Análisis de resultados

Integrantes:

- Gabriela González Gómez - 202121554
- Daniel Felipe Diab G-202212289

Requerimientos individuales:

Req_3: Gabriela González Gómez
Req_4: Daniel Felipe Diab G
Req_5: Juan Manuel

Complejidades de cada requerimiento:

Requerimiento 1:

Función: logic.req_1(control, min_duracion)

1. Acceso a lt.size(catalog['duracion']):
    - Asumimos que lt.size tiene complejidad O(1), ya que está obteniendo el tamaño de una lista.
    - Complejidad: O(1)
2. Iteración sobre catalog['duracion']:
    - El bucle for i in range(0, total_peliculas) recorre todos los elementos de catalog['duracion']. Si total_peliculas es el número de elementos en catalog['duracion'], el bucle itera exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número de películas.
3. Obtención de la duración de cada película:
    - Dentro del bucle, se realiza lt.get_element(catalog['duracion'], i) para obtener la duración de la película en la posición i. Suponiendo que esta operación tiene complejidad O(1) para cada elemento, se ejecuta una vez por cada iteración del bucle.
    - Complejidad por iteración: O(1)
4. Conversión de la duración a float:
    - El bloque try-except que convierte la duración a float tiene complejidad O(1) por cada iteración, ya que la conversión de cadenas cortas a flotantes es una operación constante.
    - Complejidad por iteración: O(1)
5. Condicional para verificar la duración:
    - La condición if duracion >= min_duracion se evalúa en tiempo constante O(1) por cada iteración.
    - Complejidad por iteración: O(1)
6. Obtención de la fecha:
    - Si la duración cumple la condición, se ejecuta lt.get_element(catalog['fecha'], i) para obtener la fecha de la película, lo cual nuevamente tiene complejidad O(1) por cada acceso.
    - Complejidad por iteración: O(1) Complejidad total: O(n) (solo para las películas que cumplen la condición).

7. Inserción de películas en peliculas_con_fecha:
    - Agregar un elemento a una lista de Python (peliculas_con_fecha.append) tiene complejidad amortizada O(1).
    - Complejidad total: O(n)
8. Ordenamiento de peliculas_con_fecha:
    - Después de recolectar las películas que cumplen la condición, se ejecuta peliculas_con_fecha.sort(). El algoritmo de ordenamiento utilizado por Python es Timsort, cuya complejidad es O(n log n), donde n es el número de elementos a ordenar. En el peor de los casos, n (todas las películas cumplen la condición).
    - Complejidad del ordenamiento: O(n log n), donde n es el número de películas que cumplen la condición.
9. Acceso a la película más reciente:
    - El acceso a la película más reciente en la lista ordenada es una operación constante O(1).
    - Complejidad: O(1)
10. Llamada a get_data:
    - Finalmente, se llama a la función get_data(catalog, peli_reciente_id). Si asumimos que get_data también tiene complejidad O(1), esta operación es constante.
    - Complejidad: O(1)

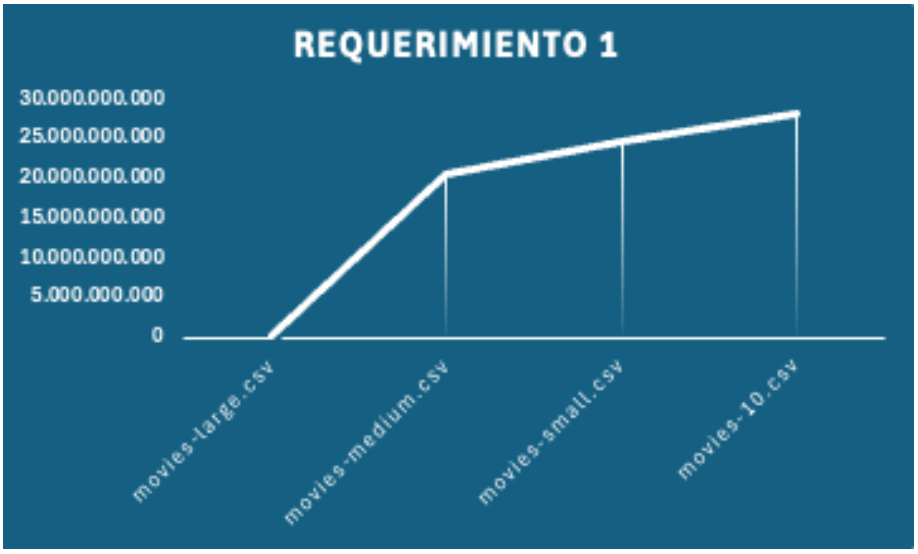
Complejidad total:

- En el peor de los casos, el bucle principal recorre todas las películas, realiza operaciones de acceso y comparación en O(1), y luego ordena la lista resultante.
- El tiempo total para la función se puede descomponer en dos partes principales:
    1. El bucle que tiene una complejidad de O(n).
    2. El ordenamiento de las películas que cumplen la condición, que tiene complejidad O(n log n), donde n es el número de películas que cumplen la condición.

