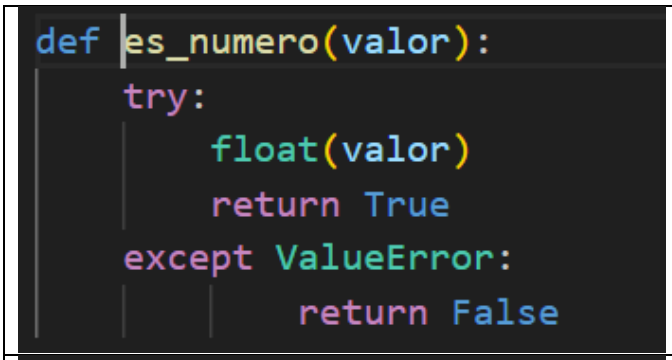
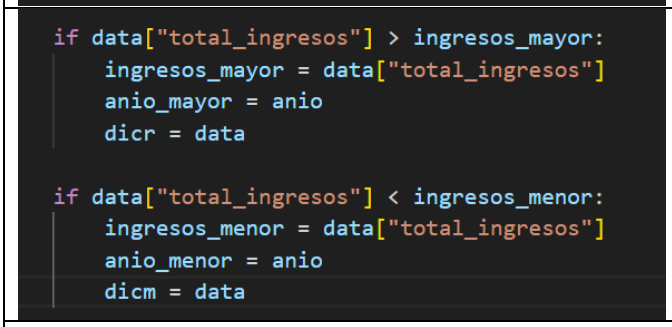
Descripción

El punto del requerimiento es conocer el año con mayor y menor ingresos para un departamento de interés en un rango de fechas de recolección de los registros dado. Para esta consulta, solo se deben tener en cutna los registros que contengan un “$”.

Entrada	`def req_7(agro, departmento: str, año_inicio: str, año_fin: str):`
Salidas	`return ((anio_mayor,"MAYOR",ingresos_mayor, dicr["validos"], dicr["no_validos"], dicr["survey"], dicr["census"]), (anio_menor, "MENOR", ingresos_menor, dicm["validos"], dicm["no_validos"], dicm["survey"], dicm["census"]), total_registros)`
Implementado (Sí/No)	Implementado, Iker Barbosa. Camilo Castro

Pasos	**Complejidad**
`año_inicio = int(año_inicio)` `año_fin = int(año_fin)` `lista = buscar_entre_anios(agro, año_inicio, año_fin)` `def buscar_entre_anios(agro, fecha_inicio:int, fecha_fin` `"""` `Retorna una single linked list con los registros que` `"""` `registros = sl.new_list()` `node = agro['agricultural_records']['first']` `while node is not None:` `if int(node['info']['year_collection']) >= int(f` `sl.add_last(registros, node['info'])` `node = node['next']` `return registros`	O(n), buscar_entre_anios() es una función que se usa para recorrer la lista completa y filtrar entre dos años que recibe por parámetro. El retorno de esta función se guarda en la variable lista.
`lista_filtrada = remover_value_lista(lista, departmento)` `def remover_value_lista(filtro, departamento):` `lista = sl.new_list()` `node = filtro["first"]` `while node is not None:` `item = node["info"]` `if item["state_name"] == departamento and "$" in item["unit_measurement"` `sl.add_last(lista, item)` `node = node["next"]` `return(lista)`	O(n), la variable lista_filtrada guarda el valor de retorno de la función remover_value_lista(). Esta función recibe un departamento por parametro y se encarga de filtrar por departamento, guarda solamente las funciones que contengan $ en su unidad de medida y también ignora los que contienen $ y value (D)
`total_registros = sl.size(lista_filtrada)`	O(1), guarda en la variable total_registros el tamaño de la lista_filtrada calculada anteriormente.

