Integrantes G05:

Juan David Huertas Canchala, 202413851, jd.huertasc1@uniandes.edu.co
Juan Camilo Defelipe Malagon, 202410537, j.defelipe@uniandes.edu.co

Organización de requerimientos:

Para este reto realizamos los requerimientos 1, 3, 5, 6 y 7.
Requerimiento 1 : Grupal
Requerimiento 3 : Individual (Juan David Huertas C.)
Requerimiento 5 : Individual (Juan Camilo Defelipe M.)
Requerimiento 6 : Grupal
Requerimiento 7 : Grupal

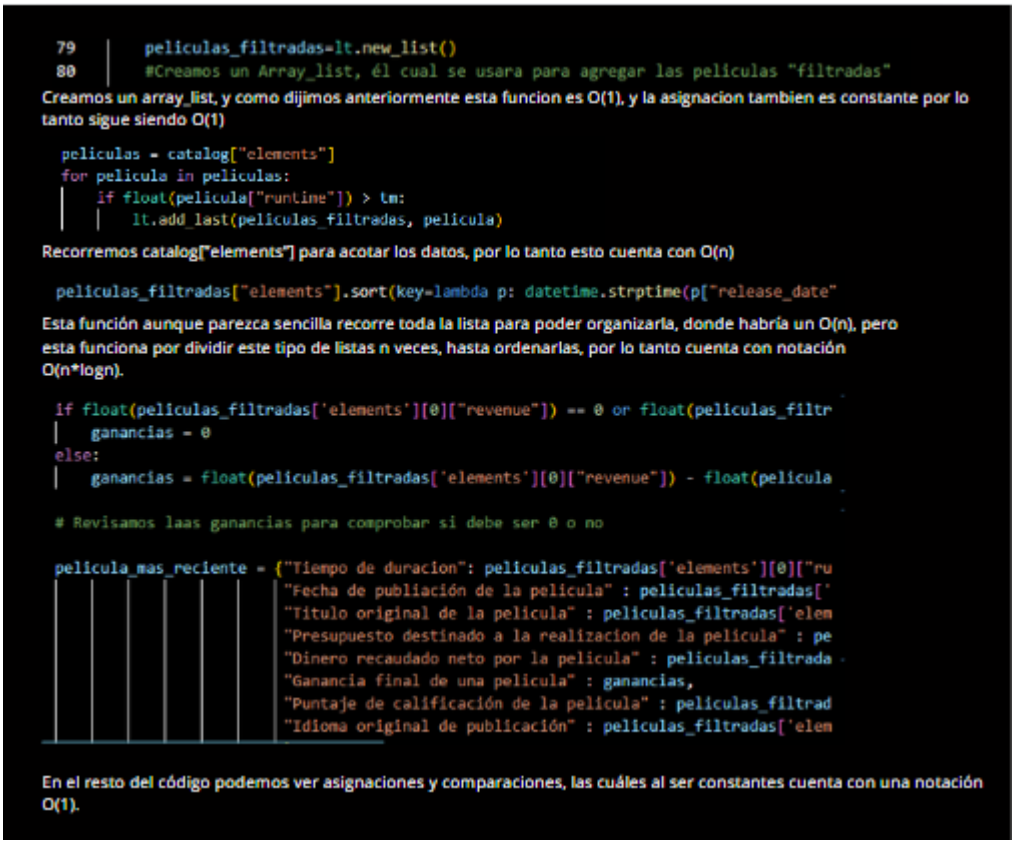
Análisis de complejidad en Notación O:

Load Data:

Para analizar la función Load Data, la dividiremos para analizar de mejor forma las operaciones que se Realizan en cada una:

Conclusión: La mayoría de la estructura de load_data(), está conformada por asignaciones las cuales al ser constantes, tienen una complejidad de O(1). Sin embargo, podemos observar que al momento de recorrer las películas extraídas del archivo .csv, se recorre con un for, lo cual hace que obtenga complejidad O(n), siendo esta la mayor y concluyendo que la notación O de Load_data() es O(n).