Por lo tanto, la complejidad en el peor caso de la función es: O(n log n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo de ejecución (mm)
Req 1	movies-large.csv	310582.659
Req 1	movies-medium.csv	2042088.7641
Req 1	movies-small.csv	2484692.8956
Req 1	movies-10.csv	2807591.7105



Análisis de comparación de resultados:

El análisis de los resultados muestra que el tiempo de ejecución crece de manera proporcional al tamaño de los datos, lo cual es consistente con la complejidad O(n log n) en el peor de los casos, dominada por la iteración sobre las películas (O(n)) y el ordenamiento de las que cumplen la condición (O(n log n)). Las operaciones individuales de acceso y comparación, que tienen complejidad O(1), no tienen un impacto significativo en los tiempos. Sin embargo, el ordenamiento de las películas es el paso que contribuye más al aumento de tiempo en archivos grandes, como se observa en los resultados de movies-large.csv.

Requerimiento 2:

Función: logic.req_2(control, idioma)

- Acceso a lt.size(movies['fecha']):
    - Asumimos que lt.size tiene complejidad O(1), ya que solo está obteniendo el tamaño de la lista movies['fecha'].
    - Complejidad: O(1)
- Iteración sobre movies['fecha']:
    - El bucle for i in range(0, total_peliculas) recorre todas las películas. Si total_peliculas es el número de elementos en movies['fecha'], el bucle itera exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número total de películas en el catálogo.
- Obtención del idioma de cada película:
    - Dentro del bucle, se realiza lt.get_element(movies['idioma'], i) para obtener el idioma de la película en la posición i. Asumimos que esta operación tiene complejidad O(1) por cada acceso.
    - Complejidad por iteración: O(1)
- Comparación del idioma:
    - La comparación if idioma_pelicula == idioma_buscado se evalúa en tiempo constante O(1) para cada iteración, dado que es una simple operación de comparación entre cadenas.

    - Complejidad por iteración: O(1)
- Actualización de variables:
    - Si el idioma coincide con idioma_buscado, se actualiza el contador buscadas y se asigna una nueva referencia a ultima_pelicula. Ambas operaciones son constantes (O(1)).
    - Complejidad por iteración (si se cumple la condición): O(1)
- Obtención de datos de la película:
    - Si la película tiene el idioma buscado, se llama a get_data(movies, i), que en este caso se asume que tiene una complejidad O(1), dado que está accediendo a los datos de una película específica.
    - Complejidad por iteración (si se cumple la condición): O(1)
- Manejo de excepciones:
    - Si no se encontraron películas en el idioma buscado (buscadas == 0), se lanza una excepción IndexError. Lanzar una excepción es una operación O(1).
    - Complejidad: O(1)

Complejidad total:

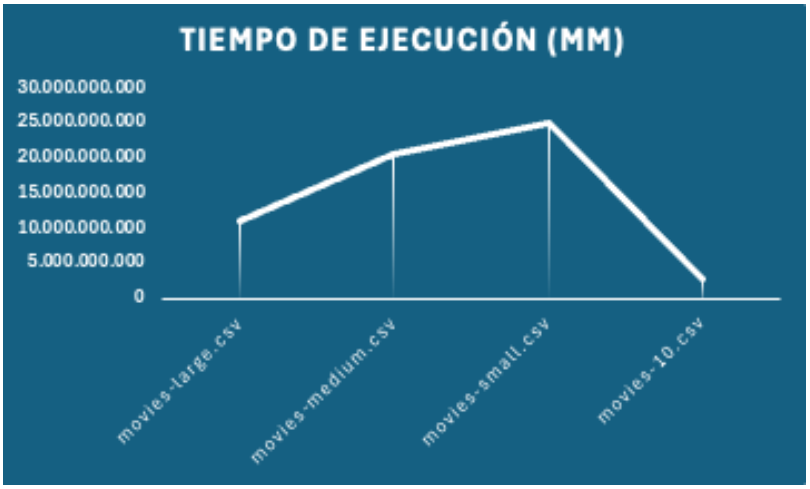
La función recorre la lista de películas una vez, verificando el idioma de cada película y actualizando las variables cuando encuentra una coincidencia. Todos los accesos y comparaciones son de tiempo constante O(1) y se realizan una vez por cada película, por lo que la complejidad de la función es dominada por el bucle principal que itera sobre todas las películas.

