ANÁLISIS DE RESULTADOS RETO 2

GRUPO #3

TABLA DE CONTENIDOS:

INTEGRANTES DEL GRUPO

1. Santiago Beltran Melo, s.beltranm1@uniandes.edu.co, 201911611.
2. Samuel Esteban Pardo Forero, s.pardof@uniandes.edu.co, 202410675.
3. Gerónimo Rojas, g.rojasr@uniandes.edu.co, 202215835.

ANÁLISIS DE COMPLEJIDAD DE LOS REQUERIMIENTOS

1. **Load Data:**

```python
def load_data(catalog, filename):
    movies = csv.DictReader(open(".\\Data\\Challenge-2\\"+filename, encoding='utf-8'))


    for elemento in movies:

        rta = {}
        rta['id'] = elemento['id']
        rta['title'] =  elemento['title']
        rta['original_language'] = elemento['original_language']
        rta['release_date'] = elemento['release_date']
        rta['revenue'] = elemento['revenue']
        rta['runtime'] = elemento['runtime']
        rta['status'] = elemento['status']
        rta['vote_average'] = elemento['vote_average']
        rta['vote_count'] = elemento['vote_count']
        rta['budget'] = elemento['budget']
        genres_list = json.loads(elemento['genres'].replace("'","\""))
        k=lt.new_list()
        for genre in genres_list:
            lt.add_last(k,genre)
        rta['genres'] = k

        production_companies_list = json.loads(elemento['production_companies'])
        x = lt.new_list()
        for production_companies in production_companies_list:
            lt.add_last(x,production_companies)

        rta['production_companies'] = x
        lt.add_last(catalog["movies"],rta)

        idioma = elemento['original_language']


        movies_in_language = mp.get(catalog['ordenado_idioma'], idioma)

        if movies_in_language is None:
            lista_peliculas = lt.new_list()
            lt.add_last(lista_peliculas, rta)
            mp.put(catalog['ordenado_idioma'], idioma, lista_peliculas)
        else:
            lt.add_last(movies_in_language, rta)

        año=get_anio(elemento["release_date"])
        movies_in_anio = mp.get(catalog['ordenado_año'], año)
        if movies_in_anio is None:
            lista_año = lt.new_list()
            lt.add_last(lista_año, rta)
            mp.put(catalog['ordenado_año'], año, lista_año)
        else:
            lt.add_last(movies_in_anio, rta)

    return catalog
```

La complejidad de la función de carga de datos es O(n). Hay ciclos anidados, pero estos ni iteran sobre las mismas películas, por lo que la complejidad no aumenta.

2. **Get data:**

```python
def get_data(catalog, id):
    """
    Retorna un dato por su ID.
    """
    for i in range(0,lt.size(catalog["movies"])):
        if catalog["movies"]["elements"][i]["id"]==id:
            return catalog["movies"]["elements"][i]
        else:
            return "no se encontro"
```

Esta función llama la función “size” de array list, la cual es O(1), además, realiza un único ciclo sobre el catálogo de películas, por ende, es O(n).

3. **Requerimiento 1(**Samuel Pardo**):**

```python
def req_1(catalog, idioma, movie_title):
    """
    Retorna el resultado del requerimiento 1
    """
    entry = mp.get(catalog['ordenado_idioma'], idioma)

    if entry is None:
        return "Ninguna película fue encontrada"

    movies_in_language = entry['elements']

    for movie in movies_in_language:
        if movie['title'].lower() == movie_title.lower():

            net_profit = None
            if float(movie["revenue"]) != 0 and float(movie["budget"]) != 0:
                net_profit = float(movie["revenue"]) - float(movie["budget"])

            respuesta = {
                "Título original": movie['title'],
                "Duración (minutos)": movie['runtime'],
                "Fecha de publicación": movie['release_date'],
                "Presupuesto": movie['budget'],
                "Dinero recaudado": movie['revenue'],
                "Ganancia": net_profit,
                "Puntaje de calificación": movie['vote_average'],
                "Idioma original": movie['original_language']
            }
            return respuesta
```

```python
def get(my_map, key):
    for i in range(len(my_map["table"]["elements"])):
        entry = my_map['table']["elements"][i]

        if entry['key'] == key:
            return entry["value"]
```

El requerimiento 1 inicialmente realiza un llamado a la función *get, cuya complejidad es O(n), ya que esta realiza un único ciclo*.

En cuanto al requerimiento 1, en el algoritmo nos encontramos un único ciclo y algunas asignaciones y operaciones matemáticas, lo cual cuenta con complejidad O(1)

En conclusión, la complejidad del algoritmo es O(2n), lo cual equivale a O(n).

4. **Requerimiento 2(**Samuel Pardo**):**

```python
def req_2(catalog, n, idioma):
    """
    Retorna el resultado del requerimiento 2
    """
    movies_in_language_entry = mp.get(catalog['ordenado_idioma'], idioma)

    if movies_in_language_entry is None:
        return {
            "total_movies": 0,
            "movies": []
        }

    movie_list = movies_in_language_entry['elements']
    movie_list_released = [movie for movie in movie_list if movie['status'] == 'Released']
    total_movies = len(movie_list_released)

    if total_movies == 0:
        return {
            "total_movies": 0,
            "movies": []
        }

    recent_movies = []
    for movie in movie_list_released:
        if len(recent_movies) < n:
            recent_movies.append(movie)
            recent_movies.sort(key=lambda x: x['release_date'], reverse=True)
        else:
            if movie['release_date'] > recent_movies[-1]['release_date']:
                recent_movies[-1] = movie
                recent_movies.sort(key=lambda x: x['release_date'], reverse=True)

    resultado = {
        "total_movies": total_movies,
        "movies": []
    }

    for movie in recent_movies:
        budget = movie['budget'] if movie['budget'] else "Undefined"
        revenue = movie['revenue'] if movie['revenue'] else "Undefined"
        profit = None
        if budget != "Undefined" and revenue != "Undefined":
            profit = float(revenue) - float(budget)

        resultado['movies'].append({
            "release_date": movie['release_date'],
            "original_title": movie['title'],
            "budget": movie['budget'],
            "revenue": movie['revenue'],
            "profit": profit,
            "runtime": movie['runtime'],
            "vote_average": movie['vote_average']
        })

    return resultado
```

```python
def get(my_map, key):
    for i in range(len(my_map["table"]["elements"])):
        entry = my_map['table']["elements"][i]

        if entry['key'] == key:
            return entry["value"]
```

En el caso del requerimiento 2, una vez más iniciamos con una llamada a la función get, cuya complejidad, como ya fue mencionada anteriormente, es O(n).

Siguiendo con el código, encontramos 2 ciclos no anidados, lo que aumenta la complejidad del algoritmo a O(n).

En el primer ciclo For, nos encontramos con una condición que, de ser cumplida, da paso al llamado de la función sort, cuya complejidad es O(N log N).

Para concluir, la complejidad del algoritmo en el peor de los casos es O(N Log N), y en el mejor es O(n).

