

## 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:

```
28 | with open(filename, encoding='utf-8') as file:
29 |     movies_file = csv.DictReader(file)

Podemos ver que en esta parte del código se abre el archivo, por lo cual es una acción constante: O(1)

31 |     for movie in movies_file:

Se recorre una lista ( Movies_files), suponiendo que tiene n elementos podemos decir que es O(n)

32 |         # se extraen todos los datos de cada película
33 |         id = movie.get('id', 'Desconocido')
34 |         title = movie.get('title', 'Desconocido')
35 |         org_language = movie.get('original_language', 'Desconocido')
36 |         release_date = movie.get('release_date', 'Desconocido')
37 |         revenue = movie.get('revenue', 'Desconocido')
38 |         runtime = movie.get('runtime', 'Desconocido')
39 |         status = movie.get('status', 'Desconocido')
40 |         vote_avg = movie.get('vote_average', 'Desconocido')
41 |         vote_count = movie.get('vote_count', 'Desconocido')
42 |         budget = movie.get('budget', 'Desconocido')
43 |         profit = float(revenue) - float(budget)
44 |         genre = json.loads(movie.get('genres', 'Desconocido'))
45 |         companies = json.loads(movie.get('production_companies', 'Desconocido'))
46 |         # se crea un diccionario para cada película
47 |         movie_info = {

Se asignan valores a las llaves que hemos creado, y las funciones get, float, y json son de O(1), por lo cual podemos decir que es una constante con O(1)

63 |         lt.add_last(catalog, movie_info)

Se agrega la al catalogo la información de la película con la funcion add-last, pero esta tiene complejidad de O(1) porque es una asignación constante, por lo tanto la complejidad total es O(1)
```

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 | películas_filtradas=lt.new_list()
80 | #Creamos un Array_list, el cual se usara para agregar las películas "filtradas"
Creamos un array_list, y como dijimos anteriormente esta función es  $O(1)$ , y la asignación también es constante por lo tanto sigue siendo  $O(1)$ 

películas = catalog["elements"]
for película in películas:
    if float(película["runtime"]) > tm:
        lt.add_last(películas_filtradas, película)

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

películas_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 función por dividir este tipo de listas  $n$  veces, hasta ordenarlas, por lo tanto cuenta con notación  $O(n \cdot \log n)$ .

if float(películas_filtradas["elements"][0]["revenue"]) == 0 or float(películas_filtr
| ganancias = 0
else:
| ganancias = float(películas_filtradas["elements"][0]["revenue"]) - float(película

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

película_mas_reciente = {"Tiempo de duración": películas_filtradas["elements"][0]["ru
| "Fecha de publicación de la película" : películas_filtradas["
| "Título original de la película" : películas_filtradas["elem
| "Presupuesto destinado a la realización de la película" : pe
| "Dinero recaudado neto por la película" : películas_filtrada
| "Ganancia final de una película" : ganancias,
| "Puntaje de calificación de la película" : películas_filtrad
| "Idioma original de publicación" : películas_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 DateTime para organizar fechas, nos damos cuenta que esta cuenta con una notación  $O(n \cdot \log n)$  siendo esta la mayor de todas. Por lo tanto, la notación del requerimiento \_1 es  $O(n \cdot \log n)$

### Requerimiento \_3:

```
suma = 0
fi_dt = datetime.strptime(fi, "%Y-%m-%d")
ff_dt = datetime.strptime(ff, "%Y-%m-%d")
```

Hacemos una asignación y realizamos una conversión con las funciones implementadas por la biblioteca Datetime, por lo tanto son acciones constantes que tienen notación  $O(1)$ .

```
películas_filtradas = lt.new_list()
películas_filtradas_con_formato = lt.new_list()
```

Creamos dos `array_list` que como hemos visto anteriormente cuentan con una notación  $O(1)$  ya que son asignaciones.

```
for película in catalog["elements"]:
    if película["original_language"] == idioma:
        if fi_dt <= datetime.strptime(película["release_date"], "%Y-%m-%d") and datetime.strptime(película["release_date"], "%Y-%m-%d") <= ff_dt:
            lt.add_last(películas_filtradas, película)
```