Por lo tanto, la complejidad en notación O de esta función es:

- O(n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo de ejecución (mm)
Req_2	movies-large.csv	1115503.3651
Req_2	movies-medium.csv	2058185.4404
Req_2	movies-small.csv	2513110.9814
Req_2	movies-10.csv	2850695.414



Análisis de comparación de resultados:

El análisis muestra que el tiempo de ejecución aumenta proporcionalmente al tamaño de los datos, lo que concuerda con la complejidad esperada de O(n). La función está dominada por un bucle que itera sobre todas las películas del catálogo, lo que significa que el tiempo de ejecución crece linealmente a medida que aumenta el número de películas en los archivos, como se refleja en los tiempos de ejecución para movies-large.csv, movies-medium.csv y movies-small.csv. Las operaciones de acceso y comparación dentro del bucle, que tienen complejidad constante O(1), no impactan significativamente el rendimiento general. Esto confirma que la complejidad observada corresponde al análisis teórico.

Requerimiento 3:

Función: logic.req_3(control, idioma, fecha_inicio, fecha_final)

1. Acceso a lt.size(catalog['fecha']):
    - Asumimos que la función lt.size tiene complejidad O(1), ya que simplemente obtiene el tamaño de la lista de fechas.
    - Complejidad: O(1)
2. Iteración sobre catalog['fecha']:
    - El bucle for i in range(0, total_peliculas) recorre todas las películas del catálogo. Si total_peliculas es el número total de películas en el catálogo, el bucle itera exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número total de películas.
3. Obtención de la fecha e idioma de cada película:
    - Dentro del bucle, se realiza lt.get_element(catalog['fecha'], i) para obtener la fecha de la película en la posición i y lt.get_element(catalog['idioma'], i) para obtener el idioma de la película. Asumimos que ambas operaciones tienen complejidad O(1) cada una.
    - Complejidad por iteración: O(1)
4. Comparación del idioma:
    - La comparación if idioma_pelicula == idioma es una simple operación de comparación entre cadenas, que tiene complejidad constante O(1).
    - Complejidad por iteración: O(1)

5. Conversión de fechas:
    - La conversión de las fechas con datetime.strptime se realiza para la fecha de la película, la fecha de inicio y la fecha final. Cada conversión de cadena a fecha tiene una complejidad O(1), ya que la operación trabaja con cadenas de longitud constante.
    - Complejidad por iteración: O(1) para cada conversión.
6. Verificación de rango de fechas:
    - La condición if fecha_inicio_dt <= fecha_pelicula <= fecha_final_dt se evalúa en tiempo constante O(1), porque se trata de una comparación entre objetos de tipo datetime.
    - Complejidad por iteración: O(1)
7. Obtención de los datos de la película:
    - La llamada a get_data(catalog, i) tiene complejidad O(1), asumiendo que accede a los datos de una película en tiempo constante.
    - Complejidad por iteración: O(1)
8. Procesamiento de los datos de la película:
    - La construcción de la lista pelicula_filtrada a partir de los datos de la película tiene complejidad O(1) por iteración, ya que se está extrayendo una cantidad fija de elementos.
    - Complejidad por iteración: O(1)
9. Suma de la duración de las películas:
    - El código que intenta convertir la duración de la película a float y sumar dicha duración tiene complejidad O(1) por iteración. La excepción ValueError también se maneja en tiempo constante.
    - Complejidad por iteración: O(1)
10. Filtrado final de películas:
    - Si se encuentran más de 20 películas, la función toma las primeras 5 y las últimas 5. Esta operación de obtener subconjuntos de la lista tiene complejidad O(1), ya que se están accediendo a índices específicos de una lista.
    - Complejidad: O(1)
11. Cálculo de la duración promedio:
    - El cálculo de la duración promedio duracion_promedio = duracion_total / total_filtradas es una operación aritmética de complejidad constante O(1).
    - Complejidad: O(1)

Complejidad total:

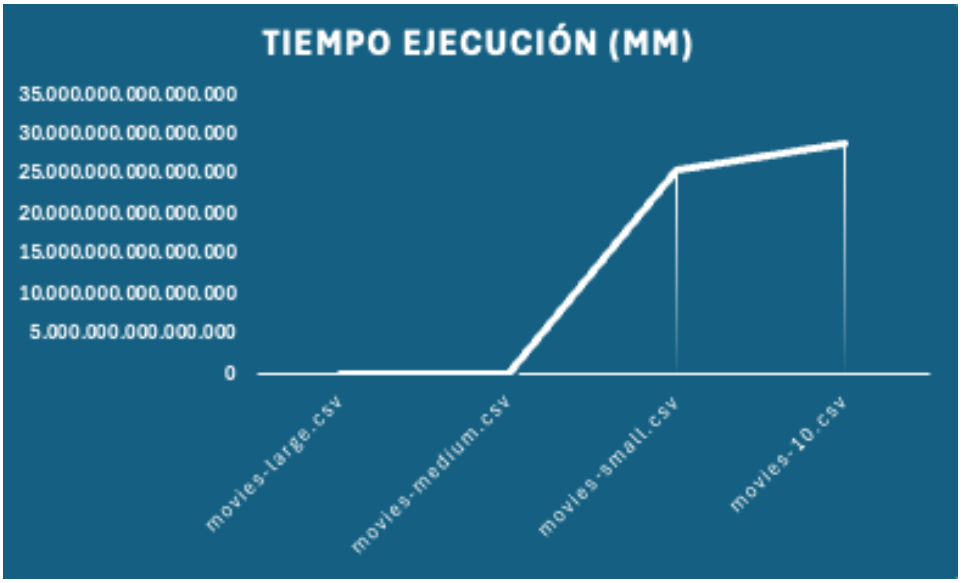
La complejidad total está dominada por el bucle principal, que itera sobre todas las películas en el catálogo y realiza operaciones de tiempo constante O(1) en cada iteración.

Por lo tanto, la complejidad en notación O de esta función es:

O(n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo ejecución (mm)
req_3	movies-large.csv	1592400.8758
req_3	movies-medium.csv	2091435.5369
req_3	movies-small.csv	2567880.7794999997
req_3	movies-10.csv	2891488.0979999998



Análisis de comparación de resultados:

El análisis muestra que el tiempo de ejecución aumenta de manera lineal con el tamaño de los datos, lo que concuerda con la complejidad esperada de O(n). El bucle principal, que itera sobre todas las películas del catálogo, domina el tiempo de ejecución, lo que se refleja en el incremento de los tiempos a medida que el tamaño de los archivos aumenta. Las operaciones internas, como la obtención de fechas, idiomas, comparaciones y conversiones de fecha, tienen complejidad O(1) por iteración y no contribuyen significativamente al aumento del tiempo. Los tiempos para archivos más grandes como movies-large.csv y movies-medium.csv son consistentes con el comportamiento esperado de la función, lo que confirma que la complejidad observada en los resultados coincide con el análisis teórico O(n).

Requerimiento 4:

Función: logic.req_4(control, status_bs, f_inicial, f_final)

1. Inicialización de variables:
    - La creación de pelis_bs = {"peli": [], "size": 0} tiene complejidad constante O(1), ya que solo se están inicializando estructuras vacías.

    - La conversión de fechas fecha_inicio_dt = datetime.strptime(f_inicial, formato_fecha) y fecha_final_dt = datetime.strptime(f_final, formato_fecha) también tiene complejidad O(1) cada una, ya que se están trabajando con fechas en formato constante.
    - Complejidad: O(1)
2. Acceso a lt.size(catalog['fecha']):
    - Se obtiene el tamaño de la lista de fechas con lt.size(catalog['fecha']), lo que asume una complejidad O(1).
    - Complejidad: O(1)
3. Iteración sobre las películas del catálogo:
    - El bucle for i in range(0, total_peliculas) recorre todas las películas del catálogo. Si el catálogo contiene total_peliculas, entonces el bucle iterará exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número total de películas.
4. Obtención de la fecha y el estado de cada película:
    - Dentro del bucle, lt.get_element(catalog['fecha'], i) y lt.get_element(catalog['status'], i)son operaciones de acceso a los elementos del catálogo. Suponiendo que cada acceso tiene una complejidad O(1), estas operaciones tienen una complejidad constante por iteración.
    - Complejidad por iteración: O(1)
5. Conversión de la fecha de la película:
    - La conversión de la fecha de la película datetime.strptime(fecha, formato_fecha) tiene una complejidad O(1) por iteración, ya que el formato de la fecha es constante.
    - Complejidad por iteración: O(1)
6. Comparación del estado y las fechas:
    - La condición if status == status_bs and fecha_inicio_dt <= fecha_pelicula <= fecha_final_dt compara el estado y la fecha de la película. Las comparaciones entre cadenas y fechas tienen complejidad constante O(1).
    - Complejidad por iteración: O(1)
7. Actualización de variables:
    - Si la película cumple las condiciones, se actualizan pelis_bs["size"] y duracion_promedio, que son operaciones O(1).
    - Complejidad por iteración: O(1)
8. Obtención de otros datos de la película:
    - Las llamadas a lt.get_element para obtener los datos como presupuesto, ingresos, idioma, etc., tienen complejidad O(1) por cada acceso, y hay múltiples accesos en cada iteración cuando la película cumple las condiciones.
    - Complejidad por iteración: O(1)
9. Cálculo de las ganancias:
    - La lógica para calcular ganancias verifica tipos de datos y realiza operaciones aritméticas simples, lo que también tiene complejidad constante O(1).
    - Complejidad por iteración: O(1)
10. Adición de información de la película a la lista:
    - La operación pelis_bs["peli"].append(informacion) para agregar los datos de la película a la lista tiene complejidad amortizada O(1).