5. **Requerimiento 3(**Gerónimo Rojas**):**

```
def req_3(catalog,idioma,fecha_ini,fecha_final):
    lista=mp.get(catalog["ordenado_idioma"],idioma)

    fecha_i=fecha_str_a_fecha_dias(fecha_ini)
    fecha_f=fecha_str_a_fecha_dias(fecha_final)
    dic={}
    total=0
    tiempo_prom=0
    for j in range(0,lt.size(lista)):
        if fecha_str_a_fecha_dias(lista["elements"][j]["release_date"])>=fecha_i and fecha_str_a_fecha_dias(lista["elements"][j]["release_date"])<= fecha_f:
            total+=1
            tiempo_prom+=float(lista["elements"][j]["runtime"])
            if fecha_str_a_fecha_dias(lista["elements"][j]["release_date"]) not in dic:
                lista["elements"][j]["net_profit"]="undefined"
                if float(lista["elements"][j]["revenue"])!=0 and float(lista["elements"][j]["budget"])!=0:
                    lista["elements"][j]["net_profit"]=float(lista["elements"][j]["revenue"])-float(lista["elements"][j]["budget"])
                dic[fecha_str_a_fecha_dias(lista["elements"][j]["release_date"])]=lista["elements"][j]
    new_dic=dict(sorted(dic.items()))
    new_dic["total"]=total
    if total!=0:
     new_dic["tiempo_prom"]=tiempo_prom/total
    else:
        new_dic["tiempo_prom"]="no hay ninguna pelicula para promediar"
    return new_dic
```

```
def get(my_map, key):
    for i in range(len(my_map["table"]["elements"])):
        entry = my_map['table']["elements"][i]

        if entry['key'] == key:
            return entry["value"]
```

```
def fecha_str_a_fecha_dias(date):
    año=float(date[:4])
    mes=float(date[5:7])
    dias=float(date[8:])
    return (año*365)+(mes*30)+(dias)
```

Para el caso del requerimiento 3, una vez más iniciamos con un llamado a la función get (complejidad O(n)).

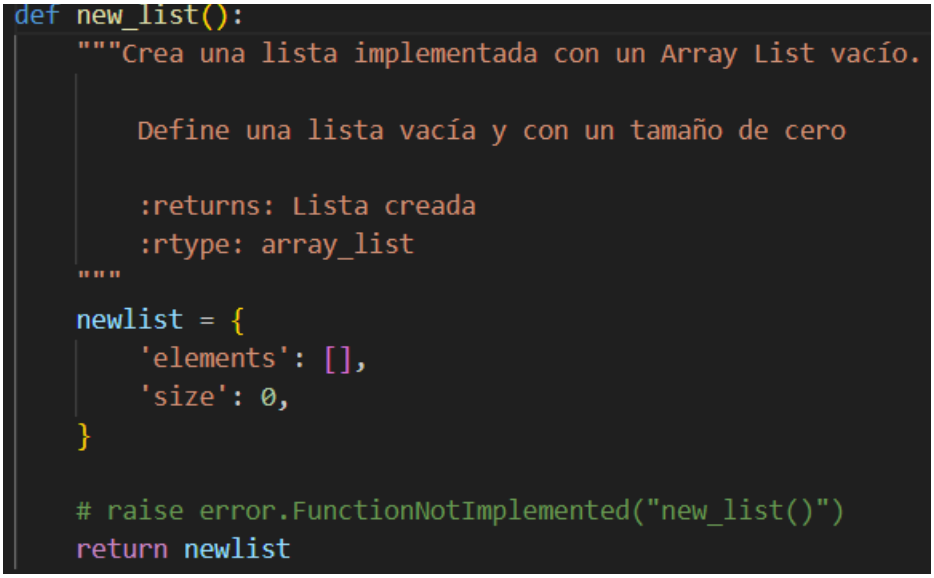
De forma complementaria, usamos la función fecha_str_a_fecha_dias, la cual toma una fecha en formato “YYYY-MM-DD”, y la convierte a la suma total de los días. Su complejidad es O(1).

Adicionalmente, nos encontramos con un único ciclo For (complejidad O(n)), y dentro de si se realizan operaciones aritméticas y de asignación (complejidad O(1)).

Finalmente, podemos observar un llamado a la función sorted(dic.items()), la cual cuenta con una complejidad O(N Log N).

En conclusión, la complejidad general del algoritmo es O(N Log N).

