Análisis Reto 2

- **Nicolas Contreras 202210717**
- **Alejando Pineda 202216447**
- **Joban Mejia 202213845**

Requerimiento 1

Análisis de complejidad

```
def año_estreno(catalogo,año_consulta):
    mapa_año=catalogo["release_year"]
    entry=mp.get(mapa_año,año_consulta)
    año_solicitado=me.getValue(entry)
    lista_peli=lt.newList()
    for video in lt.iterator(año_solicitado):
        if video["type"]=="Movie":
            lt.addLast(lista_peli,video)
    cantidad=lt.size(lista_peli)
    return lista_peli, cantidad
```

```
def orden_titulo_duracion(lista):
    mrg.sort(lista,condicion_orden_titulo_duracion)
```

Para el requerimiento 1 se tiene una complejidad **O**(n logn) dado a que aunque para el caso de la función que filtra la información se tenga solo un ciclo, la lista luego es ordenada mediante un merge sort que tiene complejidad **O**(n logn).

Pruebas

- **Tablas**

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	39.30	11.71
0.5	38.17	6.09
0.7	32.96	5.82

0.9	38.17	7.87

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	37.75	4.70
4.00	37.54	6.53
6.00	36.89	6.85
8.00	36.62	7.41

- **Comparación reto 1 tiempos**

Con respecto a el tiempo promedio obtenido en el reto1 el cual fue de 726 ms para large, hubo una mejora significativa la cual se dio principalmente porque al tener un mapa que filtra por el año de estreno la función solo debe invocar la lista asociada a este año y recorrer los video correspondientes a ese año.

Comparación reto 1 (Complejidad)

Con respecto al reto 1 esta función tiene la misma complejidad dado a que ambas hacen uso de un merge sort para ordenar los datos.

Requerimiento 2

Análisis de complejidad

```
def fecha(catalogo,fecha_consulta):
    mapa_fecha=catalogo["date_added"]
    entry=mp.get(mapa_fecha,fecha_consulta)
    fecha_solicita=me.getValue(entry)
    lista_tv=lt.newList()
    for video in lt.iterator(fecha_solicita):
        if video["type"]!="Movie":
            lt.addLast(lista_tv,video)
    cantidad=lt.size(lista_tv)
    return lista_tv, cantidad
```

```
def orden_titulo_duracion(lista):
    mrg.sort(lista,condicion_orden_titulo_duracion)
```

Para el requerimiento 2 se tiene una complejidad (n logn) ya que la función que filtra los datos solo tiene un ciclo for y la lista se ordena mediante un merge el cual tiene una complejidad de (n logn).

pruebas

- **Tablas**

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	41.11	8.49
0.5	30.66	7.24
0.7	30.54	6.40
0.9	30.66	6.25

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	30.54	6.63
4.00	30.98	6.35
6.00	31.52	6.57
8.00	30.78	6.65

Comparación reto 1 tiempos

Con respecto a el tiempo promedio obtenido en el reto1 el cual fue de 2543 ms para large, los tiempos de espera decrementaron significativamente esto se da la búsqueda de manera más rápida, gracias a la implementación de los árboles y también del código implementado.

Comparación reto 1 (Complejidad)

La complejidad esperada es la misma que en el requerimiento ya que implementa prácticamente el mismo código así que la complejidad de ambos de O (n logn)

Requerimiento 3 (Joban Mejia)

Análisis de complejidad

```python
def actor(catalogo, actor_consulta):
    mapa_actor = catalogo["cast"]
    programas = 0
    peliculas = 0
    entry = mp.get(mapa_actor,actor_consulta)
    actor_solicitado = me.getValue(entry)
    listado = lt.newList()
    for video in lt.iterator(actor_solicitado):
        if actor_consulta in video["cast"]:
            lt.addFirst(listado,video)
            if video["type"] == "Movie":
                peliculas += 1
            else:
                programas += 1
    lista_de_cantidades=lt.newList()
    lt.addLast(lista_de_cantidades,{"type":"Movies","count: ":peliculas})
    lt.addLast(lista_de_cantidades,{"type":"Tv Show","count: ":programas})
    return  lista_de_cantidades, listado
```

```python
def orden_actor(lista):
    mrg.sort(lista, condicion_actor)
```

Para el requerimiento 3 se tiene una complejidad **O**(n logn) ya que la función que filtra los datos solo tiene un ciclo for y la lista se ordena mediante un merge el cual tiene una complejidad **de O**(n logn).