 	O(n), se crea un diccionario para poder almacenar los ingresos por año. Se recorre la lista filtrada y se extraen los años para que sean llaves de diccionario. Diccionarios de listas enlazadas. También cuenta con las llaves lista, que almacena las listas enlazadas del año en cuestión, total_ingresos, que almacena la sumatoria de ingresos, census, que almacena el conteo de registros tipo CENSUS, survey, que almacena el conteo de registros tipo SURVEY, validos, almacena el conteo de registros validos, no_validos, que almacena el conteo de registros no_validos. La inserción en la última posición es O(1)
 	Una vez creados los diccionarios, se recorren y se accede a la llave lista. El recorrido de esta llave es O(n), ya que se recorren listas enlazadas por años dentro del diccionario y se hacen los respectivos incrementos al propio diccionario. Como el número de años es fijo, la complejidad completa es O(n), ya que se recorren solamente una vez. La función es_numero() O(1), solamente verifica si el valor del registro se puede convertir a flotante, si retorna True, cuenta como registro valido. De lo contrario retorna false y se cuenta como registro no valido.

	
	*O(n), ya que en el peor de los casos estas comparaciones se harían n veces si tenemos muchos años.*
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos, CALIFORNIA, 1970, 2024.

Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 7: 136.391 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 7: 432.197 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 7: 710.842 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 7: 982.640 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 7: 1298.784 [ms]

Muestra	**Salida**	**Tiempo (ms)**

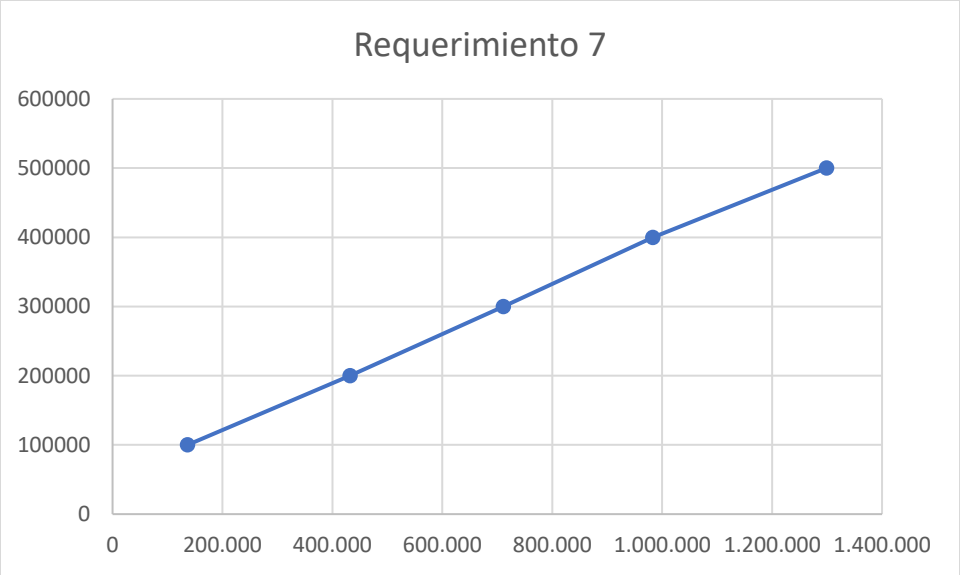
100000 datos	Año de recopilación: 1992 Indicación del periodo: MAYOR Valor de ingresos del periodo: 794.4 Número de registros validos: 2 Número de registros no validos: 1 Número de registros tipo SURVEY: 3 Número de registros tipo CENSUS: 0 Año de recopilación: 2015 Indicación del periodo: MENOR Valor de ingresos del periodo: 0 Número de registros validos: 0 Número de registros no validos: 1 Número de registros tipo SURVEY: 1 Número de registros tipo CENSUS: 0	136.391
200000 datos	Año de recopilación: 1992 Indicación del periodo: MAYOR Valor de ingresos del periodo: 826.75 Número de registros validos: 4 Número de registros no validos: 1 Número de registros tipo SURVEY: 5 Número de registros tipo CENSUS: 0 Año de recopilación: 2015 Indicación del periodo: MENOR Valor de ingresos del periodo: 0 Número de registros validos: 0 Número de registros no validos: 2 Número de registros tipo SURVEY: 2 Número de registros tipo CENSUS: 0	432.197
300000 datos	Año de recopilación: 1992 Indicación del periodo: MAYOR Valor de ingresos del periodo: 898.11 Número de registros validos: 7 Número de registros no validos: 1 Número de registros tipo SURVEY: 8 Número de registros tipo CENSUS: 0 Año de recopilación: 2017 Indicación del periodo: MENOR Valor de ingresos del periodo: 0 Número de registros validos: 0 Número de registros no validos: 49 Número de registros tipo SURVEY: 1 Número de registros tipo CENSUS: 48	710.842