Requerimiento _1:



```
79      peliculas_filtradas=lt.new_list()
80      #Creamos un Array_list, él cual se usara para agregar las peliculas "filtradas"
```

Creamos un array_list, y como dijimos anteriormente esta funcion es O(1), y la asignacion tambien es constante por lo tanto sigue siendo O(1)

```
peliculas = catalog["elements"]
for pelicula in peliculas:
    if float(pelicula["runtime"]) > tm:
        lt.add_last(peliculas_filtradas, pelicula)
```

Recorremos catalog["elements"] para acotar los datos, por lo tanto esto cuenta con O(n)

```
peliculas_filtradas["elements"].sort(key=lambda p: datetime.strptime(p["release_date"
```

Esta función aunque parezca sencilla recorre toda la lista para poder organizarla, donde habría un O(n), pero esta funciona por dividir este tipo de listas n veces, hasta ordenarlas, por lo tanto cuenta con notación O(n*logn).

```
if float(peliculas_filtradas['elements'][0]["revenue"]) == 0 or float(peliculas_filtr
    ganancias = 0
else:
    ganancias = float(peliculas_filtradas['elements'][0]["revenue"]) - float(pelicula

# Revisamos laas ganancias para comprobar si debe ser 0 o no

pelicula_mas_reciente = {"Tiempo de duracion": peliculas_filtradas['elements'][0]["ru
                         "Fecha de publiación de la pelicula" : peliculas_filtradas['
                         "Titulo original de la pelicula" : peliculas_filtradas['elem
                         "Presupuesto destinado a la realizacion de la pelicula" : pe
                         "Dinero recaudado neto por la pelicula" : peliculas_filtrada
                         "Ganancia final de una pelicula" : ganancias,
                         "Puntaje de calificación de la pelicula" : peliculas_filtrad
                         "Idioma original de publicación" : peliculas_filtradas['elem
```

En el resto del código podemos ver asignaciones y comparaciones, las cuáles al ser constantes cuenta con una notación O(1).

Conclusión: Del mismo modo que en Load_Data(), esta función cuenta mayoritariamente con funciones constantes, es decir asignaciones, comparaciones o condicionales. Más sin embargo, al recorrer la lista para filtrar los elementos se realizó un O(n).Posteriormente hallamos en la función proporcionada por la biblioteca DataTime para organizar fechas, nos damos cuenta que esta cuenta con una notación O(n*logn) siendo esta la mayor de todas. Por lo tanto, la notación del requerimiento _1 es O(n¨logn)

Requerimiento _3:

Conclusión: Vemos que el requerimiento 3 se acompleja más por su gran extensión. Sin embargo en cuanto a su notación en O sigue contando con O(n) porque la mayoría de este código sino completamente está conformado por asignaciones y comparaciones las cuales al ser constantes cuentan con una notación O(1).
En conclusión, el requerimiento 3 cuenta con una notación O(n)

Requerimiento _4:

```
# se crea un una lista que guardara las peliculas que cumplen los criterios
pelis = []
# se crea una variable para sumar la duracion de todas las peliculas que cumplan los criterios
total_duracion = 0
# se crea una variable que suma las peliculas que cumplen los criterios
total_pelis = 0
# se convierten "fi" y "ff" a obajetos tipo datetiDme para poder hacer comparaciones de fechas
fecha_i = datetime.strptime(fi, "%Y-%m-%d")
fecha_f = datetime.strptime(ff, "%Y-%m-%d")
```

Realizamos asignaciones de valores y usamos las funciones brindadas por el DataTime, las cuales cuentan con una notación O(1), y al ser todo constante podemos concluir que cuenta con O(1).

```
for movie in catalog['elements']:
    # se saca el status y la fecha de publicacion de cada pelicula
    status = movie['status']
    release_date = movie['release_date']
    # se compara si el estado de la pelicula es igual al requerido
    if status == estado:
        # se saca la fecha de publicacion de cada pelicula en tipo datetime
        fecha_publicacion = datetime.strptime(release_date, "%Y-%m-%d")
        # se compara si la fecha de publicacion de la pelicula esta en el rango de fechas dadas
        if fecha_i <= fecha_publicacion <= fecha_f:
            # se agrega la pelicula a la lista "pelis"
            pelis.append(movie)
            # se suma la duracion de las peliculas y se suma el conteo total de peliculas que cumplen con l
            duracion_movie = movie.get('runtime', 'Desconocido')
            if duracion_movie != 'Desconocido':
```

