

# Estructuras de Datos y Algoritmos

Jose Luis Tavera - 201821999  
Juan Diego Yepes - 202022391

## Reto II

### 1 Construcción del Catálogo

Para dar solución a los requerimientos, fue necesario replantear la organización del catálogo del reto 1, para organizar la información en mapas en vez de listas.

#### Funciones del View

Para empezar a inicializar el catálogo y cargar la información, el usuario debe digitar las siguientes opciones 1 y 2, las cuales empiezan las siguientes instrucciones:

```
1  if int(inputs[0]) == 1:
2      x = controller.initCatalog()
3
4  elif str(inputs[0]) == "2":
5      print("\nCargando informacion de los archivos ....")
6      print('-----\n')
7      answer = controller.loadData(x)
8      print('Videos cargados: ' + str(controller.videosSize(x)))
9      print("Tiempo [ms]: ", f"{answer[0]:.3f}", " || ",
10           "Memoria [kB]: ", f"{answer[1]:.3f}")
11      print('\n')
12      loaded_categories = controller.categoriesSize(x)
13      print('Categorias cargadas: ' + str(loaded_categories))
14      print_table(x)
15      print('\n')
16      loaded_countries = controller.countriesSize(x)
17      print('Pa ses cargados: ' + str(controller.countriesSize(x)))
18      print_list(x)
```

Estas funciones hacen referencia a las funciones en el controller que se exponen a continuación. A su vez, imprimen la información retorno de lo que se cargó en el catálogo; es decir cuántas y cuáles categorías se cargaron, cuantos y cuáles países, etc.

#### Funciones del Controller

Las funciones utilizadas en el controller se detallan a continuación.

```
1  def initCatalog():
2      """
3      Llama la funcion de inicializacion del catalogo del modelo.
4      """
5      catalog = model.newCatalog()
6      return catalog
```

Esta función sirve de intermediaria con la función del model llamada newCatalog().

```
1  def loadData(catalog):
2      """
3      Carga los datos de los archivos y cargar los datos en la
```

```

4     estructura de datos, a su vez mide el tiempo de ejecución
5     """
6     delta_time = -1.0
7     delta_memory = -1.0
8
9     tracemalloc.start()
10    start_time = getTime()
11    start_memory = getMemory()
12
13    loadVideos(catalog)
14    loadCategories(catalog)
15
16    stop_memory = getMemory()
17    stop_time = getTime()
18    tracemalloc.stop()
19
20    delta_time = stop_time - start_time
21    delta_memory = deltaMemory(start_memory, stop_memory)
22
23    return delta_time, delta_memory

```

Esta función carga los datos de los archivos y mide el tiempo de ejecución de las operaciones utilizando la librería time. Para cargar los datos, referencia las funciones loadVideos() y loadCategories(), las cuales se exponen a continuación:

```

1 def loadVideos(catalog):
2     """
3     Carga los videos del archivo.
4     """
5     videosfile = cf.data_dir + 'videos/videos-large.csv'
6     input_file = csv.DictReader(open(videosfile, encoding='utf-8'))
7     for video in input_file:
8         model.addVideo(catalog, video)
9
10 def loadCategories(catalog):
11     """
12     Carga todas las categorías del archivo y los agrega a la lista de
13     categorías
14     """
15     categoriesfile = cf.data_dir + 'videos/category-id.csv'
16     input_file = csv.DictReader(
17         open(categoriesfile, encoding='utf-8'), delimiter='\t')
18     for category in input_file:
19         model.addCategory(catalog, category)
20

```

Estas funciones son las que leen el archivo .csv y van llamando a las otras funciones que agregan la información en el catálogo, las cuales se encuentran en el model.

## Funciones del Model

Esta primera función inicializa el catálogo vacío, especificando cuáles índices van a ser mapas y cuáles van a ser listas. Una vez inicializado el catálogo, se agrega la información mediante las siguientes funciones:

```

1 def newCatalog():
2     catalog = {'videos': None,
3               'category_id': None,
4               'category': None,
5               'country': None}
6
7
8     catalog['videos'] = lt.newList('ARRAY_LIST')
9     catalog['category_id'] = lt.newList('ARRAY_LIST')
10    catalog['category'] = mp.newMap(20, maptype='PROBING', loadfactor=0.5)
11    catalog['country'] = mp.newMap(20, maptype='PROBING', loadfactor=0.5)
12    return catalog