400000 datos	Indicación del periodo: MAYOR Valor de ingresos del periodo: 898.11 Número de registros validos: 7 Número de registros no validos: 2 Número de registros tipo SURVEY: 9 Número de registros tipo CENSUS: 0 Año de recopilación: 2016 Indicación del periodo: MENOR Valor de ingresos del periodo: 0 Número de registros validos: 0 Número de registros no validos: 3 Número de registros tipo SURVEY: 2 Número de registros tipo CENSUS: 1	982.640
500000 datos	Indicación del periodo: MAYOR Valor de ingresos del periodo: 898.11 Número de registros validos: 7 Número de registros no validos: 2 Número de registros tipo SURVEY: 9 Número de registros tipo CENSUS: 0 Año de recopilación: 2016 Indicación del periodo: MENOR Valor de ingresos del periodo: 0 Número de registros validos: 0 Número de registros no validos: 4 Número de registros tipo SURVEY: 2 Número de registros tipo CENSUS: 2	1298.784

Graficas

Las gráficas con la representación de las pruebas realizadas.

Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11



Análisis

El requerimiento 7 presenta un crecimiento lineal. Se recorre al inicio dos veces para cumplir ciertos filtros, se recorre otra vez para crear diccionarios de listas enlazadas y se recorren al final para hacer las sumatorias y comparaciones necesarias para el retorno.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

Requerimiento 8

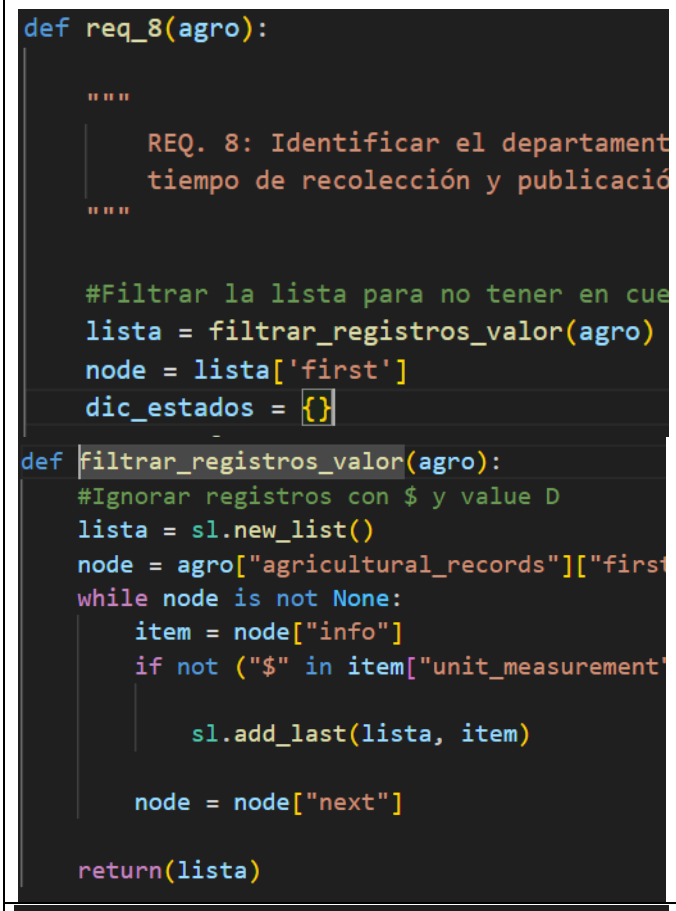
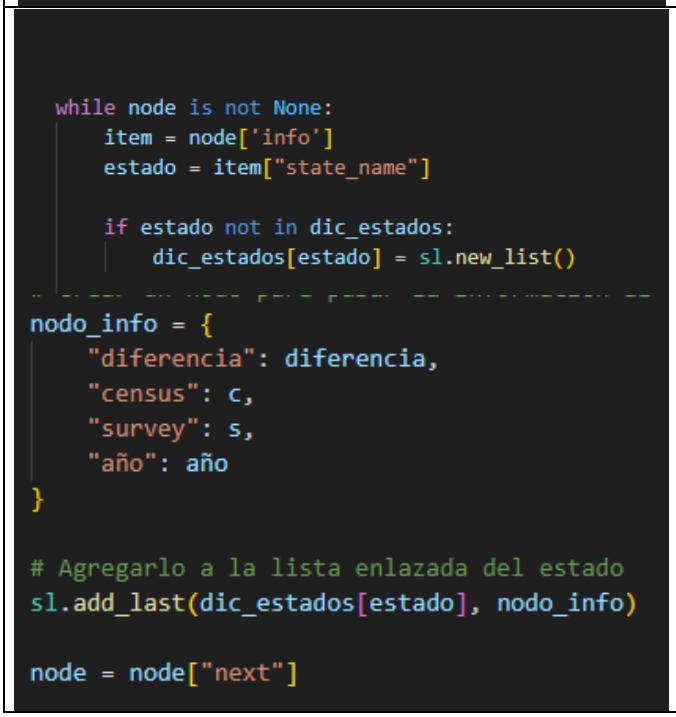
Descripción