Hcemos un recorrido a catalog["elements"], por lo tanto la notación de este es O(n), y por dentro de este ciclo podemos ver que solamente se realizan comparaciones y asignaciones, por lo tanto es todo constante y tiende a una notación de O(1)

```
resultado = {
    'total_peliculas': total_pelis,
    'tiempo_promedio': tiempo_promedio,
    # la lista de diccionarios con los detalles de todas las peliculas que cumplen los criterios
    'peliculas': [{
        'fecha_publicacion': movie['release_date'],
        'titulo': movie['title'],
        'presupuesto': movie['budget'],
        'recaudado': movie['revenue'],
        'ganancia': movie['profit'],
        'duracion': movie['runtime'],
        'calificacion': movie['vote_average'],
        'idioma': movie['original_language']
    } for movie in pelis_resultado]
}
```

Vemos que se realizan asignaciones pero en el ultimo, vemos un ciclo for, más sin emabro vemos que no recorre todo "n", el cual tomamos como todos los elementos de catalog["elements"], por lo tanto es un O(k) donde k<n, y O(k)<o(n)

Conclusión: Vimos en la estructura más participación de ciclos y comparaciones, más sin embargo el for que recorre más distancia fue el inicial, ya que era de tamaño n, y de ahi se formó otro Array_list para formar una lista la cual sería un “subconjunto” de cat[…], por lo tanto O(n) > O(k).
En conclusión, la notación que define este requerimiento es O(n).

Requerimiento _6:

```
peliculas_filtradas = lt.new_list() #Creamos un Array list en él cual agregaremos las peliculas filtradas
diccionario = {} # Creamos un diccionario que usaremos posteriormente
suma_votos = 0 # Inicializamos una suma en 0 para promediar el voto de las peliculas
suma_tiempo = 0 #Inicializamos una suma en 0 para promediar el tiempo de duración de las peliculas
```

Podemos ver al inicio del código como en la mayoría de requerimientos que contamos con asignaciones, las cuales cuentan con una notación O(1).

```
for pelicula in catalog["elements"]:
    if pelicula["original_language"] == idioma:
        if int(ai) < datetime.strptime(pelicula['release_date'], '%Y-%m-%d').year and datetime.strptime(pel
            lt.add_last(peliculas_filtradas,pelicula)
```

Podemos ver un ciclo de for el cual recorre todo catalog["elements"], por lo cual es O(n), dentro del ciclo vemos condicionales, asignaciones y comparaciones los cuáles son constantes que tienen una notación O(1).

```
for pelicula in peliculas_filtradas["elements"]:
    if datetime.strptime(pelicula['release_date'], '%Y-%m-%d').year in diccionario:
        diccionario[datetime.strptime(pelicula['release_date'], '%Y-%m-%d').year].append(pelicula)
    else:
        diccionario[datetime.strptime(pelicula['release_date'], '%Y-%m-%d').year] = [pelicula]
    if float(pelicula["budget"]) == 0 or float(pelicula["revenue"]) == 0:
        ganancias = 0
    else:
        ganancias = float(pelicula["revenue"])-float(pelicula["budget"])
```

Del mismo modo que el bucle pasado, tiene dentro asignaciones y comparaciones las cuales son constantes, más sin emabrgo, esta tiene una notación O(m) ya que estamos recorriendo la lista filtrada a partir del catalog["elements"].

```
for ano in diccionario: #Tomamos los años que fueron previamente filtrados
    for i in range(0,len(diccionario[ano])): # Realizamos un recorrido por indice para revisar todas las pe
        mejor = float(diccionario[ano][0]["vote_average"]) #Hacemos el patron del mejor y del peor para hal
        mejor_nombre = diccionario[ano][0]["title"]
        peor = float(diccionario[ano][0]["vote_average"])
        peor_nombre = diccionario[ano][0]["title"]
```

Aquí podemos ver un ciclo anidado a otro, por lo tanto debemos revisar que recorre cada uno para ver cual es su notación en O. El primer ciclo recorre una natidad de años, que es menor que m, pero como es anidado podemos ver que su notación es O(k**2), pero como recorre una cantidad m de películas, podemos decir que es O(m),

Conclusión: Podemos ver que nos encontramos una estructura distinta a la que habiamos visto en otros requerimientos, ya que era un bucle anidado por dos for, más sin embargo por el tamaño de estos podemos decir que fueron superados por la notación de O(n).
En conclusión, la notación en O del requerimiento 6 es O(n).

