Integrantes:


```
Esteban Leiva - 202021368 - e.leivam@uniandes.edu.co
Michelle Vargas - 201914771 - bm.vargas@uniandes.edu.co
```


Máquina usada:

Procesadores	**Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz**
Memoria RAM (GB)	16.0 GB (15.8 GB usable)
Sistema Operativo	Windows 10 pro

Análisis de complejidad, tiempo y memoria:

REQ 1:

```
def req1(catalog,country,category,n):
    lista=model.getvidsby(catalog,'countries',country)
    ide=model.idporcategory(category,catalog) #1
    if lista==None or ide==None:
        print('\nNO HAY INFORMACIÓN PARA ESTE PAÍS O CATEGORÍA\n')
    else:
        mapa=model.countryid(lista,ide) #2
        i=1
        print('\nINFORMACIÓN DE LOS '+str(n)+' VIDEOS CON MÁS VIEWS EN '+country.
upper()+' PARA LA CATEGORÍA '+category.upper())
        while i<=n:
            x=model.dlv(catalog,mapa,'views') #3
            info=x[1]
            print ('\nPUESTO '+str(i)+'\ntrending_date: '+str(info['trending_date
'])+' || title: '+info['title']+' || channel_title: '+info['channel_title']+' ||
publish_time: '+info['publish_time']+' || views: '+str(x[2])+' || likes: '+info['
likes']+' || dislikes: '+info['dislikes'])
            mp.remove(mapa,x[0])
            i+=1
        print('\n')
```

1:

```
def idporcategory(name,catalog):
    """
    Devuelve el ID dado el nombre de una categoría
    """
    categorias=catalog['categories']
```

```
    i=1
    while i<=lt.size(categorias):
        c=lt.getElement(categorias,i)
        if name.lower() in (c['name']).lower():
            return c['id']
        i+=1
    if i==lt.size(categorias):
        return None
```

O(n) donde n es el tamaño de catalog[“categories”]

2:

```
def countryid(lista,ide):
    """
    Devuelve un mapa con videos de un category_id en particular, a partir de la l
ista de videos ya filtrada por pais, cuyas llaves son los títulos de los videos y
 cuyos valores son entradas de newtviews
    """
    mapa=mp.newMap(numelements=4096,
                   maptype="PROBING",
                   loadfactor=0.5)
    i=it.newIterator(lista)
    while it.hasNext(i):
        vid=it.next(i)
        if vid['category_id']==ide:
            tit = vid["title"]
            existit=mp.contains(mapa,tit)
            if existit:
                entry = mp.get(mapa,tit)
                value = me.getValue(entry)
                if vid["views"]>value["views"]:
                    value["views"] = vid["views"]
            else:
                entry=newtviews(vid['title'])
                mp.put(mapa,vid['title'],entry)
                entry['views']=vid['views']
                entry['info']=vid

    return mapa
```

O(n) donde n es el tamaño de la lista dada

3:

```
def dlv(catalog,mapa,dlv):
    """
    Busca el video con mayor número de dias/likes/views de un map con videos y re
torna una tupla con (título del video, información video, valor mayor)
    """
    info=None
    mayor=0
    llaves=mp.keySet(mapa)
    i=it.newIterator(llaves)
    while it.hasNext(i):
        llave=it.next(i)
        entry=mp.get(mapa,llave)
        value=me.getValue(entry)
        m=int(value[dlv])
        if m>mayor:
            mayor=m
            info=value['info']
    return info['title'],info,mayor
```

O(x*n) donde x es un parametró dado por el usuario que se puede considerar como pequeño (este x se debe al while que precede a #3) y n es el tamaño de la lista de llaves del mapa. Entonces, la complejidad se considera como O(n).

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 1 es O(N).