6. **Requerimiento 4(**Samuel Pardo**):**



```
def new_list():
    """Crea una lista implementada con un Array List vacío.

        Define una lista vacía y con un tamaño de cero

        :returns: Lista creada
        :rtype: array_list
    """
    newlist = {
        'elements': [],
        'size': 0,
    }

    # raise error.FunctionNotImplemented("new_list()")
    return newlist
```

```python
def req_4(catalog, estado, fecha_inicial, fecha_final):
    """
    Retorna el resultado del requerimiento 4 basado en el estado de producción y rango de fechas.
    """
    peliculas_filtradas = lt.new_list()
    fecha_inicial = fecha_str_a_fecha_dias(fecha_inicial)
    fecha_final = fecha_str_a_fecha_dias(fecha_final)

    for i in range(lt.size(catalog['movies'])):
        movie = lt.get_element(catalog['movies'], i)
        fecha_pelicula = fecha_str_a_fecha_dias(movie['release_date'])
        if movie['status'] == estado and fecha_inicial <= fecha_pelicula <= fecha_final:
            lt.add_last(peliculas_filtradas, movie)

    num_peliculas = lt.size(peliculas_filtradas)
    if num_peliculas == 0:
        return {'total_peliculas': 0, 'mensaje': 'No hay peliculas que cumplan los criterios'}

    total_duracion = 0
    for i in range(lt.size(peliculas_filtradas)):
        movie = lt.get_element(peliculas_filtradas, i)
        duracion = movie['runtime']
        if duracion and duracion != '':
            total_duracion += float(duracion)

    tiempo_promedio = total_duracion / num_peliculas if num_peliculas > 0 else 0
    lt.merge_sort(peliculas_filtradas, lambda x, y: fecha_str_a_fecha_dias(x['release_date']) > fecha_str_a_fecha_dias(y['release_date']))
    peliculas_mostradas = lt.sub_list(peliculas_filtradas, 0, min(10, num_peliculas))

    respuesta = {
        "total_peliculas": num_peliculas,
        "tiempo_promedio_duracion": tiempo_promedio,
        "peliculas": []
    }

    for i in range(lt.size(peliculas_mostradas)):
        movie = lt.get_element(peliculas_mostradas, i)
        presupuesto = movie['budget'] if movie['budget'] != '' else None
        recaudado = movie['revenue'] if movie['revenue'] != '' else None
        ganancia = None if presupuesto is None or recaudado is None else float(recaudado) - float(presupuesto)

        respuesta["peliculas"].append({
            "release_date": movie['release_date'],
            "original_title": movie['title'],
            "budget": presupuesto,
            "revenue": recaudado,
            "profit": ganancia,
            "runtime": movie['runtime'],
            "vote_average": movie['vote_average'],
            "original_language": movie['original_language']
        })

    return respuesta
```

```python
def fecha_str_a_fecha_dias(date):
    año=float(date[:4])
    mes=float(date[5:7])
    dias=float(date[8:])
    return (año*365)+(mes*30)+(dias)
```

```python
def size(my_list):
    """ Retorna el número de elementos de la lista.

        :param my_list: La lista a examinar
        :type my_list: array_list

        :returns: Número de elementos de la lista
        :rtype: int
    """
    try:
        # raise error.FunctionNotImplemented("size()")
        return my_list['size']
    except Exception as exp:
        error.reraise(exp, 'arraylist->size: ')
```

```python
def get_element(my_list, pos):
    """ Retorna el elemento en la posición ``pos`` de la lista.

        Se recorre la lista hasta el elemento ``pos``, el cual debe ser igual o mayor
        que cero y menor al tamaño de la lista.
        Se retorna el elemento en dicha posición sin eliminarlo.
        La lista no puede ser vacía.

        :param my_list: La lista a examinar
        :type my_list: array_list
        :param pos: Posición del elemento a retornar
        :type pos: int

        :returns: Elemento en la posición ``pos``
        :rtype: any
    """

    try:
        # raise error.FunctionNotImplemented("get_element()")
        return my_list['elements'][pos]
    except Exception as exp:
        error.reraise(exp, 'arraylist->getElement: ')
```

```python
def add_last(my_list, element):
    """ Agrega un elemento en la última posición de la lista.

        Al agregar un elemento en la última posición de la lista y se incrementa el tamaño de la lista en uno.

        :param my_list: ArrayList en la que se va a insertar el elemento
        :type my_list: array_list
        :param element: Elemento a insertar
        :type element: any

        :returns: ArrayList con el elemento insertado en la última posición
        :rtype: array_list
    """
    try:
        my_list['elements'].append(element)
        my_list['size'] += 1
        # raise error.FunctionNotImplemented("add_last()")
    except Exception as exp:
        error.reraise(exp, 'arraylist->addLast: ')
```

```python
def merge_sort(my_list, sort_crit):

    """
    n = size(my_list)
    if n > 1:
        mid = (n // 2)
        #se divide la lista original, en dos partes, izquierda y derecha, desde el punto mid.
        left_list = sub_list(my_list, 0, mid)
        right_list = sub_list(my_list, mid, n - mid)

        #se hace el llamado recursivo con la lista izquierda y derecha
        merge_sort(left_list, sort_crit)
        merge_sort(right_list, sort_crit)

        #i recorre la lista izquierda, j la derecha y k la lista original
        i = j = k = 0

        left_elements = size(left_list)
        righ_telements = size(right_list)

        while (i < left_elements) and (j < righ_telements):
            elem_i = get_element(left_list, i)
            elem_j = get_element(right_list, j)
            # compara y ordena los elementos
            if sort_crit(elem_j, elem_i):   # caso estricto elem_j < elem_i
                change_info(my_list, k, elem_j)
                j += 1
            else:                           # caso elem_i <= elem_j
                change_info(my_list, k, elem_i)
                i += 1
            k += 1

        # Agrega los elementos que no se comprararon y estan ordenados
        while i < left_elements:
            change_info(my_list, k, get_element(left_list, i))
            i += 1
            k += 1

        while j < righ_telements:
            change_info(my_list, k, get_element(right_list, j))
            j += 1
            k += 1
    return my_list
```

```python
def sub_list(my_list, pos, num_elem):
    """ Retorna una sub-lista de la lista recibida.

        Retorna una lista que contiene los elementos a partir de la posición ``pos``,
        con una longitud de ``num_elem`` elementos.
        Se crea una copia de dichos elementos y se retorna una lista nueva.

        :param my_list: La lista origen
        :type my_list: single_linked_list
        :param pos: Posición del primer elemento
        :type pos: int
        :param num_elem: Posición del segundo elemento
        :type pos: int

        :returns: Sub-lista de la lista original
        :rtype: single_linked_list
    """
    try:
        # raise error.FunctionNotImplemented("sublist()")
        sublst = {'elements': [],
                  'size': 0,
                  'type': 'ARRAY_LIST'}
        elem = pos-1
        cont = 1
        while cont <= num_elem:
            sublst['elements'].append(my_list['elements'][elem])
            sublst['size'] += 1
            elem += 1
            cont += 1
        return sublst
    except Exception as exp:
        error.reraise(exp, 'arraylist->subList: ')
```

En el caso del requerimiento 4 tenemos varios llamados a funciones con diferentes complejidades, estás, en orden de complejidad son:

1. New_list, fecha_srt_a_fecha_dias, size,get_element, add_last, cuya complejidad es O(1).
2. Sub_list, con complejidad O(n).
3. Merge_sort, al ser una función recursiva, cuenta con una complejidad O(N Log N).

A lo largo del algoritmo nos encontramos con 3 ciclos For independientes y una llamada a la función sub_list, lo cual hace que la complejidad parcial del algoritmo sea O(n), pero la llamada a la función merge_sort aumenta la complejidad a O(N Log N).

En conclusión, la complejidad del requerimiento 4 es O(N Log N).