Requerimiento _7:

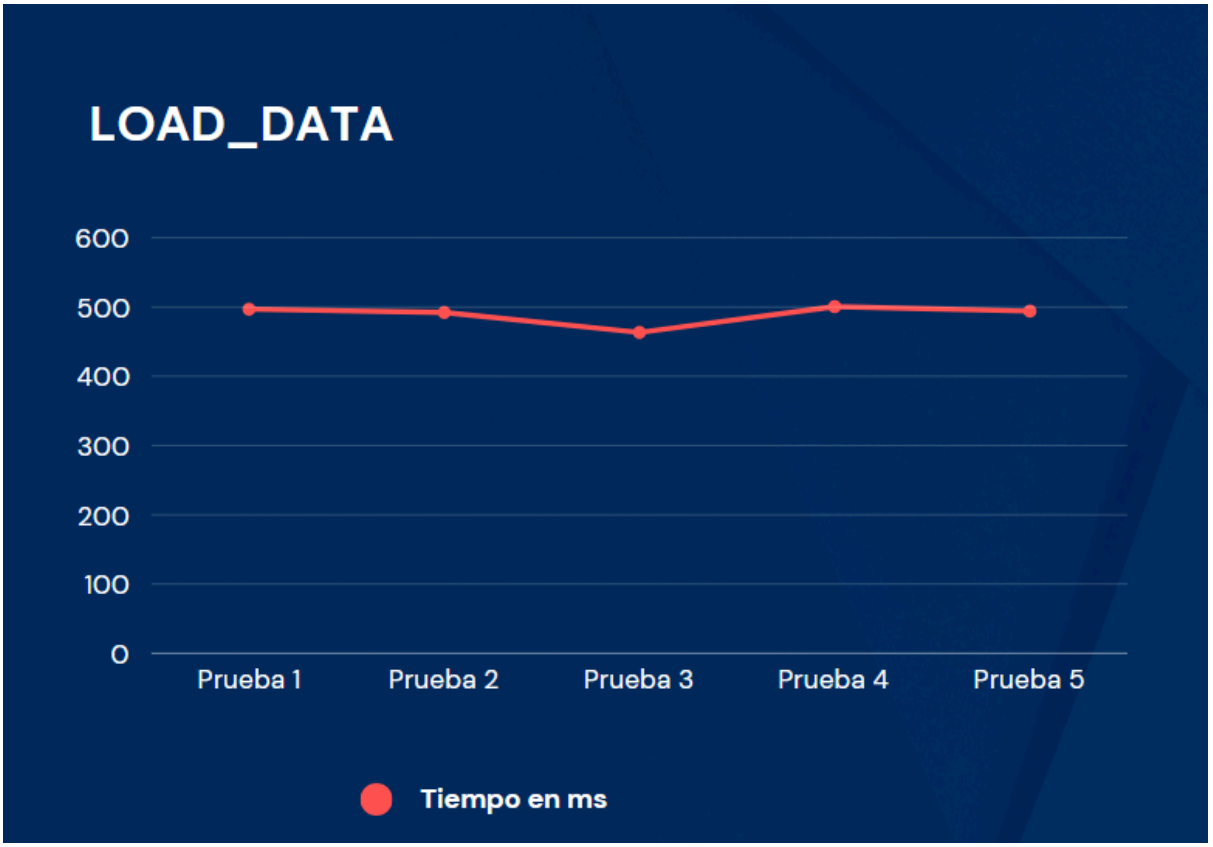
En el requerimiento 7 podemos observar que todos los bucles for tienden a tener una notación O(n), por lo tanto, la notación de éste será O(n).