Comparando con el Reto 1, la complejidad mejoró porque con el ordenamiento de listas la complejidad del Requerimiento 1 es O(N*logN)

Tiempo y consumo de memoria:

	Tiempo [ms]	Memoria [kB]
RETO 1	681.184	82.730
RETO 2	239.879	2388.898

Como se esperaba, el tiempo de ejecución del Requerimiento 1 del Reto 2 es considerablemente menor que el del Reto 1, específicamente es aproximadamente 3 veces menor; lo cual puede ser explicado por la diferencia en las complejidades: O(N) vs O(NlogN). Con respecto a la memoria utilizada, la implementación del Reto 1 uso poca memoria comparado al Reto 2, lo cual es consecuencia de la creación de mapas en la segunda implementación.

REQ 2:

```
def req2(catalog,country,td):
    lista = model.getvidsby(catalog,'countries',country)
    if lista == None:
        return 'NO HAY INFORMACIÓN PARA ESTE PAÍS'
    else:
        mapa=model.titleporidc('dias',lista,td) #1
        x=model.dlv(catalog,mapa,'dias') #2
        info=x[1]
        return 'INFORMACIÓN DEL VIDEO TENDENCIA POR MÁS DÍAS EN '+country.upper()
+'\ntitle: '+info['title']+' || channel_title: '+info['channel_title']+' || count
ry: '+info['country']+' || días: '+str(x[2])
```

1:

```
def titleporidc(parametro,lista,titoid):
    """
    Recibe una lista de videos y devuelve un mapa cuyas llaves son los nombres de
 los videos y cuyo valor es una entrada de la función newtlikes o newtdias
    """
    mapa=mp.newMap(numelements=65536,
                   maptype="PROBING",
                   loadfactor=0.5)
    i=it.newIterator(lista)
    while it.hasNext(i):
        vid=it.next(i)
        td=vid[titoid]
        existit=mp.contains(mapa,td)
        if existit:
            entry=mp.get(mapa,td)
            value=me.getValue(entry)
            if parametro=='likes':
                if vid['likes']>value['likes']:
                    value['likes']=vid['likes']
            elif parametro=='dias':
                value['dias']+=1
        else:
            if parametro=='likes':
                value=newtlikes(td)
                mp.put(mapa,td,value)
```

```
                value['info']=vid
                value['likes']=vid['likes']
            elif parametro=='dias':
                value=newtdias(td)
                mp.put(mapa,td,value)
                value['info']=vid
                value['dias']+=1

    return mapa
```

O(n) donde n es el tamaño de la lista dada

2:

```
def dlv(catalog,mapa,dlv):
    """
    Busca el video con mayor número de dias/likes/views de un map con videos y re
torna una tupla con (título del video, información video, valor mayor)
    """
    info=None
    mayor=0
    llaves=mp.keySet(mapa)
    i=it.newIterator(llaves)
    while it.hasNext(i):
        llave=it.next(i)
        entry=mp.get(mapa,llave)
        value=me.getValue(entry)
        m=int(value[dlv])
        if m>mayor:
            mayor=m
            info=value['info']
    return info['title'],info,mayor
```

O(n) donde n es el tamaño de la lista de llaves del mapa.

Conclusión:

Teniendo en cuenta lo anterior, la complejidad total del Requerimiento 2 es O(N).

Comparando con el Reto 1, la complejidad mejoró porque con el ordenamiento de listas la complejidad del Requerimiento 2 está entre O(Nlog(N)) y O(N^2)

Tiempo y consumo de memoria:

	Tiempo [ms]	Memoria [kB]
RETO 1	2918.428	717.352
RETO 2	1437.810	37599.402

Como se esperaba, el tiempo de ejecución del Requerimiento 2 del Reto 2 es considerablemente menor que el del Reto 1, específicamente es aproximadamente 2 veces menor; lo cual puede ser explicado por la diferencia en las complejidades: O(N) vs O(NlogN)-O(N^2). Con respecto a la memoria utilizada, la implementación del Reto 1 uso poca memoria comparado al Reto 2, lo cual es consecuencia de la creación de mapas en la segunda implementación.