Pruebas

- **Tablas**

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	6.01	1.7509
0.5	6.01	1.8518
0.7	6.01	2.0148
0.9	6.82	1.6904

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	5.39	1.8853
4.00	4.57	1.1200
6.00	5.39	0.7232
8.00	4.57	0.5374

Comparación reto 1 tiempos

Con respecto a el tiempo promedio obtenido en el reto1 el cual fue de 16 ms para large, los tiempos de espera decrementaron significativamente esto se da la búsqueda de manera más rápida, gracias a la implementación de los árboles.

Comparación reto 1 (Complejidad)

La complejidad esperada es la misma que en el requerimiento ya que implementa prácticamente el mismo código así que la complejidad de ambos de O (n logn)

<u>Requerimiento 4 (Nicolas Contreras)</u>

Análisis complejidad

```
def genero(catalogo,genero_consulta):
    mapa_genero=catalogo["listed_in"]
    generos_en_mapa=mp.keySet(mapa_genero)
    lista_generos_consultados=lt.newList()
    for genero in lt.iterator(generos_en_mapa):
        if genero_consulta in genero:
            lt.addLast(lista_generos_consultados,genero)
    lista_retorno=lt.newList()
    c_m=0
    c_tv=0
    for genero in lt.iterator(lista_generos_consultados):
        entry=mp.get(mapa_genero,genero)
        lista_genero=me.getValue(entry)
        for video in lt.iterator(lista_genero):
            lt.addLast(lista_retorno,video)
            if video["type"]=="Movie":
                c_m+=1
            else:
                c_tv+=1
    d_cantidad = {}
    d_cantidad["Movies"]=c_m
    d_cantidad["Tv_shows"]=c_tv
    return lista_retorno, d_cantidad
```

Para este requerimiento se tiene una complejidad O(n ^2) dado a que hay dos ciclos anidados que recorren los géneros solicitados

Pruebas

- **Tablas**

<u>PROBING</u>

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	1820.39	13494.68
0.5	1820.11	13878.06
0.7	1823.66	12363.72
0.9	1820.43	12613.34

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	1821.24	12350.89
4.00	1822.58	12489.65
6.00	1821.15	12588.98
8.00	1821.57	12675.26

- **Comparación reto 1 tiempos**

Con respecto a el tiempo promedio obtenido en el reto1 el cual fue de 34 ms para large, los tiempos de espera incrementaron significativamente esto se da porque en para el caso de los géneros hay diferentes formas de representar un mismo género por lo que se debe primero filtrar en el mapa de géneros que llaves corresponden al género solicitado, esto implica una complejidad mayor y por tanto un tiempo mayor de espera.

Comparación reto 1 (Complejidad)

Con respecto al reto 1 esta función tiene una complejidad mayor, para el caso del reto 1 esta función teína una complejidad O(nlogn) heredada del orden que se usaba mientras que para este caso se tiene una complejidad O(n ^2) que resulta de la forma de filtrar los géneros.