7. **Requerimiento 5(**Santiago Beltran**):**

```python
def req_5(catalog, rango_presupuesto, fecha_inicial, fecha_final):
    Retorna el resultado del requerimiento 5.
    """
    filtro_presupuesto = lt.new_list()


    rango_inferior, rango_superior = map(int, rango_presupuesto.split('-'))


    for pelicula in catalog['movies']['elements']:
        if pelicula['budget'] is not None and pelicula['budget'] != "0":
            presupuesto = int(pelicula['budget'])

            if rango_inferior <= presupuesto <= rango_superior:
                lt.add_last(filtro_presupuesto, pelicula)


    fecha_inicial_mod = time.strptime(fecha_inicial, "%Y-%m-%d")
    fecha_final_mod = time.strptime(fecha_final, "%Y-%m-%d")

    filtro_final = lt.new_list()

    for elemento in filtro_presupuesto['elements']:
        if elemento['status'] == "Released":
            fecha = time.strptime(elemento["release_date"], "%Y-%m-%d")
            if fecha_inicial_mod <= fecha <= fecha_final_mod:
                lt.add_last(filtro_final, elemento)

    filtro_final = lt.quick_sort(filtro_final, compare_dates)

    for movie in filtro_final['elements']:
        if movie['budget'] is None or movie['revenue'] is None or movie['budget'] == "0" or movie['revenue'] == "0":
            movie['net_profit'] = "Undefined"
        else:
            movie['net_profit'] = float(movie['revenue']) - float(movie['budget'])

    return filtro_final
```

```python
def new_list():
    """Crea una lista implementada con un Array List vacío.

        Define una lista vacía y con un tamaño de cero

        :returns: Lista creada
        :rtype: array_list
    """
    newlist = {
        'elements': [],
        'size': 0,
    }

    # raise error.FunctionNotImplemented("new_list()")
    return newlist
```

```python
def add_last(my_list, element):
    """ Agrega un elemento en la última posición de la lista.

        Al agregar un elemento en la última posición de la lista y se incrementa el tamaño de la lista en uno.

        :param my_list: ArrayList en la que se va a insertar el elemento
        :type my_list: array_list
        :param element: Elemento a insertar
        :type element: any

        :returns: ArrayList con el elemento insertado en la última posición
        :rtype: array_list
    """
    try:
        my_list['elements'].append(element)
        my_list['size'] += 1
        # raise error.FunctionNotImplemented("add_last()")
    except Exception as exp:
        error.reraise(exp, 'arraylist->addLast: ')
```

```python
def quick_sort(my_list, sort_crit):
    """ Función de ordenamiento que implementa el algoritmo de **Quick Sort**

        Se selecciona un elemento como **pivot** y se ordenan los elementos

        Si la lista es vacía o tiene un solo elemento, se retorna la lista original.

        Dependiendo de la función de comparación, se ordena la lista de manera ascendente o descendente.

        :param my_list: Lista a ordenar
        :type my_list: array_list
        :param sort_crit: Función de comparación de elementos para ordenar
        :type sort_crit: function

        :returns: Lista ordenada
        :rtype: array_list

    """
    quick_sort_recursive(my_list, 0, size(my_list)-1, sort_crit)
    return my_list
```

```python
def quick_sort_recursive(my_list, lo, hi, sort_crit):
    """ Función recursiva que implementa el algoritmo de **quick sort**, esta es llamada por la función ``quick_sort()``

        Se localiza el **pivot**, utilizando la funcion de particion.

        Luego se hace la recursión con los elementos a la izquierda del **pivot**
        y los elementos a la derecha del **pivot**

        :param my_list: Lista a ordenar
        :type my_list: array_list
        :param lo: Posición del primer elemento
        :type lo: int
        :param hi: Posición del último elemento
        :type hi: int
        :param sort_crit: Función de comparación de elementos para ordenar
        :type sort_crit: function
    """
    if (lo >= hi):
        return
    pivot = partition(my_list, lo, hi, sort_crit)
    quick_sort_recursive(my_list, lo, pivot-1, sort_crit)
    quick_sort_recursive(my_list, pivot+1, hi, sort_crit)
```

```python
def partition(my_list, lo, hi, sort_crit):

    """ Función que implementa la partición de la lista en **quick sort**, esta es llamada por la función ``quick_sort_recursive()``

        Se selecciona un **pivot** y se ordenan los elementos menores a la izquierda del **pivot**
        y los elementos mayores a la derecha del **pivot**

        :param my_list: Lista a ordenar
        :type my_list: array_list
        :param lo: Posición del primer elemento
        :type lo: int
        :param hi: Posición del último elemento
        :type hi: int
        :param sort_crit: Función de comparación de elementos para ordenar
        :type sort_crit: function

        :returns: Posición del **pivot**
        :rtype: int
    """
    follower = leader = lo
    while leader < hi:
        if sort_crit(
           get_element(my_list, leader), get_element(my_list, hi)):
            exchange(my_list, follower, leader)
            follower += 1
        leader += 1
    exchange(my_list, follower, hi)
    return follower
```

```python
def compare_dates(pelicula1,pelicula2):
    fecha1 = time.strptime(pelicula1["release_date"], "%Y-%m-%d")
    fecha2 = time.strptime(pelicula2["release_date"], "%Y-%m-%d")

    return fecha1 > fecha2
```

En el caso del requerimiento 5, usamos algunas funciones auxiliares como lo son new_list y add_last, cuya complejidad es O(1).

Adicionalmente, usamos la función strptime de la librería time, cuya complejidad es O(1) ya que la longitud de las fechas es constante, y la función map nativa de python, que cuenta con complejidad O(n).

Finalmente, el llamado a la función quick_sort, en el peor de los casos tiene una complejidad O(N Log N).

El algoritmo del requerimiento 5 encontramos 5 ciclos For independientes, en cuya estructura no tenemos llamados a funciones adicionales que aumenten la complejidad del algoritmo.

En conclusión, la complejidad del algoritmo es dada por la función quick_sort, es decir, O(N Log N).