REQ 3:

```
def req3(catalog,category,td):
    ide=model.idporcategory(category,catalog) #1
    if ide==None:
        return 'NO HAY INFORMACIÓN PARA ESTA CATEGORÍA'
    else:
        lista = model.getvidsby(catalog,'ids',ide)
        mapa=model.titleporidc('dias',lista,td) #2
        x=model.dlv(catalog,mapa,'dias') #3
        info=x[1]
        return 'INFORMACIÓN DEL VIDEO TENDENCIA POR MÁS DÍAS PARA LA CATEGORÍA '+
category.upper()+'\ntitle: '+info['title']+' || channel_title: '+info['channel_ti
tle']+' || category_id: '+str(info['category_id'])+' || días: '+str(x[2])
```

1:

```
def idporcategory(name,catalog):
    """
    Devuelve el ID dado el nombre de una categoría
    """
    categorias=catalog['categories']
    i=1
    while i<=lt.size(categorias):
        c=lt.getElement(categorias,i)
        if name.lower() in (c['name']).lower():
            return c['id']
        i+=1
```

```
    if i==lt.size(categorias):
        return None
```

O(n) donde n es el tamaño de catalog[“categories”]

2:

```
def titleporidc(parametro,lista,titoid):
    """
    Recibe una lista de videos y devuelve un mapa cuyas llaves son los nombres de
 los videos y cuyo valor es una entrada de la función newtlikes o newtdias
    """
    mapa=mp.newMap(numelements=65536,
                   maptype="PROBING",
                   loadfactor=0.5)
    i=it.newIterator(lista)
    while it.hasNext(i):
        vid=it.next(i)
        td=vid[titoid]
        existit=mp.contains(mapa,td)
        if existit:
            entry=mp.get(mapa,td)
            value=me.getValue(entry)
            if parametro=='likes':
                if vid['likes']>value['likes']:
                    value['likes']=vid['likes']
            elif parametro=='dias':
                value['dias']+=1
        else:
            if parametro=='likes':
                value=newtlikes(td)
                mp.put(mapa,td,value)
                value['info']=vid
                value['likes']=vid['likes']
            elif parametro=='dias':
                value=newtdias(td)
                mp.put(mapa,td,value)
                value['info']=vid
                value['dias']+=1

    return mapa
```

O(n) donde n es el tamaño de la lista dada.

3:

```
def dlv(catalog,mapa,dlv):
    """
    Busca el video con mayor número de dias/likes/views de un map con videos y re
torna una tupla con (título del video, información video, valor mayor)
    """
    info=None
    mayor=0
    llaves=mp.keySet(mapa)
    i=it.newIterator(llaves)
    while it.hasNext(i):
        llave=it.next(i)
        entry=mp.get(mapa,llave)
        value=me.getValue(entry)
        m=int(value[dlv])
        if m>mayor:
            mayor=m
            info=value['info']
    return info['title'],info,mayor
```

O(n) donde n es el tamaño de la lista de llaves del mapa.

Conclusión:

Teniendo en cuenta lo anterior, la complejidad total del Requerimiento 3 es O(N).

Comparando con el Reto 1, la complejidad mejoró porque con el ordenamiento de listas la complejidad del Requerimiento 3 está entre O(Nlog(N)) y O(N^2)

Tiempo y consumo de memoria:

	Tiempo [ms]	Memoria [kB]
RETO 1	3636.009	709.250
RETO 2	1294.207	34365.246

Como se esperaba, el tiempo de ejecución del Requerimiento 3 del Reto 2 es considerablemente menor que el del Reto 1, específicamente es aproximadamente 3 veces menor; lo cual puede ser explicado por la diferencia en las complejidades: O(N) vs O(NlogN)- O(N^2). Con respecto a la memoria utilizada, la implementación del Reto 1 uso poca memoria comparado al Reto 2, lo cual es consecuencia de la creación de mapas en la segunda implementación.