Pruebas de tiempo:

En todas las pruebas se utilizo el archivo movies-large-csv
Las pruebas se hicieron en un portatil con i5 12450H y 32GB de ram

Funciones	**Tiempo promedio de ejecución en milisegundos**
load_data	489.4
req_1	910.6
req_3	1841.6
req_4	1705.6
req_6	2164.4
req_7	1453.2

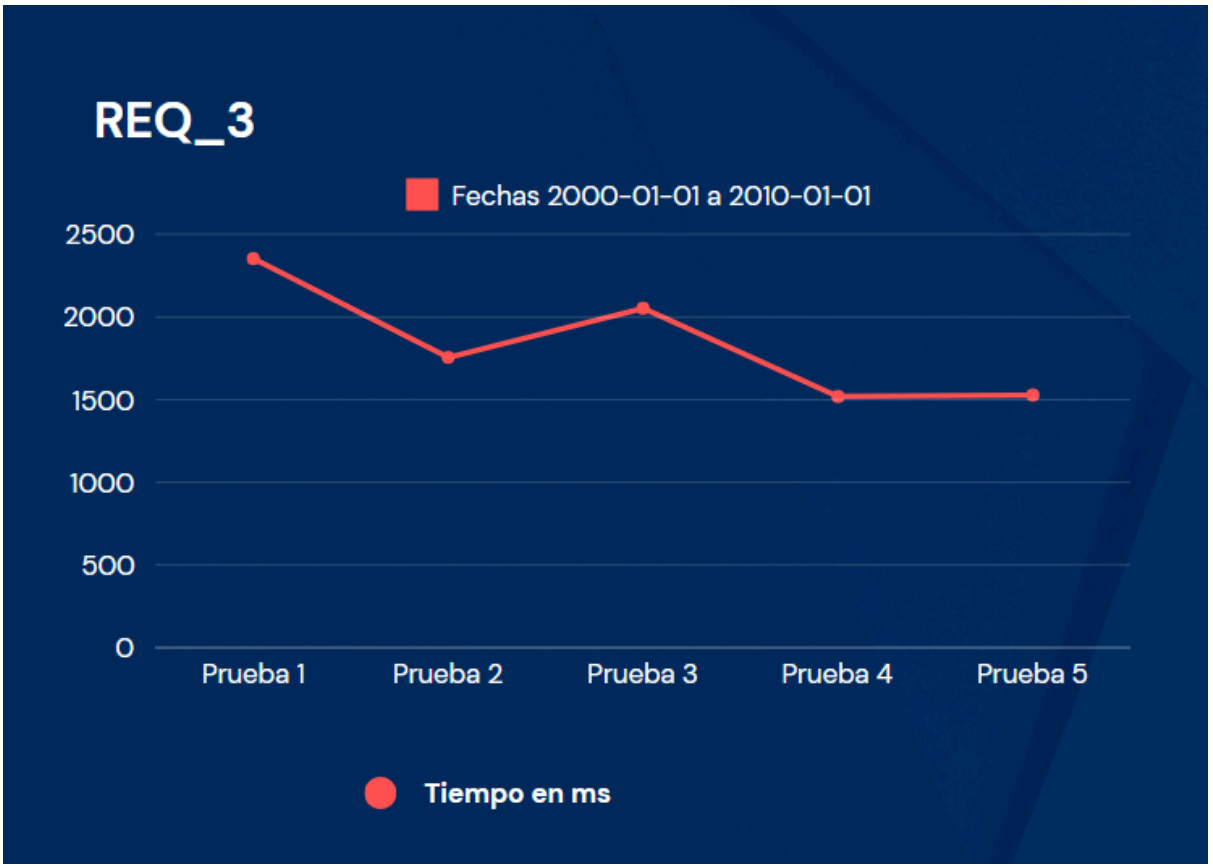
Load_data


LOAD_DATA
600
500
400
300
200
100
0
Prueba 1
Prueba 2
Prueba 3
Prueba 4
Prueba 5
Tiempo en ms