El punto del requerimiento es conocer el cuál es el departamento con mayor diferencia promedio de tiempo entre la recopilación de los registros y su carga a la plataforma de entre todos los registros. Para este requerimiento decidimos crear un diccionario de listas enlazadas en donde la llave del diccionario representa un departamento y la lista enlazada contiene la información necesaria para saber la mayor diferencia. Para el cálculo de diferencias hicimos uso de la librería Datetime.

Entrada	`req_8(agro):`
Salidas	`ret = (estado, promedio, año_max, año_min,diferencia_max, diferencia_min, census, survey)`
Implementado (Sí/No)	Implementado, Iker Barbosa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

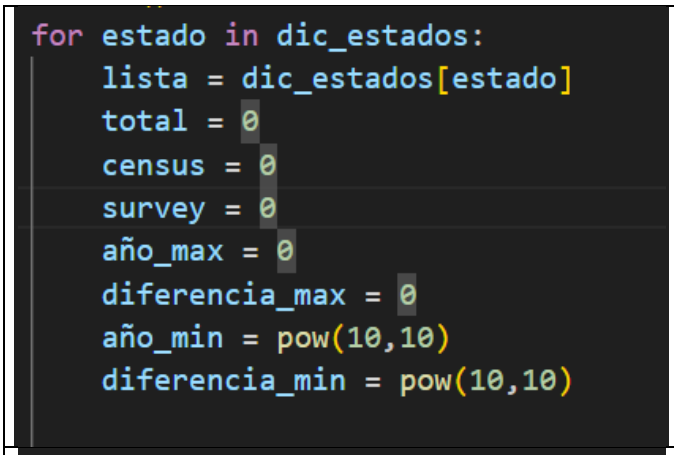
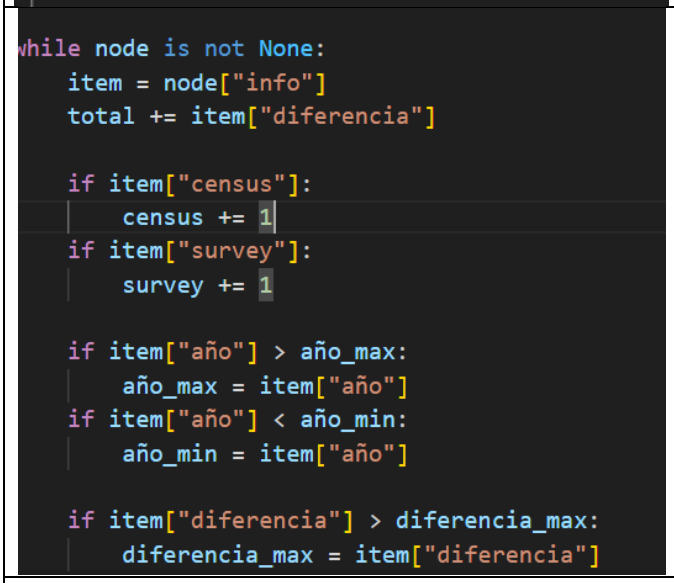
Pasos	Complejidad
	O(n), filtrar los registros para evitar añadir los valores que contengan “$” y (“D”). Se recorre la lista completa y se retorna la lista filtrada.
	O(n), recorrer la lista para así crear un diccionario donde las llaves son cada departamento. Si no existe, crea la llave y crea una nueva Single Linked List con sl.new_list(). En caso de que existe, crea un nodo con información, O(1) para añadirlo al final en la Single Linked List con sl.add_last().

```
def req_8(agro):

    """
        REQ. 8: Identificar el departament
        tiempo de recolección y publicació
    """

    #Filtrar la lista para no tener en cue
    lista = filtrar_registros_valor(agro)
    node = lista['first']
    dic_estados = {}
```

```
def filtrar_registros_valor(agro):
    #Ignorar registros con $ y value D
    lista = sl.new_list()
    node = agro["agricultural_records"]["first
    while node is not None:
        item = node["info"]
        if not ("$" in item["unit_measurement"

            sl.add_last(lista, item)

        node = node["next"]

    return(lista)
```