```

Luego, las funciones de carga de datos, llamadas para cada video individual en el controller, se exponen a continuación:

```

1 def addVideo(catalog, video):
2     """
3     La funcion de addVideo() adiciona los videos a una lista de videos,
4     a su vez los adiciona a los mapas de 'category' y 'country'
5     """
6     lt.addLast(catalog['videos'], video)
7     addVideoOnMap(catalog, video['category_id'], video, 'category')
8     addVideoOnMap(catalog, video['country'], video, 'country')

```

La función `addVideo()` no tiene retorno y se encarga de agregar individualmente, video por video, las funciones al catálogo. Primero, lo agrega a la lista de videos `catalog['videos']`; luego los agrega a cada uno de los mapas `video['category_id']` y `video['country']`. Para hacer esto, se referencia a la función `addVideoOnMap()` la cual se muestra a continuación:

```

1 def addVideoOnMap(catalog, int_input, video, catalog_key):
2     """
3     La funcion de addVideoOnMap() adiciona el video al mapa
4     que se ha seleccionado.
5     Args:
6         catalog: Catalogo de videos
7         int_input: Llave a analizar
8         video: Video a aadir
9         catalog_key: Especifica cu l catalogo
10    """
11    selected_map = catalog[catalog_key]
12    existkey = mp.contains(selected_map, int_input)
13    if existkey:
14        entry = mp.get(selected_map, int_input)
15        value = me.getValue(entry)
16    else:
17        value = newSeparator(int_input, catalog_key)
18        mp.put(selected_map, int_input, value)
19    lt.addLast(value['videos'], video)

```

Esta función agrega el video individual al mapa seleccionado, que se define con el parámetro `'catalog_key'`. Si la llave del video, que puede ser la categoría o el país, ya se encuentra en el mapa, lo que hace la función es agregarlo a esa llave expandiendo la lista de valores organizada en Array. Si no, crea una nueva lista y una nueva llave organizada en Array y agrega ese valor.

## 2 Primer Requerimiento

El equipo de análisis quiere conocer cuáles son los  $n$  videos con más views que son tendencia en un determinado país, dada una categoría específica. Para dar respuesta a este requerimiento el equipo de desarrollo debe recibir como entrada la siguiente información:

- category\_name
- country
- Número de videos que quiere listar ( $n$ )

Y como respuesta debe presentar en consola la siguiente información:

- trending\_date
- title
- channel\_title
- publish\_time
- views
- likes
- dislikes

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

### Funciones del View

El view le pide al usuario ingresar el país y la categoría de referencia que servirán como parámetro para la función del controller que une con la función homónima

```
1 elif str(inputs[0]) == "3":
2     pais = input("\nIngrese el pa s de referencia: ")
3     categoria = int(
4         input('Ingrese el c digo la categor a de referencia: '))
5     n = input("Ingrese el n mero de videos que desea imprimir: ")
6     print("\nCargando ...")
7     result = controller.getVideosByCategoryAndCountry(x, categoria, pais)
8     printResults(result[1], int(n))
```

### Funciones del Controller

Análogamente, el controller invoca la función homónima del model haciendo de puente entre el view y el model.

```
1 def getVideosByCategoryAndCountry(catalog, category, country):
2     '''
3     Retorna los videos dado un pa s y categor a espec ficos
4     '''
5     return model.getVideosByCategoryAndCountry(catalog, category, country)
```

### Funciones del Model

La función general del model, similar a la del reto 1, invoca tres funciones: dos para crear las sublistas y otro para sortear los videos.

```

1 def getVideosByCategoryAndCountry(catalog, category, country):
2     """
3     La función de getVideosByCriteriaMap() filtra los videos por una
4     llave y categoría específicas.
5     """
6     sublist = getVideosByCriteriaMap(
7         catalog, 'category', category).get('videos')
8     sublist2 = getVideosByCriteriaList(sublist, 'country', country)
9     return sortVideos(sublist2, lt.size(sublist2), 'cmpVideosByViews')

```

La función de CriteriaMap filtra los videos por categoría usando el mapa del catálogo.

```

1 def getVideosByCriteriaMap(catalog, criteria, key):
2     """
3     La función de getVideosByCriteriaMap() filtra los videos por un
4     criterio específico dado un key. El catálogo debe ser un mapa.
5     """
6     values = catalog[criteria]
7     entry = mp.get(values, str(key))
8     result = me.getValue(entry)
9     return result