- Complejidad por iteración: O(1)

11. Cálculo de la duración promedio:
    - El cálculo de la duración promedio duracion_promedio = duracion_promedio / peliculas_contadas tiene complejidad O(1), ya que es una operación aritmética sobre números.
    - Complejidad: O(1)
12. Filtrado final de películas:
    - Si se encuentran más de 20 películas, se seleccionan las primeras 5 y las últimas 5 películas con resultado = pelis_bs["peli"][:5] + pelis_bs["peli"][-5:]. Este es un acceso a un subconjunto de la lista, lo que tiene complejidad O(1), ya que se acceden a posiciones fijas en la lista.
    - Complejidad: O(1)

Complejidad total:

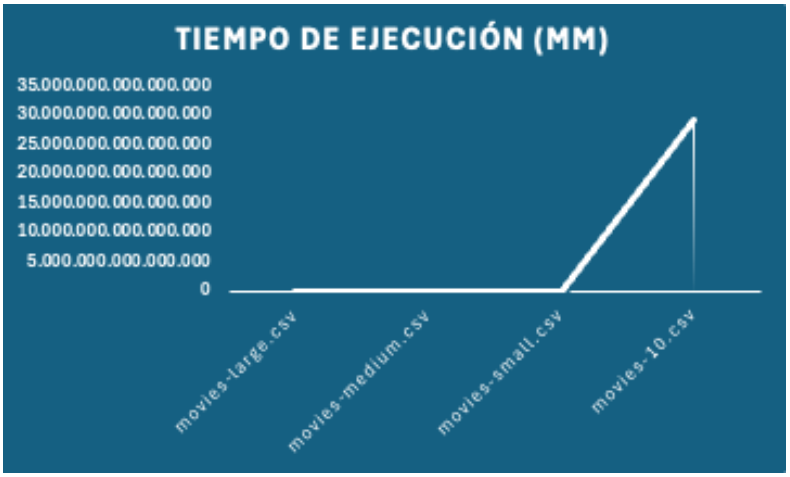
La parte más costosa de la función es el bucle que itera sobre todas las películas en el catálogo y realiza varias operaciones constantes dentro de cada iteración. Dado que la cantidad de operaciones dentro del bucle es constante, la complejidad está dominada por el número total de películas.

Por lo tanto, la complejidad en notación O de esta función es:

O(n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo de ejecución (mm)
Req_4	movies-large.csv	1755212.141
Req_4	movies-medium.csv	2145071.9533
Req_4	movies-small.csv	2608944.9844
Req_4	movies-10.csv	2930712.4705000003



Análisis de comparación de resultados:

El análisis muestra un crecimiento lineal en el tiempo de ejecución a medida que aumenta el tamaño del archivo, lo que concuerda con la complejidad O(n) de la función. El tiempo de ejecución está dominado por el bucle principal que recorre todas las películas del catálogo, realizando operaciones de tiempo constante O(1) en cada iteración, como la obtención de datos, conversiones de fecha, comparaciones y actualizaciones de variables. El incremento en los tiempos de ejecución para archivos más grandes como movies-large.csv y movies-medium.csv refleja este comportamiento lineal. Esto confirma que la complejidad observada en los tiempos de ejecución es consistente con el análisis teórico O(n).

