Análisis de Complejidad Reto-2

Integrantes:

- E1: Daniela Alvarez Rodriguez-202020209-d.alvarezr@uniandes.edu.co

Requerimiento 1: Video Tendencia por categoría y país

```
def videos_categoria_pais(catalog, nombrecategoria, pais, numero):
(1)    nombrecategoria = nombrecategoria.replace(" ", "").lower()
(1)    categoria = mp.get(catalog['videos_por_categoria'], nombrecategoria)
(1)    videos = me.getValue(categoria)
(NlogN)videos_ordenados = sortVideos(videos, compareviews)
(1)  lista_videos = lt.newList(datastructure='ARRAY_LIST')


(N)    for i in range(1, lt.size(videos_ordenados)):
(1)        video = lt.getElement(videos_ordenados, i)
(1)        if video["country"].lower() == pais.lower():
(1)           if numero > 0:
(7)                vid_t = {"Nombre del video": video["title"], "Trending date": video["trending_date"],
                         "Nombre del canal": video["channel_title"], "Fecha Publicación": video["publish_time"],
                         "Reproducciones": video["views"], "Likes": video["likes"], "Dislikes": video["dislikes"]}
(1)                lt.addLast(lista_videos, vid_t)
(1)                numero-=1
(1)            elif numero == 0:
(1)                break


    return lista_videos
```

Cálculo de complejidad:

N= Tamaño lista videos para la categoria

$$O(n) = 3 + \ NlogN + 1 + 12\text{N} = \text{NlogN} + 12\text{N} + 4 = NlogN$$

Discusión

La complejidad de este algoritmo utilizando la notación BIG O es O(NlogN) esto quiere decir que su orden de crecimiento es linearítmico por lo que el tiempo de procesamiento va a aumentar en menor proporción a medida que aumenta la cantidad de datos.

Función videos tendencia: (Se utilizará más adelante)

```
def video_tendencia(videos):
(1)    videos_por_id = mp.newMap(lt.size(videos),
                                 maptype='CHAINING',
                                 loadfactor=2.0,
                                 comparefunction=comparecategories)
(1)    tendencia_videos = mp.newMap(lt.size(videos),
                                 maptype='CHAINING',
                                 loadfactor=2.0,
                                 comparefunction=comparecategories)

(N)    for i in range(1, lt.size(videos)):
(1)        video = lt.getElement(videos, i)
(M)        if mp.contains(tendencia_videos, video["video_id"]):
(1)            vid = mp.get(tendencia_videos, video["video_id"])
(1)            dias_tendencia = me.getValue(vid) + 1
(1)        else:
(1)            dias_tendencia = 1
(1)            mp.put(videos_por_id, video["video_id"], video)
(1)        mp.put(tendencia_videos, video["video_id"], dias_tendencia)

(1)    mas_dias = 0
(1)    video = {}
(M)    keys = mp.keySet(tendencia_videos)
(M)    for i in lt.iterator(keys):
(1)        pareja_dias = mp.get(tendencia_videos, i)
(1)        num_dias = me.getValue(pareja_dias)
(1)        if num_dias > mas_dias:
(1)            mas_dias = num_dias
(1)            vid = mp.get(videos_por_id, i)
(1)            video = me.getValue(vid)

(1)    video["Dias Tendencia"] = mas_dias
    return video
```

Cálculo de complejidad:

N= Tamaño lista videos entrados por parámetro

M = Tamaño Map donde aparece solo una llave por id de video

$$O(n) = 2 + \mathrm{N}(2 + 2\mathrm{M}) + 2 + M + 6M + 1 = 5 + 7M + 2N + 2NM = 2NM = NM$$

Requerimiento 2: Video tendencia por país

```
def video_tendencia_pais(catalog, pais):
(1)       pais = pais.replace(" ", "").lower()
(1)       videos_pais = mp.get(catalog['videos_por_pais'], pais)
(1)       videos = me.getValue(videos_pais)
(NlogN)   sortVideos(videos, comparelikes)
(NM)      return video_tendencia(videos)
```

Cálculo de complejidad:

N = Tamaño lista de videos para el país

M = Tamaño Map donde aparece solo una llave por id de video

$$O(n) = 3 + NlogN + NM = NM$$

Discusión:

La complejidad de este algoritmo, utilizando la notación Big O es O(nm). Esto quiere decir que su orden de crecimiento es cuadrática y el tiempo de procesamiento incrementará considerablemente entra más datos procesados

Requerimiento 3: Video tendencia por categoría

```
def video_tendencia_categoria(catalog, categoria):
(1)     categoria = categoria.replace(" ", "").lower()
(1)    videos_categoria = mp.get(catalog['videos_por_categoria'], categoria)
(1)    videos = me.getValue(videos_categoria)
(NlogN) SortVideos(videos, comparelikes)
(NM)    return video_tendencia(videos)
```

Cálculo de complejidad:

N = Tamaño lista de videos para el país

M = Tamaño Map donde aparece solo una llave por id de video

$$O(n) = 3 + NlogN + NM = NM$$

Discusión:

La complejidad de este algoritmo, utilizando la notación Big O es O(nm). Esto quiere decir que su orden de crecimiento es cuadrática y el tiempo de procesamiento incrementará considerablemente entra más datos procesados

Requerimiento 4: Video con más likes

```
def videos_likes(catalog, pais, tag, numero):
(1)    pais = pais.replace(" ", "").lower()
(1)    videos_pais = mp.get(catalog['videos_por_pais'], pais)
(1)    videos = me.getValue(videos_pais)
(NlogN)videos_ordenados = sortVideos(videos, comparelikes)
(1)    lista_videos = lt.newList(datastructure='ARRAY_LIST')

(N)   for i in range(1, lt.size(videos)):
(1)        video = lt.getElement(videos_ordenados, i)
(1)        if tag in video["tags"]:
(1)            if numero > 0:
(7)                vid_t = {"Nombre del video": video["title"], "Nombre del canal": video["channel_title"],
                        "Fecha Publicación": video["publish_time"],"Reproducciones": video["views"],
                        "Likes": video["likes"], "Dislikes": video["dislikes"], "Tags": video["tags"]}
(1)                lt.addLast(lista_videos, vid_t)
(1)                numero-= 1
(1)            elif numero == 0:
(1)                break

    return lista_videos
```

Cálculo de complejidad:

N = Tamaño lista videos para el país

$$O(n) = 3 + NlogN + 1 + 12N = NlogN + 12N + 4 = NlogN$$

Discusión:

La complejidad de este algoritmo utilizando la notación BIG O es O(NlogN), esto quiere decir que su orden de crecimiento es linearítmico por lo que el tiempo de procesamiento va a aumentar en menor proporción a medida que aumenta la cantidad de datos.