Requerimiento 5 (Alejandro Pineda)

Análisis Complejidad

```python
def pais(catalogo,pais_consulta):
    lista=lt.newList()
    cantidad={}
    movies=0
    tv_show=0
    mapa_pais=catalogo["country"]
    paises_en_mapa=mp.keySet(mapa_pais)
    lista_pais_consultados=lt.newList()
    for pais in lt.iterator(paises_en_mapa):
        if pais_consulta in pais:
            lt.addLast(lista_pais_consultados,pais)
    for pais in lt.iterator(lista_pais_consultados):
        lista_pais=me.getValue(mp.get(mapa_pais,pais))
        for video in lt.iterator(lista_pais):
            lt.addLast(lista,video)
            if video["type"]=="Movie":
                movies+=1
            else:
                tv_show+=1
    cantidad["Movies"]=movies
    cantidad["Tv_shows"]=tv_show
    return lista, cantidad
```

Esta función tiene una complejidad de O(n^2) dado que tiene dos for anidados que recorren todo el catálogo.

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	86.74	30.49
0.5	86.58	31.74
0.7	96.47	30.55
0.9	86.58	25.07

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	86.74	26.34
4.00	86.89	26.89
6.00	86.47	27.99
8.00	86.79	27.98

- **Comparación reto 1 tiempos**

En cuanto los tiempos dados en el reto 1(17 ms), podemos observar que hubo un incremento debido a que la forma en como iteramos se debía de recorrer el catálogo entero para recolectar

los datos que cumplieran con la condición país. Es decir, primero se recorren los valores de un map y luego las entradas de las listas.

Comparación reto 1 (Complejidad)

En comparación con el reto 1, la complejidad paso de ser o(nlogn) a o(n^2), por lo cual podemos observar que aumento. Esto debido a la forma en como filtramos los datos de países en comparación al reto pasado.

Requerimiento 6

```
def director(catalogo,director_consulta):
    mapa_directores=catalogo["director"]
    entry=mp.get(mapa_directores,director_consulta)
    lista_director=me.getValue(entry)
    cant_por_tipo={}
    cant_por_plataforma={}
    cant_por_genero={}
    generos_presentes=lt.newList()
    for video in lt.iterator(lista_director):
        if video["type"] not in cant_por_tipo:
            cant_por_tipo[video["type"]]=1
        elif video ["type"] in cant_por_tipo:
            cant_por_tipo[video["type"]]+=1
        if video["stream_service"] not in cant_por_plataforma:
            cant_por_plataforma[video["stream_service"]]=1
        elif video["stream_service"] in cant_por_plataforma:
            cant_por_plataforma[video["stream_service"]]+=1
        generos=video["listed_in"].split(",")
        for genero in generos:
            if genero not in cant_por_genero.keys():
                cant_por_genero[genero]=1
                lt.addLast(generos_presentes,genero)
            else:
                cant_por_genero[genero]+=1
    lista_cant_genero=lt.newList()
    for genero in lt.iterator(generos_presentes):
        genero_como_elem={}
        genero_como_elem["listed_in"]=genero
        genero_como_elem["count"]=cant_por_genero[genero]
        lt.addLast(lista_cant_genero,genero_como_elem)
    return cant_por_tipo,cant_por_plataforma, lista_cant_genero, lista_director
```

Para este caso se tiene una complejidad de n ^2 por que se hace uso de dos for anidados

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	0.63	0.38
0.5	0.63	0.30
0.7	0.63	0.18
0.9	0.63	0.20

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	0.63	0.30
4.00	0.63	0.45
6.00	0.63	0.69
8.00	0.63	0.25

- **Comparación reto 1 tiempos**

  Con respecto a el tiempo promedio obtenido en el reto1 el cual fue de 18 ms para large, los tiempos de espera se redujeron significativamente dado al uso de maps y el filtrado en la carga de datos

Comparación reto 1 (Complejidad)

Con respecto al reto 1 esta función tiene la misma complejidad de O(n ^2)