```

La función de CriteriaList itera la lista para filtrar por el segundo criterio que corresponde al país.

```

1 def getVideosByCriteriaList(catalog, criteria, x):
2     """
3     La función de getVideosByCriteriaList() filtra los videos por un
4     criterio específico dado un x. El catálogo debe ser una lista.
5     """
6     listaretorno = lt.newList("ARRAY_LIST")
7     for element in lt.iterator(catalog):
8         nombre_pais = element.get(criteria)
9         if nombre_pais == x:
10             lt.addLast(listaretorno, element)
11
12     return listaretorno

```

Finalmente, sortVideos organiza la sublista final por merge sort para obtener los videos con más views por medio de la función de cmpVideosbyViews.

```

1 def sortVideos(catalog, size, cmp):
2     """
3     La Función sortVideos() organiza los videos de acuerdo
4     al parámetro cmp.
5     """
6     sub_list = lt.subList(catalog, 0, size)
7     sub_list = sub_list.copy()
8     start_time = time.process_time()
9
10    if cmp == 'cmpVideosByViews':
11        sorted_list = ms.sort(sub_list, cmpVideosByViews)
12    if cmp == 'comparetitles':
13        sorted_list = ms.sort(sub_list, comparetitles)
14    if cmp == 'comparelikes':
15        sorted_list = ms.sort(sub_list, comparelikes)
16
17    stop_time = time.process_time()
18    elapsed_time_mseg = (stop_time - start_time)*1000
19    return elapsed_time_mseg, sorted_list

```

### 3 Segundo Requerimiento (Juan Diego Yepes)

El equipo de análisis quiere conocer cuál es el video que más días ha sido trending para un país específico. Para dar respuesta a este requerimiento el equipo de desarrollo debe recibir como entrada la siguiente información:

- country

Y como respuesta debe presentar en consola la siguiente información:

- title
- channel.title
- country
- número de días

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

#### Funciones del View

El segundo requerimiento es invocado en el view cuando el usuario ingresa la opción número "4":

```
1 elif str(inputs[0]) == "4":
2     pais = input("Ingrese el pa s de referencia: ")
3     print("\nCargando ....")
4     videos_filtrados = controller.getVideosByCountry(x, pais)
5     result = controller.getMostTrendingDaysByID(videos_filtrados.get(
6         'videos'))
7     printResultsv3(result)
```

Para filtrar los videos, se invoca la función `getVideosByCountry()` del controller.

#### Funciones del Controller

Las funciones utilizadas en el controller se detallan a continuación.

```
1 def getVideosByCountry(catalog, country):
2     '''
3     Retorna los videos de un pa s espec fico
4     '''
5     return model.getVideosByCriteriaMap(catalog, 'country', country)
```

Esta función a su vez invoca la función del model `getVideosByCriteriaMap()`

#### Funciones del Model

```
1 def getVideosByCriteriaMap(catalog, criteria, key):
2     """
3     La funcion de getVideosByCriteriaMap() filtra los videos por un
4     criterio espec fico dado un key. El cat logo debe ser un mapa.
5     """
6     values = catalog[criteria]
7     entry = mp.get(values, str(key))
8     result = me.getValue(entry)
9     return result
```

La función de `CriteriaMap` filtra los vídeos por categoría usando el mapa del catálogo.

```

1 def getMostTrendingDaysByID(videos):
2     """
3     La funci n de getMostTrendingDaysByID() itera la lista ordenada y
4     retorna el elemento que m s se repite.
5     """
6     elemento = lt.firstElement(videos)
7     mayor_titulo = None
8     mayor = 0
9     i = 0
10
11     for video in lt.iterator(videos):
12         if video['video_id'] == elemento['video_id']:
13             i += 1
14         else:
15             if i > mayor:
16                 mayor_titulo = elemento
17                 mayor = i
18             i = 1
19             elemento = video
20
21     if i > mayor:
22         mayor_titulo = elemento
23         mayor = i
24     return (mayor_titulo, mayor)

```

## 4 Tercer Requerimiento (Jose Luis Tavera)

El equipo de análisis quiere conocer cuál es el video que más días ha sido trending para una categoría específica. Para dar respuesta a este requerimiento el equipo de desarrollo debe recibir como entrada la siguiente información:

- country

Y como respuesta debe presentar en consola la siguiente información:

- title
- channel.title
- country
- número de días

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

### Funciones del View

Al seleccionar la opción cinco el usuario solo necesita ingresar la categoría de referencia para invocar dos funciones del controller: uno que filtra usando el mapa y otra que obtiene el video que más veces aparece

```
1 .elif str(inputs[0]) == "5":
2     categoria = int(
3         input('Ingrese el c digo la categor a de referencia: '))
4     print("\nCargando ....")
5     videos_filtrados = controller.getVideosByCategory(x, categoria)
6     result = controller.getMostTrendingDaysByID(videos_filtrados.get(
7         'videos'))
8     printResultsv3(result)
```

### Funciones del Controller

Análogamente, el controller cumple su función como puente entre el view y el model, invocando las funciones homónimas del model.

```
1 def getVideosByCategory(catalog, category):
2     '''
3     Retorna los videos de una categor a espec fica
4     '''
5     return model.getVideosByCriteriaMap(catalog, 'category', category)

1 def getMostTrendingDaysByID(catalog):
2     '''
3     Retorna los videos que m s tiempo fueron tendencia
4     '''
5     sorted_videos = model.sortVideos(
6         catalog, lt.size(catalog), 'comparetitles')
7     result = model.getMostTrendingDaysByID(sorted_videos[1])
8     return result
```

### Funciones del Model

La función de getVideosByCriteriaMap usa los mapas creados en el catálogo para filtrar usando la categoría ingresada.



```

1 def getVideosByCriteriaMap(catalog, criteria, key):
2     """
3     La función de getVideosByCriteriaMap() filtra los videos por un
4     criterio específico dado un key. El catálogo debe ser un mapa.
5     """
6     values = catalog[criteria]
7     entry = mp.get(values, str(key))
8     result = me.getValue(entry)
9     return result

```

Inicialmente, la función necesita recibir la lista filtrada sorteada alfabéticamente para poder iterarla e ir guardando el video que más veces se repite.

```

1 def getMostTrendingDaysByID(videos):
2     """
3     La función de getMostTrendingDaysByID() itera la lista ordenada y
4     retorna el elemento que más se repite.
5     """
6     elemento = lt.firstElement(videos)
7     mayor_titulo = None
8     mayor = 0
9     i = 0
10
11     for video in lt.iterator(videos):
12         if video['video_id'] == elemento['video_id']:
13             i += 1
14         else:
15             if i > mayor:
16                 mayor_titulo = elemento
17                 mayor = i
18             i = 1
19             elemento = video
20
21     if i > mayor:
22         mayor_titulo = elemento
23         mayor = i
24     return (mayor_titulo, mayor)

```

## 5 Cuarto Requerimiento

El equipo de análisis quiere conocer cuáles son los  $n$  videos diferentes con más likes en un país con un tag específico. Recuerde que el campo tags está compuesto por múltiples palabras o frases entre comillas (") y separadas por el siguiente caracter —, así que el tag requerido puede ser una sub-cadena en el campo.

- tag

Y como respuesta debe presentar en consola la siguiente información:

- title
- channel\_title
- publish\_time
- views
- likes
- dislikes
- tags

Antes de analizar los ordenes de crecimiento del requerimiento, señalaremos las funciones y sus tareas dentro del modelo MVC. Por lo tanto, empezaremos por el view, seguido del controller y finalmente haremos el análisis del orden principalmente en las funciones del model.

### Funciones del View

La función del view le pide al usuario ingresar el país y el tag de referencia para posteriormente invocar la función de `getVideosByCountryAndTag`

```
1 elif str(inputs[0]) == "6":
2     pais = input("Ingrese el pa s de referencia: ")
3     tag = input('Ingrese el tag de referencia: ')
4     n = int(input("Ingrese el n mero de videos que desea imprimir: "))
5     print("\nCargando ....")
6
7     result = controller.getVideosByCountryAndTag(
8         x, tag, pais)
```

### Funciones del Controller

Posteriormente, el controller actúa como puente entre el view y el controller invocando su función homónima de `getVideosByCountryAndTag`

```
1 def getVideosByCountryAndTag(catalog, tag, country):
2     '''
3     Retorna los videos de un pa s y tag espec ficos
4     '''
5     return model.getVideosByCountryAndTag(catalog, tag, country)
```

### Funciones del Model

La función de `getVideosByCountryAndTag` usa tres funciones del model: la primera filtra usando los mapas por categoría, la segunda itera para filtrar los vídeos que incluyen un tag específico y la última sortea los vídeos por likes.

```

1 def getVideosByCountryAndTag(catalog, tag, country):
2     """
3     La funcion de getVideosByCountryAndTag() filtra los videos por un pa s
4     y tag espec fico
5     """
6     sublist = getVideosByCriteriaMap(catalog, 'country', country).get('videos')
7     sublist2 = getVideosByTag(sublist, tag)
8     sorted_list = sortVideos(
9         sublist2, int(lt.size(sublist2)), 'comparelikes')
10    return sorted_list