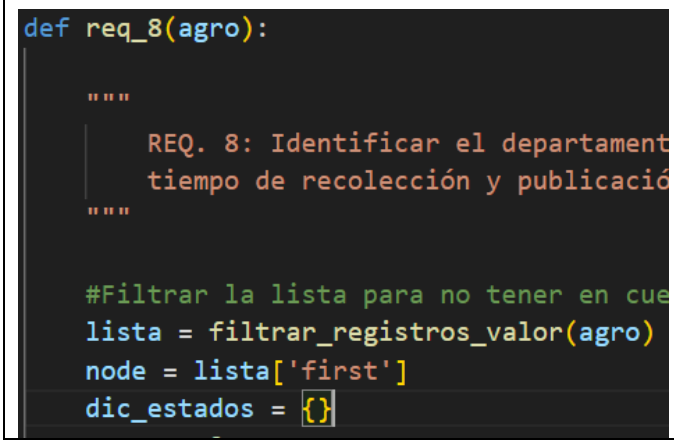
```
  while node is not None:
      item = node['info']
      estado = item["state_name"]

      if estado not in dic_estados:
          dic_estados[estado] = sl.new_list()
```

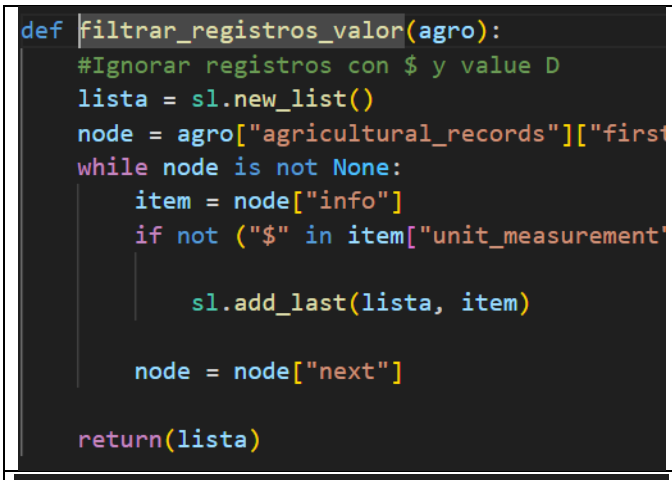
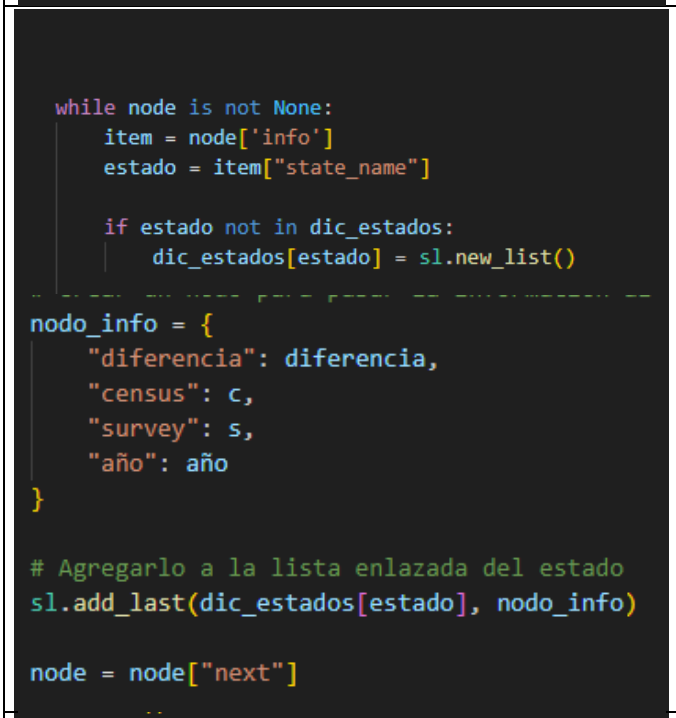
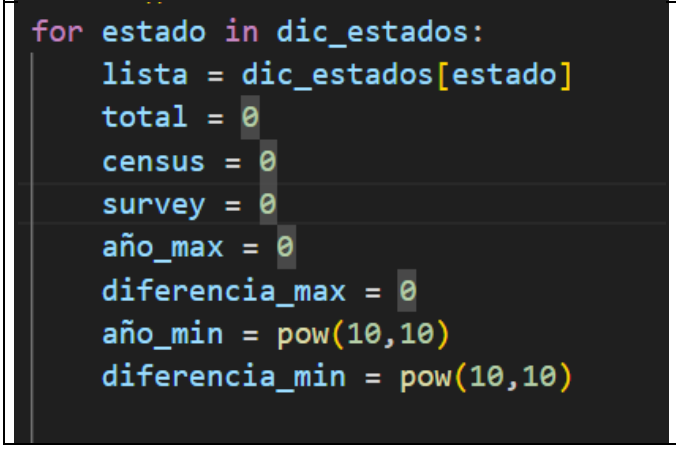
```
nodo_info = {
    "diferencia": diferencia,
    "census": c,
    "survey": s,
    "año": año
}

# Agregarlo a la lista enlazada del estado
sl.add_last(dic_estados[estado], nodo_info)

node = node["next"]
```