Realizamos un `for...` que tiene como notación  $O(n)$  y dentro de él podemos ver comparaciones y asignaciones constantes que tienen una notación de  $O(1)$ .

```
for a in películas_filtradas["elements"]:
    if float(a["budget"]) == 0 or float(a["revenue"]) == 0:
        ganancias = 0
    else:
        ganancias = float(a["revenue"]) - float(a["budget"])

    movie = {"Fecha de publicación de la película": a["release_date"],
            "Título original de la película": a["title"],
            "Presupuesto destinado a la realización de la película": a["budget"],
            "Dinero recaudado por la película": a["revenue"],
            "Ganancia de final de la película": ganancias,
            "Tiempo de duración en minutos de la película": a["runtime"],
            "Género de la película": a["genre"]}
    lt.add_last(películas_filtradas_con_formato, movie)
```

Como en el paso anterior, tenemos un `for` con demasiadas asignaciones y comparaciones, pero esto no va más allá de  $O(n)$  ya que estas asignaciones terminan siendo constantes.

```
promedio = suma / películas_filtradas_con_formato["size"]

# Realizamos el promedio del tiempo, suma estaba dentro del for e iba sumando los tiempos de las películas

if películas_filtradas_con_formato["size"] > 20:
    p_iniciales = lt.sub_list(películas_filtradas_con_formato, 1, 5)
    p_finales = lt.sub_list(películas_filtradas_con_formato, películas_filtradas_con_formato["size"] - 5, películas_filtradas_con_formato["size"])
else:
    p_iniciales = películas_filtradas_con_formato
    p_finales = películas_filtradas_con_formato
```

Por último, tenemos más asignaciones y comparaciones con condicionales que son constantes, por lo tanto su notación es  $O(1)$ .

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 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 objetos tipo datetime 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 DateTime, 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 i
            duracion_movie = movie.get('runtime', 'Desconocido')
            if duracion_movie != 'Desconocido':
```

Hicemos 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 embargo 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 ahí 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:

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

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(películas_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 películas_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 embargo, 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 índice 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 habíamos 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



Req\_1



Req\_3



Req\_4

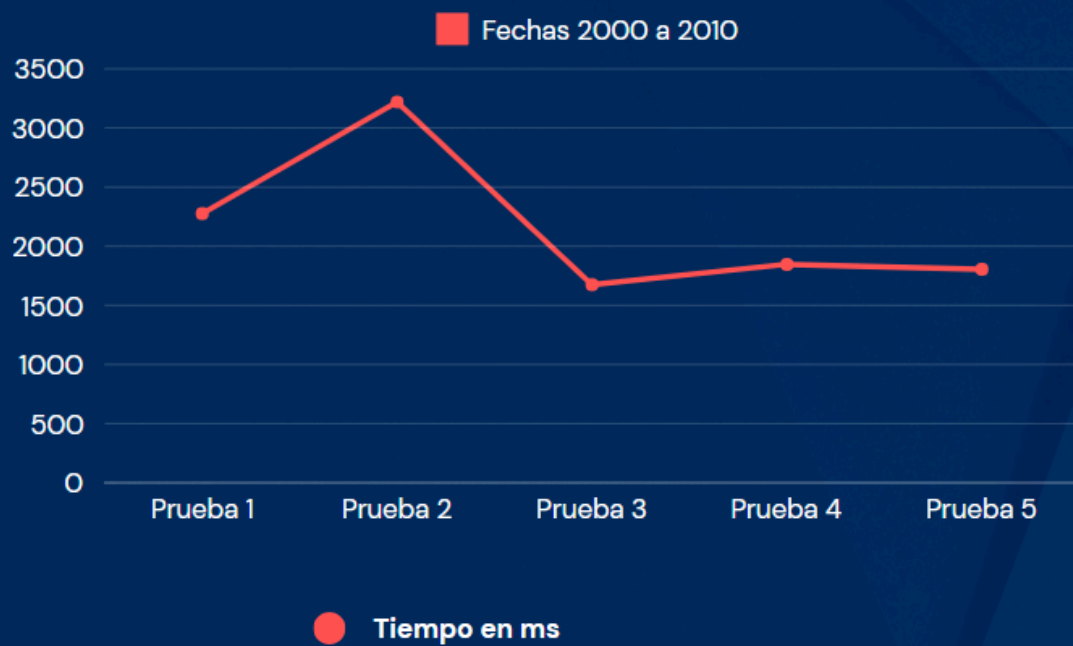


## REQ\_4



Req\_6

## REQ\_6



Req\_7

## REQ\_7