8. **Requerimiento 6(**Santiago Beltran**):**

```python
def req_6(catalog, idioma, año_inicial, año_final):
    """
    Retorna el resultado del requerimiento 6
    """

    peliculas_idioma = mp.get(catalog['ordenado_idioma'], idioma)

    if peliculas_idioma is None:
        print("No se encontraron películas para el idioma " + idioma )
        return None

    estadisticas_por_año = {}

    for pelicula in peliculas_idioma['elements']:
        año_pelicula = time.strptime(pelicula["release_date"], "%Y-%m-%d").tm_year

        if int(año_inicial) <= año_pelicula <= int(año_final) and pelicula['status'] == "Released":

            if año_pelicula not in estadisticas_por_año:
                estadisticas_por_año[año_pelicula] = {
                    "total_peliculas": 0,
                    "total_votacion": 0,
                    "total_runtime": 0,
                    "total_ganancias": 0,
                    "mejor_pelicula": None,
                    "peor_pelicula": None,
                    "mejor_votacion": 0,
                    "peor_votacion": float('inf')
                }

            datos = estadisticas_por_año[año_pelicula]
            datos["total_peliculas"] += 1
            datos["total_votacion"] += float(pelicula['vote_average'])
            datos["total_runtime"] += float(pelicula['runtime']) if pelicula['runtime'] else 0

            if pelicula['budget'] != "0" and pelicula['revenue'] != "0":
                ganancias = float(pelicula['revenue']) - float(pelicula['budget'])
                datos["total_ganancias"] += ganancias

            if float(pelicula['vote_average']) > datos["mejor_votacion"]:
                datos["mejor_pelicula"] = pelicula['title']
                datos["mejor_votacion"] = float(pelicula['vote_average'])

            if float(pelicula['vote_average']) < datos["peor_votacion"]:
                datos["peor_pelicula"] = pelicula['title']
```

```
            datos["peor_votacion"] = float(pelicula['vote_average'])

resultado = lt.new_list()
for año, datos in estadisticas_por_año.items():
    if datos["total_peliculas"] > 0:
        promedio_votacion = datos["total_votacion"] / datos["total_peliculas"]
        promedio_runtime = datos["total_runtime"] / datos["total_peliculas"]

        lt.add_last(resultado,{
            "año": año,
            "total_peliculas": datos["total_peliculas"],
            "promedio_votacion": promedio_votacion,
            "promedio_runtime": promedio_runtime,
            "total_ganancias": datos["total_ganancias"],
            "mejor_pelicula": datos["mejor_pelicula"],
            "mejor_votacion": datos["mejor_votacion"],
            "peor_pelicula": datos["peor_pelicula"],
            "peor_votacion": datos["peor_votacion"]
        })


resultado = lt.quick_sort(resultado, compare_years)

return resultado
```

```
def get(my_map, key):
    for i in range(len(my_map["table"]["elements"])):
        entry = my_map['table']["elements"][i]

        if entry['key'] == key:
            return entry["value"]
```

```
def new_list():
    """Crea una lista implementada con un Array List vacío.

       Define una lista vacía y con un tamaño de cero

       :returns: Lista creada
       :rtype: array_list
    """
    newlist = {
        'elements': [],
        'size': 0,
    }

    # raise error.FunctionNotImplemented("new_list()")
    return newlist
```

```python
def add_last(my_list, element):
    """ Agrega un elemento en la última posición de la lista.

        Al agregar un elemento en la última posición de la lista y se incrementa el tamaño de la lista en uno.

        :param my_list: ArrayList en la que se va a insertar el elemento
        :type my_list: array_list
        :param element: Elemento a insertar
        :type element: any

        :returns: ArrayList con el elemento insertado en la última posición
        :rtype: array_list
    """
    try:
        my_list['elements'].append(element)
        my_list['size'] += 1
        # raise error.FunctionNotImplemented("add_last()")
    except Exception as exp:
        error.reraise(exp, 'arraylist->addLast: ')
```

```python
def quick_sort(my_list, sort_crit):
    """ Función de ordenamiento que implementa el algoritmo de **Quick Sort**

        Se selecciona un elemento como **pivot** y se ordenan los elementos

        Si la lista es vacía o tiene un solo elemento, se retorna la lista original.

        Dependiendo de la función de comparación, se ordena la lista de manera ascendente o descendente.

        :param my_list: Lista a ordenar
        :type my_list: array_list
        :param sort_crit: Función de comparación de elementos para ordenar
        :type sort_crit: function

        :returns: Lista ordenada
        :rtype: array_list

    """
    quick_sort_recursive(my_list, 0, size(my_list)-1, sort_crit)
    return my_list
```

```python
def quick_sort_recursive(my_list, lo, hi, sort_crit):
    """ Función recursiva que implementa el algoritmo de **quick sort**, esta es llamada por la función ``quick_sort()``

        Se localiza el **pivot**, utilizando la funcion de particion.

        Luego se hace la recursión con los elementos a la izquierda del **pivot**
        y los elementos a la derecha del **pivot**

        :param my_list: Lista a ordenar
        :type my_list: array_list
        :param lo: Posición del primer elemento
        :type lo: int
        :param hi: Posición del último elemento
        :type hi: int
        :param sort_crit: Función de comparación de elementos para ordenar
        :type sort_crit: function
    """
    if (lo >= hi):
        return
    pivot = partition(my_list, lo, hi, sort_crit)
    quick_sort_recursive(my_list, lo, pivot-1, sort_crit)
    quick_sort_recursive(my_list, pivot+1, hi, sort_crit)
```

```python
def partition(my_list, lo, hi, sort_crit):

    """ Función que implementa la partición de la lista en **quick sort**, esta es llamada por la función ``quick_sort_recursive()``

        Se selecciona un **pivot** y se ordenan los elementos menores a la izquierda del **pivot**
        y los elementos mayores a la derecha del **pivot**

        :param my_list: Lista a ordenar
        :type my_list: array_list
        :param lo: Posición del primer elemento
        :type lo: int
        :param hi: Posición del último elemento
        :type hi: int
        :param sort_crit: Función de comparación de elementos para ordenar
        :type sort_crit: function

        :returns: Posición del **pivot**
        :rtype: int
    """
    follower = leader = lo
    while leader < hi:
        if sort_crit(
            get_element(my_list, leader), get_element(my_list, hi)):
            exchange(my_list, follower, leader)
            follower += 1
        leader += 1
    exchange(my_list, follower, hi)
    return follower
```

```python
def compare_years(year_data1, year_data2):

    return year_data1['año'] > year_data2['año']
```

Para el caso del requerimiento 6, de nuevo hacemos uso de las funciones new_list y add_last, cuya complejidad es O(1).

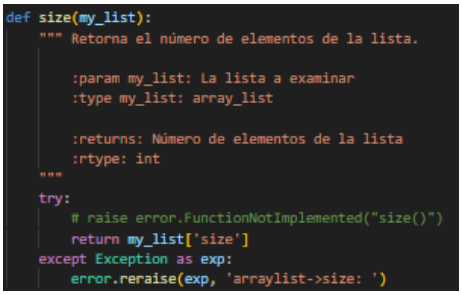
Adicionalmente, usamos la función get, con complejidad O(n).

Finalmente, como función auxiliar para ordenar la estructura usamos el algoritmo de quick_sort, cuya complejidad es O(N Log N).

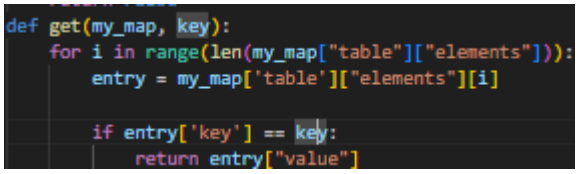
El algoritmo del requerimiento 6 tiene 2 ciclos For independientes, y dentro de ellos solo se realizan operaciones de complejidad O(1).

En conclusión, la complejidad del algoritmo es O(N Log N).

9. **Requerimiento 7(**Gerónimo Rojas**):**



```
def size(my_list):
    """ Retorna el número de elementos de la lista.

        :param my_list: La lista a examinar
        :type my_list: array_list

        :returns: Número de elementos de la lista
        :rtype: int
    """
    try:
        # raise error.FunctionNotImplemented("size()")
        return my_list['size']
    except Exception as exp:
        error.reraise(exp, 'arraylist->size: ')
```