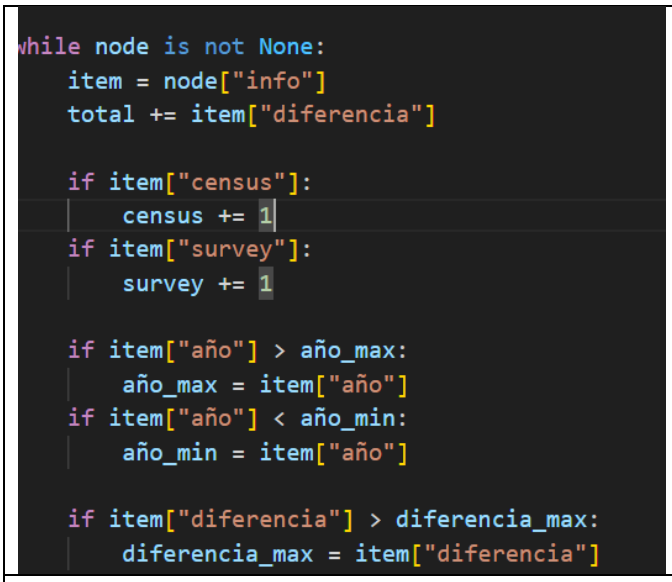
	O(N), Se accede a las llaves del diccionario y se recorre la lista enlazada solamente una vez por estado hasta el final para realizar las comparaciones y sumatorias necesarias.
	
TOTAL	***O(n)***

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
	O(n), filtrar los registros para evitar añadir los valores que contengan “$” y (“D”)

	
	O(n), recorrer la lista para así crear un diccionario donde las llaves son cada departamento. Si no existe, crea la llave y crea una nueva Single Linked List con sl.new_list(). En caso de que existe, crea un nodo con información, para añadirlo en la Single Linked List con sl.add_last().
	O(k), el diccionario tiene un numero de estados fijo. Por lo tanto, se recorre k veces, siendo k una constante.

