RETO 1: Análisis de Complejidad

Requerimiento 1:

```
def filtrar_PaisCategoria(catalog,country,category,size):
    videos = catalog['videos']
    sub_list = lt.newList(datastructure='ARRAY_LIST')
    for video in lt.iterator(videos):
        if video['country']==country and video['category_id']==category:
            al.addLast(sub_list,video)
    sorted_list= sortVideos(sub_list,cmpVideosByViews)
    lst_n = lt.subList(sorted_list,1,size)
    return filtrar_datos_req1(lst_n)

def filtrar_datos_req1(lst):
    lst_rta= lt.newList(datastructure='ARRAY_LIST')
    for video in lt.iterator(lst):
        video = {
            'trending_date': video['trending_date'],
            'title': video['title'],
            'publish_time': video['publish_time'],
            'views': int(video['views']),
            'likes': int(video['likes']),
            'dislikes': int(video["dislikes"]),
        }
        lt.addLast(lst_rta,video)
    return lst_rta
```

Total = O(n)

Como la función recorre n elementos, concluimos que su complejidad está dada por O(n)

Requerimiento 2:

```
def filtar_paisTendencia(catalog, country):
    videos = catalog['videos']
    sub_list =  lt.newList(datastructure= 'ARRAY_LIST')
    for video in lt.iterator(videos):
        if video['country'] == country:
            lt.addLast(sub_list,video)
    srt_list = sortVideos(sub_list,cmpVideosbyId)
    lst = extraer_ids(srt_list)
    id,m = id_mas_repetido(lst)
    return (video_por_id(srt_list,id,m))
```

= O(nLogn)

Este requerimiento posee una complejidad nLogn, ya que en la función de sortVideos se hace uso de merge, la cual tiene la complejidad anteriormente mencionada.

Requerimiento 3:

```
def video_mas_dias_tendencia(catalog,id_category):
    videos = catalog['videos']
    sub_list = lt.newList(datastructure='ARRAY_LIST')
    for video in lt.iterator(videos):
        if video['category_id']== id_category:
            lt.addLast(sub_list,video)

    srt_list= sortVideos(sub_list,cmpVideosbyId)
    lst_ids = extraer_idsReq3(srt_list)
    id_mayor, cant = id_mas_repetido(lst_ids)
    return video_por_id(sub_list,id_mayor,cant)

def extraer_ids(lst):
    sub_list = lt.newList(datastructure='ARRAY_LIST')
    for video in lt.iterator(lst):
        lt.addLast(sub_list,video['video_id'])
    return sub_list

def extraer_idsReq3(lst):
    sub_list = lt.newList(datastructure='ARRAY_LIST')
    for video in lt.iterator(lst):
        cmp_id= ''
        cmp_Tdate = ''
        cmp_country = ''
        if (cmp_id == video['video_id']) and (cmp_Tdate== video['trendin
            print(None)
        else:
            lt.addLast(sub_list,video['video_id'])
            cmp_id = video['video_id']
            cmp_Tdate = video['trending_date']
            cmp_country = video['country']
    return sub_list

def id_mas_repetido(lst):
    id_mayor = 0
    id_cmp = 0
    for video_id in lt.iterator(lst):
        id_cant = lst['elements'].count(video_id)
        if id_cant > id_cmp:
            id_cmp = id_cant
            id_mayor = video_id
    return (id_mayor,id_cmp)
```

```
def video_por_id(lst,id_video,freq_id):
    for video in lt.iterator(lst):
        if video['video_id'] == id_video:
            info_video ={
                'title': video['title'],
                'cannel_title': video['cannel_title'],
                'category_id': video['category_id'],
                'dias': freq_id
            }
            return info_video
```

Total = O(nlogn)

En este requerimiento aunque se usan varios ciclos que recorren n elementos, en videos_mas_dias_tendencia(), se hace uso del Tad merge sort, el cual tiene una complejidad de nlogn, por ende se concluye que este requerimiento tiene un complejidad igual.

Requerimiento 4:

```
'Requerimiento #4'
def videos_mas_likes(catalog,country,cant_videos,tag):
    videos = catalog['videos']
    sub_list = lt.newList(datastructure='ARRAY_LIST')
    for video in lt.iterator(videos):
        if (tag in video['tags']) and (video['country'] == country):
            lt.addLast(sub_list,video)
    srt_lst = sortVideos(sub_list,cmpVideosbyLikes)
    return lt.subList(srt_lst,1,cant_videos)
```

Total=O(nlogn)

Esta función de este requerimiento no se diferencia mucho de las funciones anteriores, ya que se usa merge sort, para ordenar los videos, podemos analizar, que posee una complejidad de O(NLogN)