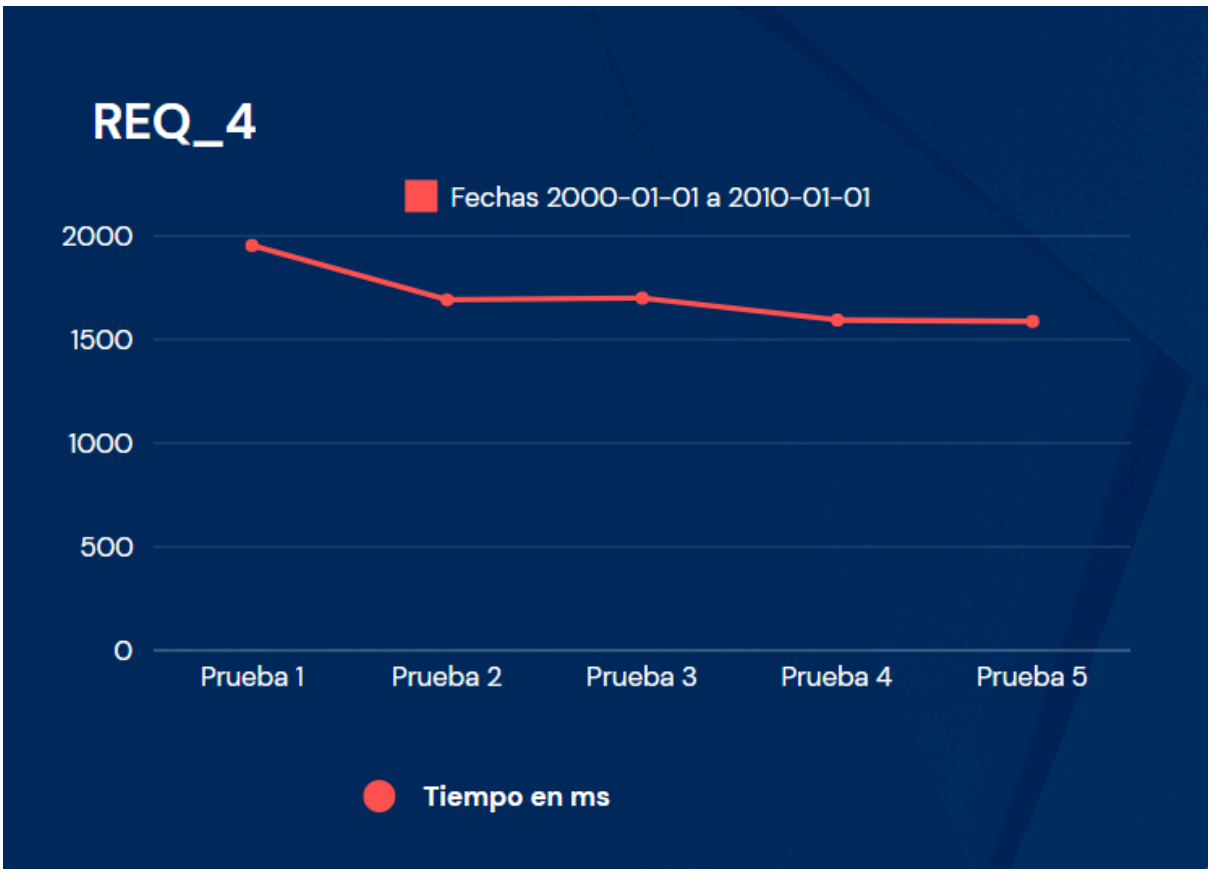
Req_1


REQ_1
Tiempo minimo 90
1200
1000
800
600
400
200
0
Prueba 1
Prueba 2
Prueba 3
Prueba 4
Prueba 5
Tiempo en ms

Req_3


REQ_3
Fechas 2000-01-01 a 2010-01-01
2500
2000
1500
1000
500
0
Prueba 1
Prueba 2
Prueba 3
Prueba 4
Prueba 5
Tiempo en ms

Req_4


REQ_4
Fechas 2000-01-01 a 2010-01-01
2000
1500
1000
500
0
Prueba 1
Prueba 2
Prueba 3
Prueba 4
Prueba 5
Tiempo en ms

Req_6


REQ_6
Fechas 2000 a 2010
3500
3000
2500
2000
1500
1000
500
0
Prueba 1
Prueba 2
Prueba 3
Prueba 4
Prueba 5
Tiempo en ms

Req_7


REQ_7
Fechas 2000 a 2010
2000
1500
1000
500
0
Prueba 1
Prueba 2
Prueba 3
Prueba 4
Prueba 5
Tiempo en ms