	O(N), recorre la lista enlazada hasta el final para realizar las comparaciones y sumatorias necesarias.
TOTAL	***O(n)***

Pruebas Realizadas

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

Procesadores	**AMD Ryzen 7 5700G**
Memoria RAM	22 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 8: 1150.047 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 8: 2524.231 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 8: 3260.040 [ms]
400000 datos	Tiempo de ejecución para el requerimiento 8: 4533.304 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 8: 5719.113 [ms]

Procesadores	**12th Gen Intel(R) Core(TM) i5-12450H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
100000 datos	Tiempo de ejecución para el requerimiento 8: 1592.890 [ms]
200000 datos	Tiempo de ejecución para el requerimiento 8: 2872.686 [ms]
300000 datos	Tiempo de ejecución para el requerimiento 8: 4478.743 [ms]

400000 datos	Tiempo de ejecución para el requerimiento 8: 5546.357 [ms]
500000 datos	Tiempo de ejecución para el requerimiento 8: 6736.069 [ms]

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

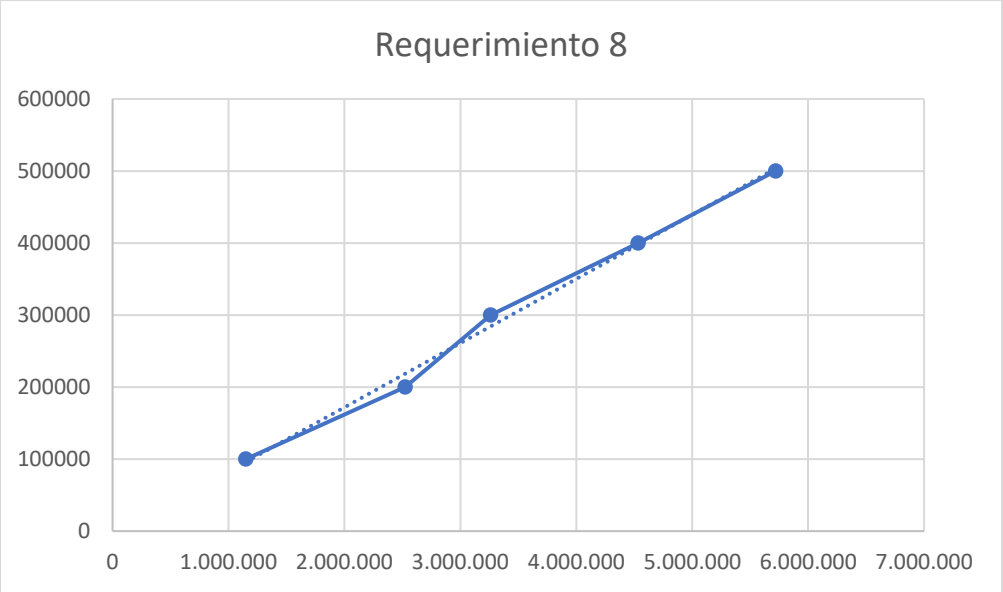
Muestra	Salida	Tiempo (ms)	
100000 datos	Numero total de departamentos: 59 Mayor año de carga: 2024 Menor año de carga: 1867 Años promedio de tiempos de carga: 2015.17 Departamento con mayor diferencia promedio: Estado: NEBRASKA Promedio: 18.96 Año máximo de recolección: 2024 Año mínimo de recolección: 1875 Diferencia máxima calculada: 137 Diferencia mínima calculada: 0 Registros tipo Census: 1635 Registros tipo Survey: 1372 Registros del departamento: 3007 Tiempo de ejecución para el requerimiento 8: 1150.04	1150.047	
200000 datos	Numero total de departamentos: 59 Mayor año de carga: 2024 Menor año de carga: 1867 Años promedio de tiempos de carga: 2015.17 Departamento con mayor diferencia promedio: Estado: NEBRASKA Promedio: 19.12 Año máximo de recolección: 2024 Año mínimo de recolección: 1875 Diferencia máxima calculada: 137 Diferencia mínima calculada: 0 Registros tipo Census: 3184 Registros tipo Survey: 2743 Registros del departamento: 5927 Tiempo de ejecución para el requerimiento 8: 2524.2	2524.231	