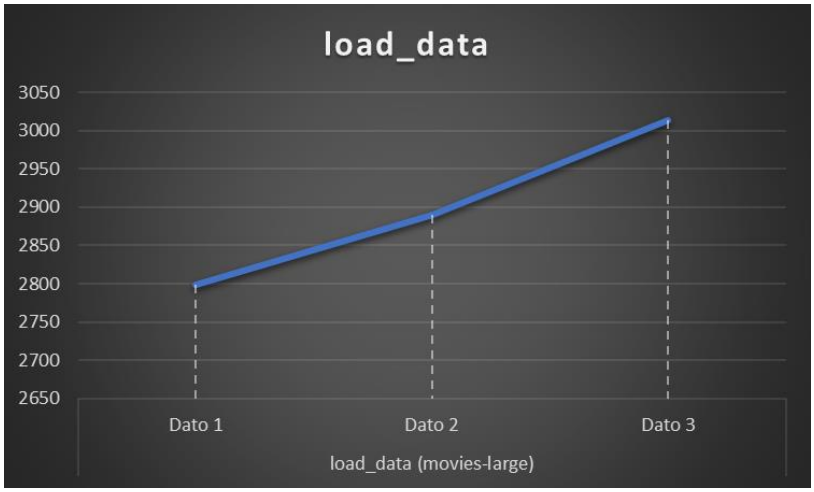
```
def get(my_map, key):
    for i in range(len(my_map["table"]["elements"])):
        entry = my_map['table']["elements"][i]

        if entry['key'] == key:
            return entry["value"]
```

```python
def req_7(catalog,productora,inicial,final):

    estadistica={}
    d=int(inicial)
    while d <= int(final):
        lista=mp.get(catalog["ordenado_año"],str(d))
        d+=1
        for k in range(0,lt.size(lista)):
         for j in range(0,lt.size(lista["elements"][k]["production_companies"])):
          if str(lista["elements"][k]["production_companies"]["elements"][j]["name"])==productora :
                        i=d-1

                        if i not in estadistica:
                           estadistica[i] = {
                           'total': 0,'votacion_prom': 0,'duracion_prom': 0,'net_profit': 0,
                           'mejor_peli': ("", float("-inf")),'peor_peli': ("", float("inf"))}
                        estadistica[i]['total'] += 1
                        estadistica[i]['votacion_prom'] += float(lista["elements"][k]["vote_average"])
                        estadistica[i]['duracion_prom'] += float(lista["elements"][k]["runtime"])
                        if ((lista["elements"][k]["revenue"]) or(lista["elements"][k]["budget"]))=="0":
                         net_profit="undefined"
                        else:
                          net_profit=int(lista["elements"][k]["revenue"])-int(lista["elements"][k]["budget"])
                        if net_profit!="undefined":
                          estadistica[i]["net_profit"]+=net_profit
                        votacion = float(lista["elements"][k]['vote_average'])
                        nombre = lista["elements"][k]["title"]
                        if votacion > estadistica[i]['mejor_peli'][1]:
                          estadistica[i]['mejor_peli'] = (nombre, votacion)
                        if votacion < estadistica[i]['peor_peli'][1]:
                          estadistica[i]['peor_peli'] = (nombre, votacion)
    estadistica_final={}
    i=int(inicial)
    fini=int(final)
    while i < fini+1:
        if i in estadistica:
          if estadistica[i]["net_profit"]==0:
              estadistica[i]["net_profit"]=="undefined"
          estadistica_final[i] = {
                'total': estadistica[i]["total"],
                'votacion_prom': estadistica[i]["votacion_prom"] / estadistica[i]["total"],
                'duracion_prom': estadistica[i]["duracion_prom"] / estadistica[i]["total"],
                'net_profit': estadistica[i]["net_profit"],
                'mejor_peli': estadistica[i]["mejor_peli"],
                'peor_peli': estadistica[i]["peor_peli"]}
        i+=1
    return estadistica_final
```

Este código utiliza funciones auxiliares como el “get” de una tabla de hash y el “size” de array list , las cuales, tienen complejidad O(n) y O(1) respectivamente. Por otro lado, el requerimiento se aproxima a una complejidad de O(n), pues, si bien este código tiene tres ciclos anidados, el primero recorre los años que hay entre el inicial y el final, y, el tercero recorre la lista de compañías de producción de cada película que fue publicada en un año especifico, por lo tanto, ambos ciclos hacen un recorrido minúsculo en comparación con el recorrido de la lista de películas por año. Por todo lo anterior, consideramos que la complejidad temporal del código se aproxima a O(n).

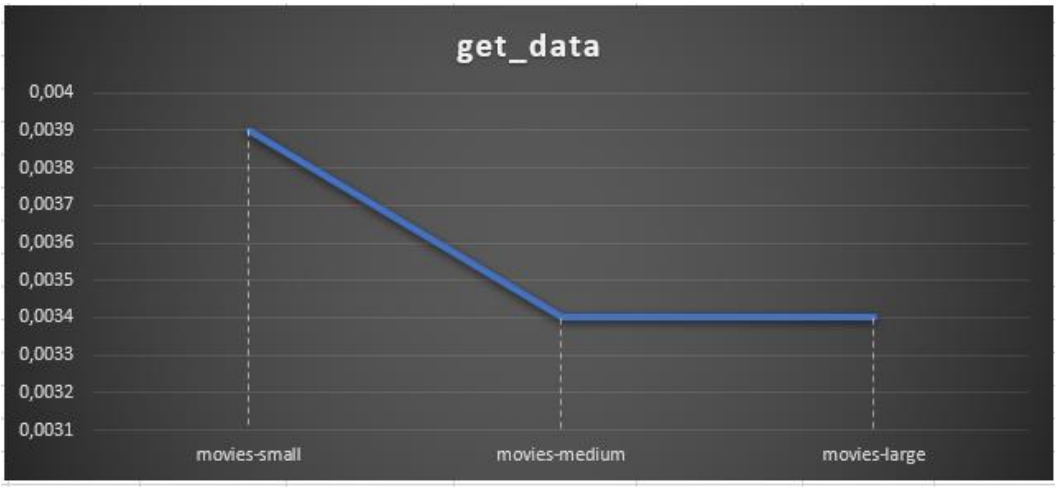
PRUEBAS DE TIEMPO DE EJECUCIÓN

load_data (movies-large)		
Dato 1	Dato 2	Dato 3
2799,047	2889,349	3013,235



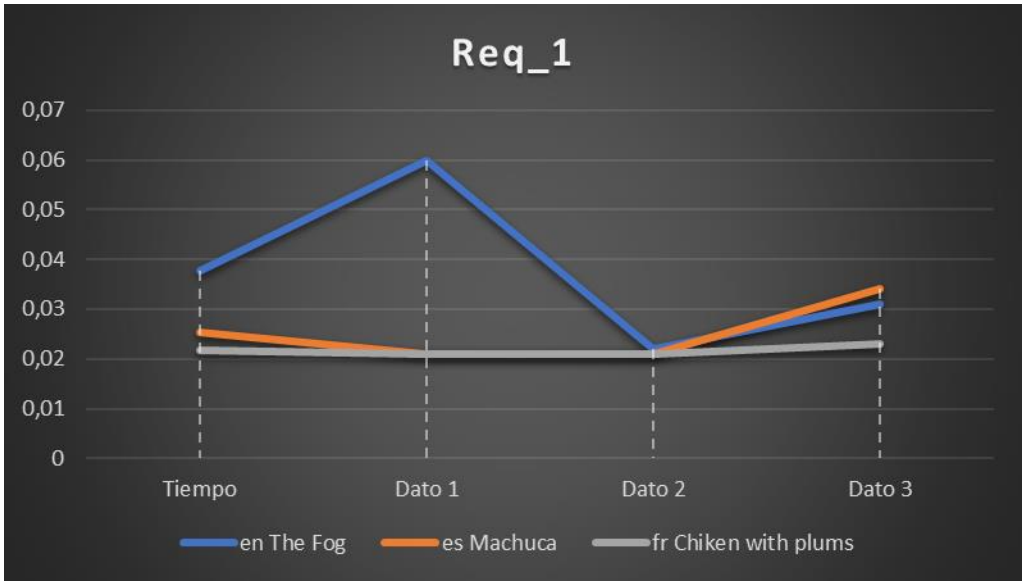
Teniendo en cuenta el análisis de complejidad de la función load_data y las pruebas de tiempo de ejecución, podemos ver que la función si muestra un comportamiento O(n).