REQ 4:

```
def req4(catalog,country,tag,n):
    lista=model.getvidsby(catalog,'countries',country)
    if lista==None:
        print('\nNO HAY INFORMACIÓN PARA ESTE PAÍS\n')
    else:
        lista2=model.tags(catalog,lista,tag) #1
        if lt.size(lista2)==0:
            print('\nNO HAY INFORMACIÓN PARA ESTE TAG\n')
        else:
            mapa=model.titleporidc('likes',lista2,'title') #2
            i=1
            print('\nINFORMACIÓN DE LOS '+str(n)+' VIDEOS CON MÁS LIKES EN '+coun
try.upper()+' CON EL TAG '+tag.upper())
            while i<=n:
                x=model.dlv(catalog,mapa,'likes') #3
                info=x[1]
                print('\nPUESTO '+str(i)+'\ntitle: '+info['title']+' || channel_t
itle: '+info['channel_title']+' || publish_time: '+info['publish_time']+' || view
s: '+info['views']+' || likes: '+str(x[2])+' || dislikes: '+info['dislikes']+'\nt
ags: '+info['tags'])
                print(info['video_id'])
                mp.remove(mapa,x[0])
                i+=1
            print('\n')
```

1:

```
def tags(catalog,lista,tag):
    """
    Dada una lista de videos retorna otra con los videos que tienen un tag dado
    """
    final=lt.newList(datastructure='ARRAY_LIST')
    i=it.newIterator(lista)
    while it.hasNext(i):
        vid=it.next(i)
        if tag in vid['tags']:
            lt.addLast(final,vid)
    return final
```

O(n) donde n es el tamaño de la lista dada.

2:

```
def titleporidc(parametro,lista,titoid):
    """
    Recibe una lista de videos y devuelve un mapa cuyas llaves son los nombres de
 los videos y cuyo valor es una entrada de la función newtlikes o newtdias
    """
    mapa=mp.newMap(numelements=65536,
                   maptype="PROBING",
                   loadfactor=0.5)
    i=it.newIterator(lista)
    while it.hasNext(i):
        vid=it.next(i)
        td=vid[titoid]
        existit=mp.contains(mapa,td)
        if existit:
            entry=mp.get(mapa,td)
            value=me.getValue(entry)
            if parametro=='likes':
                if vid['likes']>value['likes']:
                    value['likes']=vid['likes']
            elif parametro=='dias':
                value['dias']+=1
        else:
            if parametro=='likes':
                value=newtlikes(td)
                mp.put(mapa,td,value)
                value['info']=vid
                value['likes']=vid['likes']
            elif parametro=='dias':
                value=newtdias(td)
                mp.put(mapa,td,value)
                value['info']=vid
                value['dias']+=1

    return mapa
```

O(n) donde n es el tamaño de la lista dada.

3:

```
def dlv(catalog,mapa,dlv):
    """
```

```
    Busca el video con mayor número de dias/likes/views de un map con videos y re
torna una tupla con (título del video, información video, valor mayor)
    """
    info=None
    mayor=0
    llaves=mp.keySet(mapa)
    i=it.newIterator(llaves)
    while it.hasNext(i):
        llave=it.next(i)
        entry=mp.get(mapa,llave)
        value=me.getValue(entry)
        m=int(value[dlv])
        if m>mayor:
            mayor=m
            info=value['info']
    return info['title'],info,mayor
```

O(x*n) donde x es un parametró dado por el usuario que se puede considerar como pequeño (este x se debe al while que precede a #3) y n es el tamaño de la lista de llaves del mapa. Entonces, la complejidad se considera como O(n).

Conclusión:

Teniendo en cuenta lo anterior, la complejidad del Requerimiento 4 es O(N).

Comparando con el Reto 1, la complejidad mejoró porque con el ordenamiento de listas la complejidad del Requerimiento 4 es O(N^2). Sin embargo, es importante notar que el N del REQ4 del Reto 1 es considerablemente pequeño, lo cual implica que la carga computacional no es tan grande.

Tiempo y consumo de memoria:

	Tiempo [ms]	Memoria [kB]
RETO 1	918.305	98.930
RETO 2	1131.843	32768.949

A pesar de las diferencias en la complejidad entre las dos implementaciones, la diferencia en los tiempos de ejecución es de menos de 200.000 ms. Esto se debe a que el N del Reto 1 no es muy grande y es menor al N del Reto 2, generando que la primera implementación se demore menos. Con respecto a la memoria utilizada, la implementación del Reto 1 uso poca memoria comparado al Reto 2, lo cual es consecuencia de la creación de mapas en la segunda implementación.