300000 datos	Numero total de departamentos: 59 Mayor año de carga: 2024 Menor año de carga: 1866 Años promedio de tiempos de carga: 2015.18 Departamento con mayor diferencia promedio: Estado: NEBRASKA Promedio: 19.12 Año máximo de recolección: 2024 Año mínimo de recolección: 1875 Diferencia máxima calculada: 137 Diferencia mínima calculada: 0 Registros tipo Census: 4777 Registros tipo Survey: 4089 Registros del departamento: 8866 Tiempo de ejecución para el requerimiento 8: 3260.040	3260.040	
400000 datos	Numero total de departamentos: 59 Mayor año de carga: 2024 Menor año de carga: 1866 Años promedio de tiempos de carga: 2015.17 Departamento con mayor diferencia promedio: Estado: NEBRASKA Promedio: 19.0 Año máximo de recolección: 2024 Año mínimo de recolección: 1875 Diferencia máxima calculada: 137 Diferencia mínima calculada: 0 Registros tipo Census: 6418 Registros tipo Survey: 5401 Registros del departamento: 11819 Tiempo de ejecución para el requerimiento 8: 4533.304	4533.304	
500000 datos	Numero total de departamentos: 59 Mayor año de carga: 2024 Menor año de carga: 1866 Años promedio de tiempos de carga: 2015.17 Departamento con mayor diferencia promedio: Estado: NEBRASKA Promedio: 19.04 Año máximo de recolección: 2024 Año mínimo de recolección: 1868 Diferencia máxima calculada: 144 Diferencia mínima calculada: 0 Registros tipo Census: 7963 Registros tipo Survey: 6740 Registros del departamento: 14703 Tiempo de ejecución para el requerimiento 8: 5719.1	5719.113	

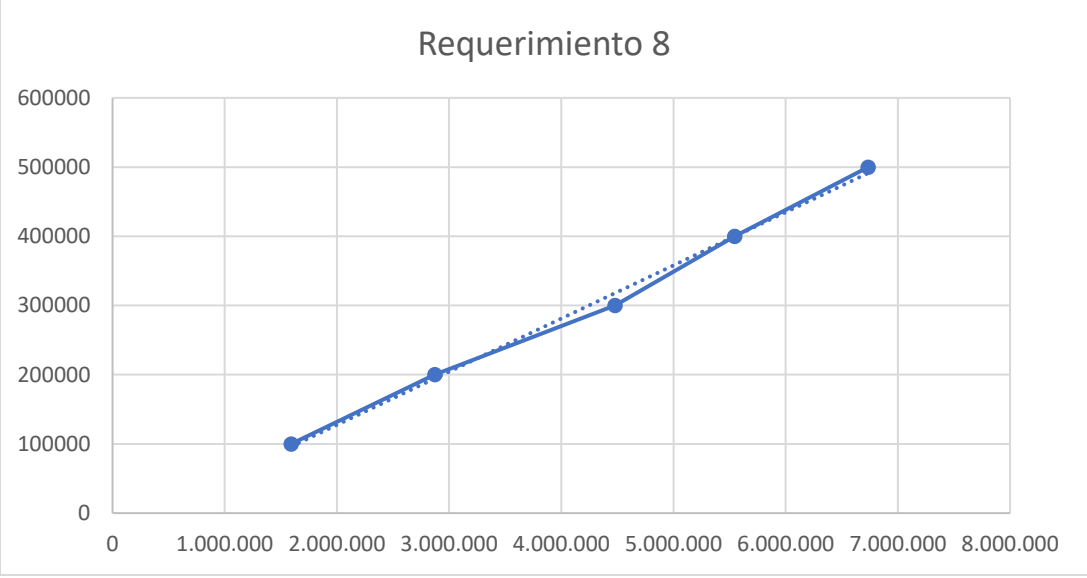
Graficas

Las gráficas con la representación de las pruebas realizadas.

Procesadores	AMD Ryzen 7 5700G
Memoria RAM	22 GB
Sistema Operativo	Windows 10



Procesadores	12th Gen Intel(R) Core(TM) i5-12450H
Memoria RAM	16 GB
Sistema Operativo	Windows 11



Análisis

El requerimiento 8 presenta un crecimiento lineal. Simplemente se recorre por completo una primera vez para filtrar los datos, una segunda para crear diccionarios de listas enlazadas y luego se recorre el diccionario completo para poder hacer los cálculos necesarios.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

Por la cantidad de variables, comparaciones y sumatorias que se deben realizar aumenta su tiempo de ejecución, comparado a otros requerimientos.