Requerimiento 7

```python
def top_generos(catalogo):
    TAD_conteo = lt.newList()
    list_generos = []
    TAD_generos = lt.newList()
    mapa_genero=catalogo["listed_in"]
    mapa_peliculas = catalogo["stream_service"]
    ll_filmes = mp.valueSet(mapa_peliculas)
    generos = mp.keySet(mapa_genero)
    for generos_sep in lt.iterator(generos):
        for genero in generos_sep.split(", "):
            if genero not in list_generos:
                list_generos.append(genero)
    for pos in range(0,len(list_generos)):
        genero = list_generos[pos]
        if genero[0] == " ":
            genero1 = genero[1:]
            list_generos[pos] = genero1
    list_generos1 = []
    for genero in list_generos:
        if genero not in list_generos1:
            list_generos1.append(genero)
    conteo_generos = {}
    for genero in list_generos1:
        conteo_generos[genero] = 0
        lt.addLast(TAD_generos,genero)
    for genero in conteo_generos:
        for l_filmes_genero in lt.iterator(ll_filmes):
            for filmes in lt.iterator(l_filmes_genero):
                generos = filmes["listed_in"].split(", ")
                for genero1 in generos:
                    if genero == genero1:
                        conteo_generos[genero] += 1
    for genero in lt.iterator(TAD_generos):
        tabla_genero={}
        tabla_genero["listed_in"]=genero
        tabla_genero["count"]=conteo_generos[genero]
        lt.addLast(TAD_conteo,tabla_genero)
    return len(list_generos1), TAD_conteo
```

La complejidad del requerimiento 7 es de O(n^3logn) Pues a pesar de que hay 4 for anidados, uno de ellos ya se encuentra filtrado, por lo tanto, no recorre todo el catalogo, disminuyendo su complejidad temporal.

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	71.90	2220.19
0.5	71.50	2223.64
0.7	72.95	2211.11
0.9	71.35	2200.19

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	71.99	2213.20
4.00	71.56	2208.98
6.00	71.23	2190.56
8.00	71.87	2189.25

Este req no fue analizado el reto pasado, por lo tanto, no hay punto de comparación.

Requerimiento 8

Análisis complejidad

```
listas_bono(lista_top_n,nuevo_mapa_actores):
lista_cantidades=lt.newList()
lista_videos=lt.newList()
lista_colaboraciones=lt.newList()
for actor in lt.iterator(lista_top_n):
    nombre_actor=actor["cast"]
    entry=mp.get(nuevo_mapa_actores,nombre_actor)
    lista_actor=me.getValue(entry)
    elem_listas_cantidades={}
    elem_lista_videos={}
    elem_lista_colaboraciones={}
    elem_listas_cantidades["cast"]=nombre_actor
    elem_lista_videos["cast"]=nombre_actor
    elem_lista_colaboraciones["cast"]=nombre_actor
    count_type={}
    lista_peliculas=lt.newList()
    lista_tv=lt.newList()
    lista_directores=lt.newList()
    lista_actores=lt.newList()
    for video in lt.iterator(lista_actor):
        if video["type"]=="Movie":
                lt.addLast(lista_peliculas,video)
        if video["type"]!="Movie":
            lt.addLast(lista_tv,video)
        if video["stream_service"] not in count_type:
            count_type[video["stream_service"]]={video["type"]:1}
        elif video["type"] not in count_type[video["stream_service"]]:
            dict_asociado=count_type[video["stream_service"]]
            dict_asociado[video["type"]]=1
        else:
            dict_asociado=count_type[video["stream_service"]]
            dict_asociado[video["type"]]+=1


        directores=video["director"].split(",")
        for director in directores:
            if not lt.isPresent(lista_directores,director):
                lt.addLast(lista_directores,director)
        actores_colab=video["cast"].split(",")
        for actor_colab in actores_colab:
            if not lt.isPresent(lista_actores,actor_colab):
                lt.addLast(lista_actores,actor_colab)
```

Para el bono se tiene una complejidad de O(n ^3) dado a que hay 3 for anidados y se recorren completamente

PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	79503.14	6830.71

0.5	79503.09	6714.97
0.7	79508.94	6742.29
0.9	79503.64	6659.83

CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	79479.23	6384.01
4.00	79479.56	6495.85
6.00	79479.68	6618.65
8.00	79479.42	6773.65