get_data				
id	Tiempo	movies-small	movies-medium	movies-large
12	0,003566667	0,0039	0,0034	0,0034



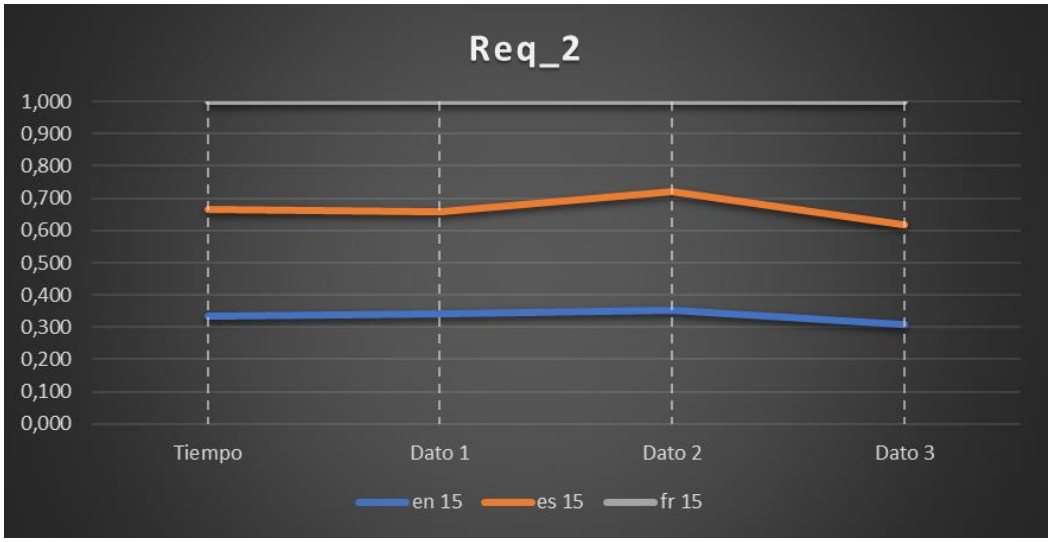
Al analizar la función get_data, podemos ver que la tendencia de comportamiento de esta es constante ya que la consulta, en este caso, se realiza con el mismo ID para los 3 archivos de películas.

req_1					
Casos	Movie_title	Tiempo	Dato 1	Dato 2	Dato 3
en	The Fog	0,03766667	0,06	0,022	0,031
es	Machuca	0,02533333	0,021	0,021	0,034
fr	Chiken with plums	0,02166667	0,021	0,021	0,023



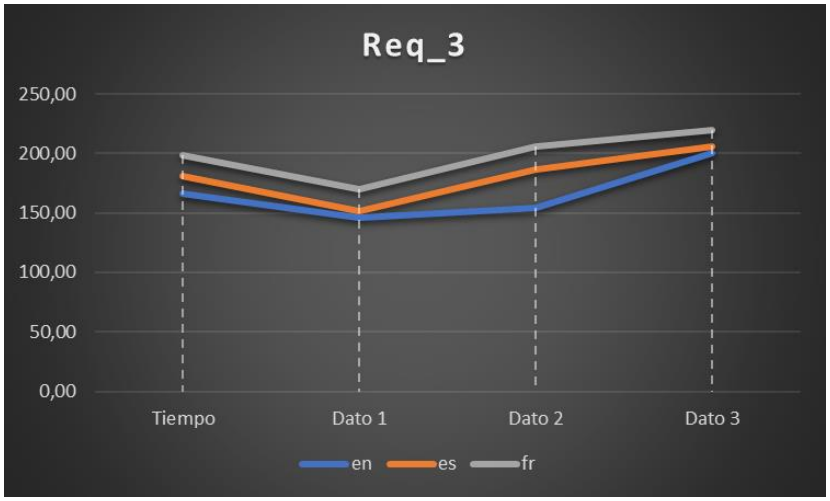
Al analizar la toma de tiempos del requerimiento 1, podemos decir que la función toma este comportamiento que se asemeja al de una función constante, pero la variación en el tamaño de los archivos puede influir en su comportamiento y asemejarla una O(n). El pico en la primera búsqueda se da ya que en este primer recorrido debe iterar sobre toda la lista hasta hallar la película.

req_2					
Casos	n	Tiempo	Dato 1	Dato 2	Dato 3
en	15	0,027666667	0,028	0,029	0,026
es	15	0,027333333	0,026	0,03	0,026
fr	15	0,027666667	0,028	0,023	0,032



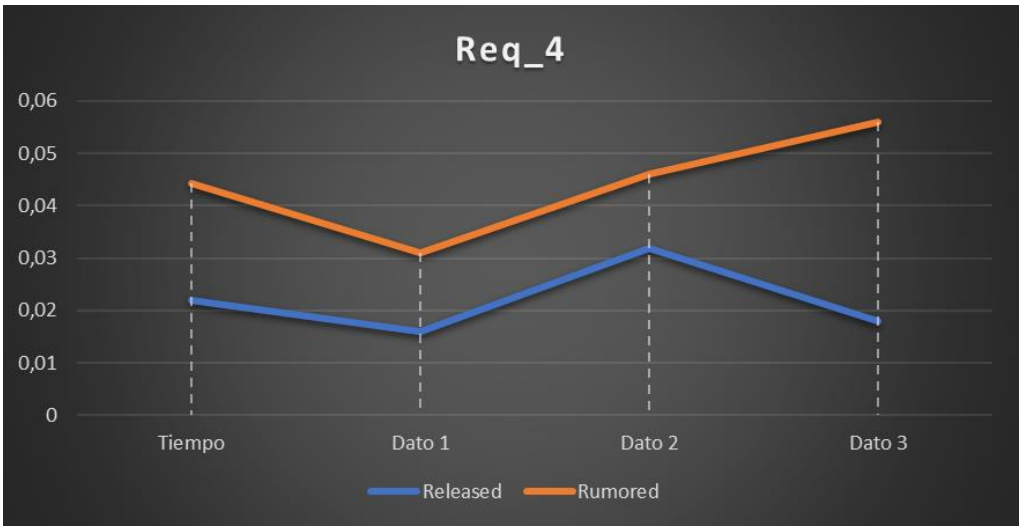
Tomando como base el análisis de complejidad y las pruebas de tiempos de ejecución, la gráfica nos muestra que el comportamiento de la función se asemeja a uno de complejidad O(N Log N).

req_3 (2001-01-01 -> 2018-01-01)				
Casos	Tiempo	Dato 1	Dato 2	Dato 3
en	166,6	145,82	153,97	200,01
es	14,75	5,89	32,25	6,11
fr	17,06333333	18,05	19,27	13,87