Requerimiento 5:

Función: logic.req_5(control, fecha_inicial, fecha_final, limite_inferior, limite_superior)

-

Requerimiento 6:

Función: logic.req_6(control, idioma_original, año_inicial_consulta, año_final_consulta)

1. Conversión de años a enteros:
    - La conversión de los años de consulta (año_inicial_consul y año_final_consul) a enteros tiene complejidad O(1), ya que se trata de una operación simple de tipo de dato. En caso de que no sean válidos, se lanza una excepción, lo cual también es una operación constante O(1).
    - Complejidad: O(1)
2. Acceso a lt.size(catalog["fecha"]):
    - El tamaño de la lista de películas se obtiene con lt.size(catalog["fecha"]), lo cual asume una complejidad O(1).
    - Complejidad: O(1)
3. Iteración sobre las películas del catálogo:
    - El bucle for i in range(1, total_peliculas + 1) recorre todas las películas del catálogo. Si el catálogo contiene total_peliculas películas, el bucle itera exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número total de películas.
4. Obtención de los datos de la película:
    - Dentro del bucle, se llama a get_data(catalog, i) para obtener los datos de cada película. Asumimos que esta operación tiene complejidad O(1) por cada llamada.
    - Complejidad por iteración: O(1)
5. Conversión de la fecha de publicación a un año:
    - La conversión de la fecha de publicación de la película a un objeto datetime y la extracción del año se realiza con datetime.strptime(fecha_publicacion, "%Y-%m-%d").year. La operación datetime.strptime tiene complejidad constante O(1), ya que está procesando una cadena de formato fijo.
    - Complejidad por iteración: O(1)
6. Verificación del rango de años, idioma y estado:

    - La condición if año_inicial_consulta <= año_publicacion <= año_final_consulta and idioma_pelicula == idioma_original and status == "Released" compara el año de la película, el idioma y el estado, lo cual tiene complejidad constante O(1) por iteración.
    - Complejidad por iteración: O(1)
7. Acceso y actualización de datos en el diccionario resultados_por_año:
    - Si la película cumple las condiciones, se accede al diccionario resultados_por_año. Si la clave correspondiente al año no existe, se inicializa con un nuevo diccionario. Tanto el acceso como la inserción en un diccionario tienen una complejidad promedio de O(1).
    - Complejidad por iteración: O(1)
8. Cálculo y acumulación de duraciones, votaciones y ganancias:
    - La duración de la película, el promedio de votación y las ganancias se suman a los totales del año correspondiente. Todas estas operaciones son aritméticas simples, que tienen complejidad constante O(1) por iteración.
    - Complejidad por iteración: O(1)
9. Comparación de votaciones para determinar la mejor y peor película:
    - En cada iteración, se compara la votación de la película actual con la votación de la mejor y peor película registrada hasta el momento. Estas comparaciones son operaciones constantes O(1) por iteración.
    - Complejidad por iteración: O(1)
10. Cálculo de promedios al final del bucle:
    - Después de completar el bucle sobre las películas, se itera sobre cada año en el diccionario resultados_por_año para calcular los promedios de votación y duración. Dado que el número de años es limitado (dependiendo del rango de años especificado), este bucle tendrá una cantidad limitada de iteraciones (digamos m, que es el número de años dentro del rango). En la práctica, m es mucho menor que n, por lo que este bucle es insignificante en comparación con la iteración sobre las películas.
    - Complejidad: O(n), donde n es el número de años distintos dentro del rango.

Complejidad total:

La parte más costosa de la función es el bucle que itera sobre todas las películas en el catálogo, y dentro de este bucle se realizan varias operaciones constantes. Por lo tanto, la complejidad total está dominada por la iteración sobre las películas.

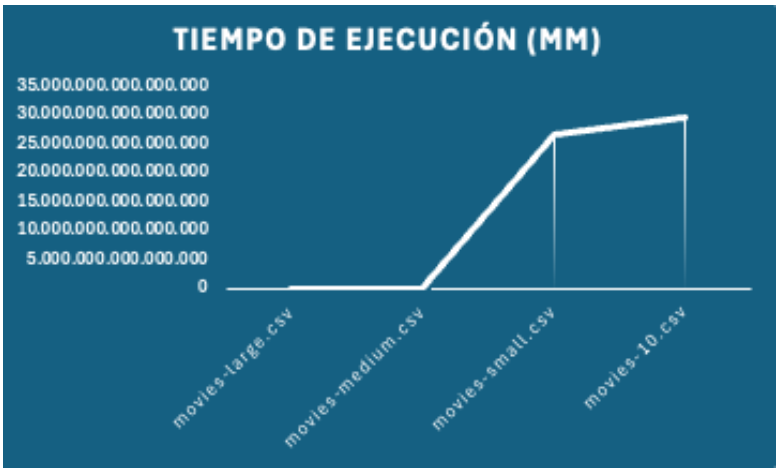
Por lo tanto, la complejidad en notación O de esta función es:

O(n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo de ejecución (mm)
Req_6	movies-large.csv	1811860.8506
Req_6	movies-medium.csv	2188311.9112

Req_6	movies-small.csv	2654606.7081999998
Req_6	movies-10.csv	2954719.3145000003



Análisis de comparación de resultados:

El análisis muestra que el tiempo de ejecución crece de manera lineal con el tamaño de los datos, lo que coincide con la complejidad esperada de O(n). La parte más costosa de la función es el bucle que itera sobre todas las películas del catálogo, realizando operaciones de tiempo constante O(1) en cada iteración, como la obtención de datos, conversión de fechas, comparaciones y actualización de valores. El incremento en los tiempos de ejecución desde movies-large.csv hasta movies-10.csv refleja este comportamiento lineal, lo cual confirma que la complejidad observada en los tiempos de ejecución es consistente con el análisis teórico O(n).

Requerimiento 7:

Función: logic.req_7(control, compania, anio_inicio, anio_final)

1. Conversión de los años de inicio y final a enteros:
    - Las conversiones de anio_inicio y anio_final a enteros tienen complejidad constante O(1) cada una.
    - Complejidad: O(1)
2. Inicialización del diccionario peliculas_por_anio:
    - El diccionario peliculas_por_anio se inicializa para cada año entre anio_inicio y anio_final. El número de años en este rango es m=anio_final−anio_inicio+1m=anio_final−anio_inicio+1. Por lo tanto, la inicialización del diccionario tiene una complejidad O(n), donde mm es la cantidad de años.
    - Complejidad: O(n)
3. Acceso a lt.size(catalog['fecha']):
    - El tamaño del catálogo se obtiene con lt.size(catalog['fecha']), lo que asume una complejidad O(1).
    - Complejidad: O(1)
4. Iteración sobre las películas del catálogo:

    - El bucle for i in range(total_peliculas) recorre todas las películas del catálogo. Si el catálogo contiene total_peliculas películas, el bucle itera exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número de películas en el catálogo.
5. Obtención de la fecha, compañías y estado de la película:
    - Dentro del bucle, lt.get_element se utiliza para obtener la fecha, las compañías y el estado de cada película. Asumimos que cada una de estas operaciones de acceso tiene complejidad O(1).
    - Complejidad por iteración: O(1)
6. Conversión de la fecha de la película a un año:
    - La conversión de la fecha de la película a un objeto datetime y la extracción del año con datetime.strptime(fecha, "%Y-%m-%d").year tiene complejidad constante O(1), ya que se está procesando una cadena de formato fijo.
    - Complejidad por iteración: O(1)
7. Verificación del rango de años, compañía y estado:
    - La condición if anio_inicio <= anio_pelicula <= anio_final and compania in companias_pelicula and status_pelicula == "Released" verifica el año de la película, la compañía y el estado. Las comparaciones y la verificación de pertenencia (compania in companias_pelicula) tienen una complejidad O(n), donde n es el número de compañías asociadas con la película (siendo pequeño en la mayoría de los casos).
    - Complejidad por iteración: O(n)
8. Obtención y procesamiento de los datos de la película:
    - La obtención de los datos de la película pelicula_data = get_data(catalog, i) tiene complejidad O(1) por cada llamada.
    - Calcular la duración, ganancias y votación promedio de la película son operaciones aritméticas simples que tienen complejidad constante O(1) por iteración.
    - La actualización de los datos del diccionario para cada año también tiene complejidad O(1).
    - Complejidad por iteración: O(1)
9. Comparación de las votaciones para encontrar la mejor y peor película:
    - La comparación de las votaciones de cada película para determinar la mejor y peor película del año se realiza en tiempo constante O(1) por iteración.
    - Complejidad por iteración: O(1)
10. Construcción del resultado final:
    - Después de procesar todas las películas, se itera sobre el diccionario peliculas_por_anio para calcular los promedios de duración y votación. Este bucle itera mm veces (donde mm es el número de años) y realiza operaciones O(1) por año.
    - Complejidad: O(n)

Complejidad total:

La parte más costosa de la función es el bucle que itera sobre todas las películas en el catálogo, y dentro de este bucle se realizan varias operaciones de tiempo constante o dependientes del número de compañías asociadas a la película. Por lo tanto, la complejidad total está dominada por la iteración sobre las películas.