```

```

1 def getVideosByTag(videos, tag):
2     """
3     La funcion de getVideosByTag() filtra los videos por un tag
4     espec fico
5     """
6     lista = lt.newList('ARRAY_LIST')
7     i = 1
8
9     while i <= lt.size(videos):
10        c_tags = lt.getElement(videos, i).get('tags')
11        tagpresence = tag in c_tags
12
13        if tagpresence:
14            element = lt.getElement(videos, i)
15            lt.addLast(lista, element)
16
17        i += 1
18
19    return lista

```

```

1 def sortVideos(catalog, size, cmp):
2     """
3     La Funci n sortVideos() organiza los videos de acuerdo
4     al par metro cmp.
5     """
6     sub_list = lt.subList(catalog, 0, size)
7     sub_list = sub_list.copy()
8     start_time = time.process_time()
9
10    if cmp == 'cmpVideosByViews':
11        sorted_list = ms.sort(sub_list, cmpVideosByViews)
12    if cmp == 'comparetitles':
13        sorted_list = ms.sort(sub_list, comparetitles)
14    if cmp == 'comparelikes':
15        sorted_list = ms.sort(sub_list, comparelikes)
16
17    stop_time = time.process_time()
18    elapsed_time_mseg = (stop_time - start_time)*1000
19    return elapsed_time_mseg, sorted_list

```

## 6 Funciones Alternas

Con el fin de aprovechar los beneficios de los mapas para disminuir los ordenes de crecimiento, propusimos dos versiones alternas a la función de `getMostTrendingDaysByID`, en esta versión solo es necesario hacer dos iteraciones y no hace falta sortear la lista lo que disminuiría en teoría el tiempo de ejecución. Sin embargo, teniendo en cuenta el uso de TAD listas, algunas funciones necesitan reiterar la lista o los mapas creados por lo que en la práctica no fue la función más rápida.

```
1 def getMostTrendingDaysByIDv1(videos):
2     ids = mp.newMap(
3         500,
4         maptype='PROBING',
5         loadfactor=0.8)
6     pos = mp.newMap(
7         500,
8         maptype='PROBING',
9         loadfactor=0.8)
10    ids_list = lt.newList('ARRAY_LIST')
11
12    i = 1
13
14    while i <= lt.size(videos):
15        video_id = lt.getElement(videos, i).get('video_id')
16        presence = lt.isPresent(ids_list, video_id) != 0
17
18        if presence:
19            n = int(mp.get(ids, video_id).get('value'))
20            n += 1
21            mp.put(ids, video_id, n)
22        else:
23            mp.put(ids, video_id, 1)
24            mp.put(pos, video_id, i)
25            lt.addLast(ids_list, video_id)
26        i += 1
27
28    mayor = video_id
29    repeticiones_mayor = int(mp.get(ids, mayor).get('value'))
30    mayor = video_id
31    repeticiones_mayor = int(mp.get(ids, mayor).get('value'))
32
33    for each_id in ids_list:
34        repetitions = int(mp.get(ids, each_id).get('value'))
35        if repetitions > repeticiones_mayor:
36            mayor = each_id
37            repeticiones_mayor = repetitions
38
39    position = mp.get(pos, mayor).get('value')
40    repetitions = mp.get(ids, mayor).get('value')
41    result = lt.getElement(videos, position)
42
43    return (result, repetitions)
```

## 7 Cálculos de Ordenes de Crecimiento

En este apartado haremos los cálculos de los ordenes de crecimiento de cada una de las funciones. Ya que sería negligente analizar las funciones en términos de "N" (siento este el número total de datos) definiremos las siguientes variables y haremos una aproximación en términos de N para su mejor y peor caso:

- $P$ : Es el número total de países cargados por el catálogo que corresponde a 10 países.
- $C$ : Es el número total de categorías cargadas por el catálogo que corresponde a 32 categorías aunque solo 18 categorías son cargadas ya que las otras no corresponden a ningún vídeo.
- $NP$ : Es el tamaño promedio de datos filtrados por país. Su mejor caso es igual a:  $NP_B$  y su peor caso es:  $NP_W$
- $NC$ : Es el tamaño promedio de datos filtrados por categoría. Su mejor caso es igual a:  $NC_B$  y su peor caso es:  $NC_W$
- $NPC$ : Es el tamaño promedio de una filtración por país y categoría. Su mejor caso es igual a:  $NPC_B$  y su peor caso es:  $NPC_W$
- $NPT$ : Es el tamaño promedio de una filtración por país y tags. Su mejor caso es igual a:  $NPT_B$  y su peor caso es:  $NPT_W$

Ahora bien, analizaremos paso a paso las funciones que de mayor orden de crecimiento por requerimiento

- **Requerimiento 1:**

**getVideosByCountry:** Se demora ( $P$ ), ya que revisa las llaves del catálogo en el apartado de `catalog["countries"]`.

**getVideosByCategory:** Se demora ( $NP$ ), ya que filtra la lista anterior por categoría para lo cual necesita iterarla.

**sortVideos:** Al usar merge sort se demora ( $NPC \cdot \log(NPC)$ ) en su mejor, peor y caso promedio.

**printResultsv1:** Se demora en congruencia con el parámetro "n", es decir ( $n$ ) porque esas son las veces que itera la lista.

- **Requerimiento 2:**

**getVideosByCountry:** Se demora ( $P$ ), ya que revisa las llaves del catálogo en el apartado de `catalog["countries"]`.

**sortVideos:** Al usar merge sort se demora ( $NP \cdot \log(NP)$ ) en su mejor, peor y caso promedio.

**getMostTrendingDaysByTitle:** Se demora ( $NP$ ) ya que debe iterar toda la lista sorteada para buscar el mayor.

- **Requerimiento 3:**

**getVideosByCategory:** Se demora ( $C$ ), ya que revisa las llaves del catálogo en el apartado de `catalog["category_id"]`.

**sortVideos:** Al usar merge sort se demora ( $NC \cdot \log(NC)$ ) en su mejor, peor y caso promedio.

**getMostTrendingDaysByTitle:** Se demora ( $NC$ ) ya que debe iterar toda la lista sorteada.

- **Requerimiento 4:**

**getVideosByCountry:** Se demora ( $P$ ), ya que revisa las llaves del catálogo en el apartado de `catalog["countries"]`

**getVideosByTag:** Se demora ( $NPT$ ), ya que filtra la lista anterior por tags para lo cual necesita iterarla.

**sortVideos:** Al usar merge sort se demora  $(NPT \cdot \log(NPT))$  en su mejor, peor y caso promedio.

**printResultsv2:** Se demora  $(NPT)$  en el peor caso ya que debe escoger los mejores videos y asegurarse de no repetirlos iterando la lista de tamaño  $(NPT)$ .

## 8 Resultados

En congruencia con lo anterior, haciendo los cálculos pertinentes para cada una de las notaciones en relación a las variables de tamaño definidas anteriormente, obtuvimos los siguientes ordenes de crecimiento para cada uno de los requerimientos:

|                       | Requerimiento 1                                   | Requerimiento 2                                 |
|-----------------------|---|---|
| <b>Notacion Tilda</b> | $\sim N(15 + P + NP + (NPC \cdot \log(NPC) + n))$ | $\sim N(10 + P + NP + (NP \cdot \log(NP)))$     |
| <b>Big Theta</b>      | $\Theta(NP)$                                      | $\Theta(NP \cdot \log(NP))$                     |
| <b>Big O</b>          | $O(NP_W)$   | $O(NP_W \cdot \log(NP_W))$                      |
| <b>Big Omega</b>      | $\Omega(NP_B)$                                    | $\Omega(NP_B \cdot \log(NP_B))$                 |
|                       | Requerimiento 3                                   | Requerimiento 4                                 |
| <b>Notacion Tilda</b> | $\sim N(10 + NC + (NC \cdot \log(NC)))$           | $\sim N(20 + P + 2NPT + (NPT \cdot \log(NPT)))$ |
| <b>Big Theta</b>      | $\Theta(NC \cdot \log(NC))$                       | $\Theta(NPT \cdot \log(NPT))$                   |
| <b>Big O</b>          | $O(NC_W \cdot \log(NC_W))$                        | $O(NPT_W \cdot \log(NPT_W))$                    |
| <b>Big Omega</b>      | $\Omega(NC_B \cdot \log(NC_B))$                   | $\Omega(NPT_B \cdot \log(NPT_B))$               |