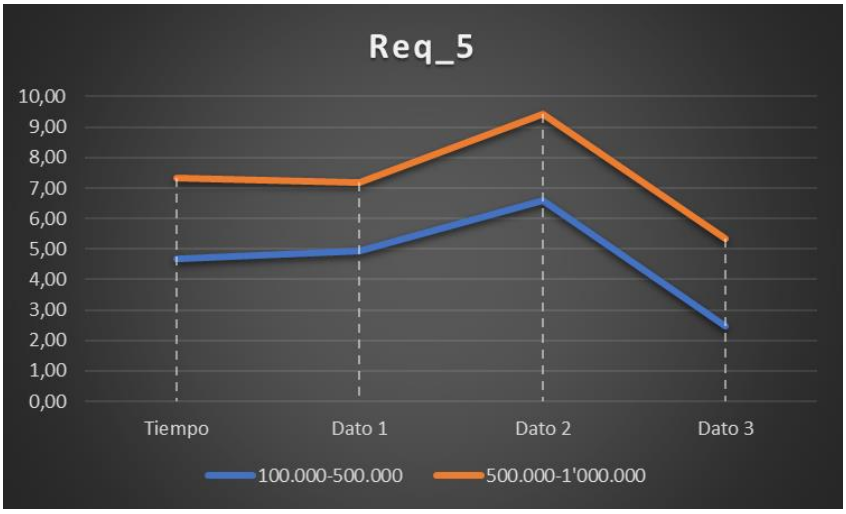
La gráfica nos muestra claramente que la función tiende hacia N Log N, lo cual concuerda con el análisis de complejidad del requerimiento. Las variaciones se deben a la cantidad de datos procesados.

req_4 (2001-01-01 -> 2018-01-01)				
Casos	Tiempo	Dato 1	Dato 2	Dato 3
Released	0,022	0,016	0,032	0,018
Rumored	0,022333333	0,015	0,014	0,038



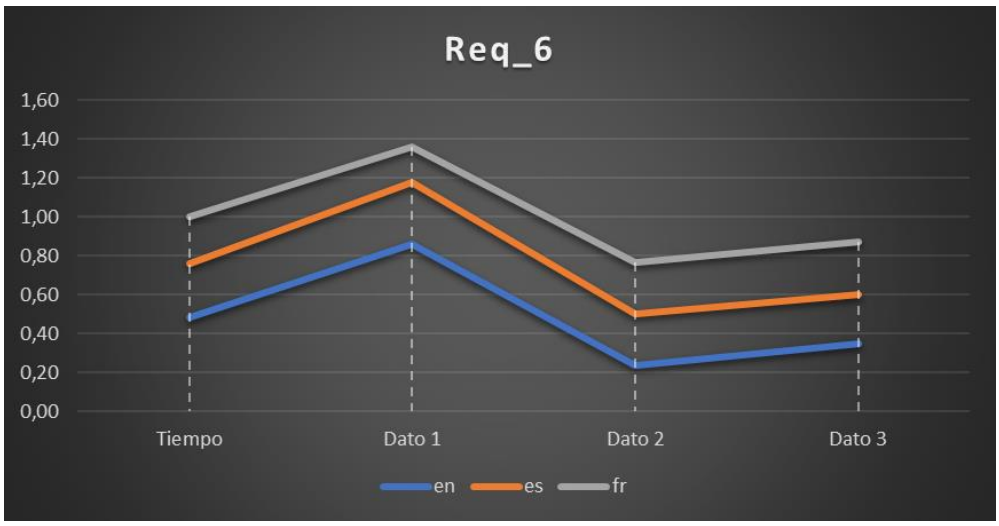
La grafica nos demuestra que la función implementada definitivamente es O(n log n), tal como se tenía previsto al momento de realizar el análisis de complejidad, haciendo que de esta forma se obtenga el resultado que deseábamos. A pesar de esto se ven unas variaciones, debido a la cantidad de datos que se revisan.

req_5 (2001-01-01 -> 2018-01-01)				
Casos	Tiempo	Dato 1	Dato 2	Dato 3
100.000-500.000	4,666666667	4,95	6,58	2,47
500.000-1'000.000	2,656666667	2,25	2,86	2,86



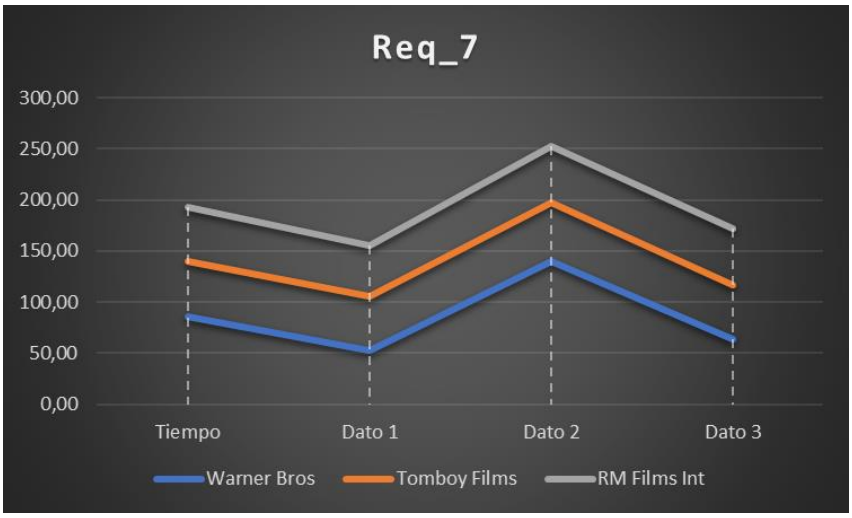
La grafica nos muestra que el comportamiento de la función se asemeja a O(n), como se había previsto teóricamente, ya que, tiene forma semejante a una función lineal, sin embargo, se presentan ciertas discontinuidades en la gráfica relativas a los datos.

req_6 (2001-> 2018)				
Casos	Tiempo	Dato 1	Dato 2	Dato 3
en	0,482666667	0,86	0,238	0,35
es	0,276666667	0,32	0,26	0,25
fr	0,24	0,18	0,27	0,27



La grafica nos muestra que el comportamiento de la función se asemeja a O(n), como se había previsto teóricamente, ya que, tiene forma semejante a una función lineal, sin embargo, se presentan ciertas discontinuidades en la gráfica relativas a los datos.

req_7 (2001 -> 2018)				
Casos	Tiempo	Dato 1	Dato 2	Dato 3
Warner Bros	85,48333333	53,11	139,48	63,86
Tomboy Films	54,5	52,87	58,33	52,3
RM Films Int	53,43666667	49,91	54,8	55,6



La grafica nos muestra que el comportamiento de la función se asemeja a O(n), como se había previsto teóricamente, ya que, tiene forma semejante a una función lineal, sin embargo, se presentan ciertas discontinuidades en la gráfica relativas a los datos.