ANÁLISIS RETO 2

David Leonardo Almanza Márquez – 202011293 – d.almanza@unaindes.edu.co
Laura Daniela Arias Flórez – 202020621 – l.ariasf@uniandes.edu.co

COMPLEJIDAD REQUERIMIENTO 1

```python
# Requerimiento 1
def bestCountryCategory(catalog, Acategory, Acountry):
    vids = getVidsByCountry(catalog, Acountry)
    #Obtiene la lista de videos correspondientes a un pais, a partir de un mapa: O(1)
    if vids:
        list_of_vids = lt.newList("ARRAY_LIST")
        for vid in lt.iterator(vids):
            if(vid["category_id"] == Acategory):
                lt.addLast(list_of_vids, vid)
        #recorre todos los videos en la lista de un cierto pais: O(m) donde m es el numero de videos en esta lista
        sorted_list = ma.sort(list_of_vids, compare_views)
        #uso de merge para ordenar la lista de videos de tamaño m: mlog(m)
        return sorted_list
    return None
```

Por la implementación de los iteradores diseñados para el ADT mismo (Uso de lt.iterator()) pensamos que el tiempo de ejecución de esta función y todas las que salen a continuación disminuyó bastante, además de pasar de usar más de un sort, más de un loop y más de una búsqueda binaria a usar solo un loop y un sort. En cuanto a memoria, es cierto que un mapa usa más memoria que una lista, pero hay que tener en cuenta que estos mapas no solo reducen los tiempos de búsqueda sustancialmente, sino que, además, después de creados se pueden usar para resolver una cantidad de posibles problemas que le puedan venir a la cabeza al usuario enorme.

COMPLEJIDAD REQUERIMIENTO 2

```python
# Requerimiento 2
def bestVidCountry(catalog, country):
    vids = getVidsByCountry(catalog, country)
    #Obtiene la lista de videos correspondientes a un pais, a partir de un mapa: O(1)
    sortedVids = sa.sort(vids, compareId)
    #uso de merge para ordenar la lista de videos de tamaño m: mlog(m)
    if(sortedVids):
        previousId = ""
        count = 1
        bestVid = ""
        MaxCount = 0
        for vid in lt.iterator(sortedVids):
            if(vid["video_id"] == previousId):
                count += 1
            else:
                count = 1
            previousId = vid["video_id"]
            if(count >= MaxCount):
                MaxCount = count
                bestVid = vid
        #recorre todos los videos en la lista de un cierto pais: O(m) donde m es el numero de videos en esta lista
        return bestVid, MaxCount


def getVidsByCountry(catalog, country):
    country = mp.get(catalog["countries"], country)
    #Obtiene la lista de videos correspondientes a un pais, a partir de un mapa: O(1)
    if country:
        return me.getValue(country)["videos"]
    return None
```

Para la solución del reto 1, en este requerimiento se hacía copia de listas que manejaban una cantidad enorme de datos. Aunque quizá esto no compensa la gran cantidad de memoria que usa un mapa, evitar crear una copia de un dato que ya existe es un logro que nos parece recalcable. En cuanto a tiempo, la duración de este requerimiento disminuye sustancialmente porque se suprime el uso de funciones con un orden considerable, pues se pasa del uso de alrededor de 2 loops, 2 sorts y una búsqueda binaria, a solo usar un loop.

COMPLEJIDAD REQUERIMIENTO 3

```
def getTrendCategory(catalog, category_id):
    category = mp.get(catalog["categories"], category_id)
    if category:
        videos = me.getValue(category)["videos"]
        sorted_list = sortVideoTitleTrend(videos)
        #uso de merge 2 veces: O(mlog(m)) por cada uno donde m es el número de videos de la c
ategoría

        count = 1
        previousTitle = ""
        previousDate = ""
        trend = None
        trendcount = 0

        for video in lt.iterator(sorted_list):
            if video["title"] != previousTitle:
                if (count > trendcount):
                    trend = video
                    trendcount = count
                count = 1
            elif (video["trending_date"] != previousDate):
                count += 1
            previousTitle = video["title"]
            previousDate = video["trending_date"]
        #for: O(m)

        return trend, trendcount
    return None
```

Debido a que al inicio del requerimiento se obtiene del catálogo la lista de los videos de solo la categoría buscada, durante la mayoría de la función las complejidades se relacionan no con el número total de videos en el catálogo sino solo una cantidad *m* que corresponde a los videos de la categoría deseada.
Para ordenar la lista tanto por título como por día de tendencia se implementa merge sort, que tiene una complejidad de O(mlog(m)) en cada instancia del sort.
El for se utiliza para recorrer todos los elementos de la lista ordenada, por lo que su complejidad es de O(m)

Comparado con el mismo requerimiento, pero en el reto anterior, este cuenta con una mejor complejidad, puesto que el anterior usaba no solo dos for en términos de *m* sino también un merge sort y un binary search (log(n)) en términos de *n* donde n son todos los videos en el catálogo. En consecuencia, el requerimiento en esta ocasión retorna un resultado en meros segundos, en vez de aproximadamente minuto y medio después como hacía su versión anterior.

COMPLEJIDAD REQUERIMIENTO 4

```
def getBestTag(catalog, tag, country, number):
    videos = getVidsByCountry(catalog, country)

    if videos:
        sorted_list = ma.sort(videos, compare_views)
        #uso de merge: O(mlog(m)) donde m es el número de videos de la del país
        tag1 = '"' + tag.lower() + '"'
        result = lt.newList('ARRAY_LIST')
        titles = lt.newList('ARRAYLIST')

        i = 0
        for video in lt.iterator(sorted_list):
            if tag1 in video['tags'].lower() and not lt.isPresent(titles, video['title']):
                i += 1
                lt.addLast(result, video)
                lt.addLast(titles, video['title'])
            if i == number:
                break
        #O(m) en el peor de los casos

        return result
    return None
```

Debido a que al inicio del requerimiento se obtiene del catálogo la lista de los videos de solo el país buscado, durante la mayoría de la función las complejidades se relacionan no con el número total de videos en el catálogo sino solo una cantidad *m* que corresponde a los videos del país deseado.
Para ordenar la lista de videos por views se implemente un merge sort, que tiene una complejidad de O(mlog(m)). Un for, naturalmente, recorre todos los elementos de la lista que se está iterando, por lo cual su ordenamiento sería O(m), sin embargo, se especifica que aquí ese es solo el peor caso puesto que este for se encuentra sujeto al número de videos que el usuario haya pedido listar. Si encuentra la cantidad de videos pedidos antes de realizar todo el recorrido el proceso se culmina.

En este reto se puede evidenciar claramente la mejora de la complejidad del requerimiento con respecto al reto 1. En nuestro caso, la función no fue capaz de completar sus operaciones en un tiempo estipulado, mientras que aquí no se demora más de un par de segundos en devolver un resultado.