Por lo tanto, la complejidad en notación O de esta función es:

O(n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo de ejecución (mm)
Req_7	movies-large.csv	1861988.3299
Req_7	movies-medium.csv	2257929.8647000003
Req_7	movies-small.csv	2690007.5143
Req_7	movies-10.csv	2990808.3391

Análisis de comparación de resultados:

El análisis muestra un crecimiento lineal en el tiempo de ejecución a medida que aumenta el tamaño de los datos, lo que concuerda con la complejidad teórica O(n). La parte dominante de la función es el bucle que itera sobre todas las películas en el catálogo, realizando operaciones de tiempo constante O(1) en la mayoría de los casos, con una excepción en la verificación de las compañías asociadas a las películas, que podría implicar una pequeña variación en algunos casos. Los tiempos de ejecución, que van desde movies-large.csv hasta movies-10.csv, reflejan este comportamiento lineal, confirmando que el análisis teórico O(n) es consistente con los tiempos observados en la ejecución.

Requerimiento 8:

Función: logic.req_8(control, anio_consulta, genero_consulta)

1. Inicialización de variables:
    - Inicializar las variables como pelis_bs, total_votos, total_duracion, etc., tiene complejidad constante O(1).
    - Complejidad: O(1)
2. Acceso a lt.size(catalog['fecha']):

    - Obtener el tamaño del catálogo usando lt.size(catalog['fecha']) tiene complejidad O(1).
    - Complejidad: O(1)
3. Iteración sobre las películas del catálogo:
    - El bucle for i in range(1, total_peliculas + 1) recorre todas las películas en el catálogo. Si el catálogo contiene total_peliculas películas, este bucle iterará exactamente total_peliculas veces.
    - Complejidad: O(n), donde n es el número de películas en el catálogo.
4. Obtención de los datos de la película:
    - Dentro del bucle, se obtienen varios elementos del catálogo como lt.get_element(catalog['fecha'], i), lt.get_element(catalog["genero"], i), lt.get_element(catalog["status"], i), etc. Asumimos que cada uno de estos accesos tiene complejidad O(1).
    - Complejidad por iteración: O(1)
5. Conversión de la fecha a un año:
    - La conversión de la fecha de la película a un objeto datetime y la extracción del año se realiza con datetime.strptime(fecha, formato_fecha).year, lo que tiene una complejidad O(1) por iteración, ya que se está trabajando con una cadena de formato fijo.
    - Complejidad por iteración: O(1)
6. Verificación del año, género y estado:
    - La condición if año_pelicula == consulta and genero in genero_gn and estado == "Released"verifica el año de la película, si el género está presente y si el estado es "Released". La verificación de pertenencia genero in genero_gn tiene una complejidad O(n), donde n es el número de géneros asociados a la película.
    - Complejidad por iteración: O(n)
7. Acumulación de votos, duración y ganancias:
    - Las operaciones de acumulación de votos, duración y ganancias son aritméticas simples que tienen complejidad constante O(1) por iteración.
    - La excepción ValueError en el cálculo de las ganancias también se maneja en tiempo constante O(1).
    - Complejidad por iteración: O(1)
8. Comparación para encontrar la mejor y peor película:
    - Las comparaciones para determinar la mejor y peor película se realizan en tiempo constante O(1) por iteración.
    - Complejidad por iteración: O(1)
9. Cálculo de promedios:
    - El cálculo de los promedios de votación y duración se realiza fuera del bucle principal y tiene complejidad constante O(1), ya que son operaciones simples de división.
    - Complejidad: O(1)

Complejidad total:

La parte más costosa de la función es el bucle que itera sobre todas las películas en el catálogo y realiza varias operaciones en cada iteración. En el peor caso, el costo de las operaciones dentro del bucle dependerá del número de géneros asociados a cada película.

Por lo tanto, la complejidad total es:

O(n)

Tablas de tiempo de ejecución y gráficas comparativas:

	Archivo	Tiempo de ejecución (mm)
Req_8	movies-large.csv	1901552.6136999999
Req_8	movies-medium.csv	2313996.2042
Req_8	movies-small.csv	2736367.5817
Req_8	movies-10.csv	3040980.8046

Análisis de comparación de resultados:

El análisis muestra que el tiempo de ejecución crece de manera lineal con el tamaño del archivo, lo que coincide con la complejidad esperada de O(n). El bucle principal, que itera sobre todas las películas del catálogo y realiza operaciones de tiempo constante O(1), es la parte dominante. La verificación de si un género está presente en una película tiene una complejidad ligeramente mayor O(n), donde n es el número de géneros asociados a cada película, pero esto no parece afectar significativamente el crecimiento general del tiempo de ejecución. Los tiempos para archivos más grandes, como movies-large.csv y movies-medium.csv, reflejan este comportamiento lineal, confirmando que la complejidad observada en los tiempos es consistente con el análisis teórico O(n).