ANÁLISIS DEL RETO

Estudiante 1 - Jessica Sofía Garay Acosta, 202310514, js.garay@uniandes.edu.co

Estudiante 2 - María Lucía Benavides, 202313423, m.benavidesd@uniandes.edu.co

Estudiante 3 - Daniel Mancilla Triviño, 202221038, d.mancilla@uniandes.edu.co

Requerimiento <<1>>

Descripción

```python
def req_1(catalog, n, equipo, condicion):
    """
    Función para obtener los ultimos N partidos jugados por un equipo según su condición

    Args:
        catalog (dict): Catálogo con todas las estructuras que organizan la información de partidos
        n (int): El número de N partidos a consultar
        equipo (str): Nombre del equipo a consultar
        condicion (str): Condición del equipo a consultar

    Returns:
        lista_partidos (ARRAY_LIST): La lista ordenada de los partidos que jugó el equipo con la condición
        totales (tuple): Tupla que cuenta con el número de equipos y el número de partidos
    """
    equipos = catalog['equipos']
    equipo_buscado = mp.get(equipos, equipo)
    part = me.getValue(equipo_buscado)
    condicion_n = mp.get(part['partidos'], condicion)
    #Se obtienen los partidos como un single_linked el cual se organizará
    lista_partidos = me.getValue(condicion_n)
    totales = (mp.size(equipos), part['total_partidos'], lt.size(lista_partidos))
    #Sacamos la sublista y la comparamos según los criterios
    lista_partidos= merg.sort(lista_partidos, comparePartidos)
    if n > lt.size(lista_partidos):
        sub_lista = lista_partidos
    else:
        sub_lista = lt.subList(lista_partidos, 1, n)
    return (sub_lista, totales)
```

Este requerimiento toma un mapa de equipos y saca el equipo del mapa. Este mapa contiene 3 mapas por cada equipo, divididos en 'home', 'away' y 'ind'. Se sacará la lista según la condición que se busca y mientras tanto se irán buscando los totales generales que se utilizarán para especificaciones del view. Luego de sacar la lista de la condición, se organizará con mergeSort y se hará una sublista, según lo pertinente, para sacar el top que pide el usuario. De esta manera, junto al catálogo, el top, la condición y el equipo sacamos una lista con el top y los totales.

Entrada	- Catálogo - Equipo solicitado

	- Número de partidos jugados que se desea conocer - Condición del equipo
Salidas	- Sublista ordenada de los n partidos que se desean conocer - Total de partidos - Total de equipos disponibles - Total de partidos jugados por ese equipo en la condición requerida.
Implementado (Sí/No)	Si. Grupal.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación variable “equipos”	O (1)
Retorno de la pareja llave – valor a partir del nombre del goleador en la variable “equipo_buscado”	O(N)
Asignación variable “part” a partir del valor proveniente de la llave (getValue)	O (1)
Retorno de la pareja llave – valor a partirla variable "part“, esto se asigna a la variable “condicion_n”	O (1)
Asignación variable “lista_partidos” a partir de la condición_n anterior	O (1)
Asiganción de la variable “totales” a partir del tamaño del mapa “equipos”	O (1)
Merge sort de la lista de partidos	O(NlogN)
Creación de sublista a partir de los n partidos requeridos y el tamaño de la lista_partidos	O (1)
TOTAL	***O(NlogN)***

Pruebas Realizadas

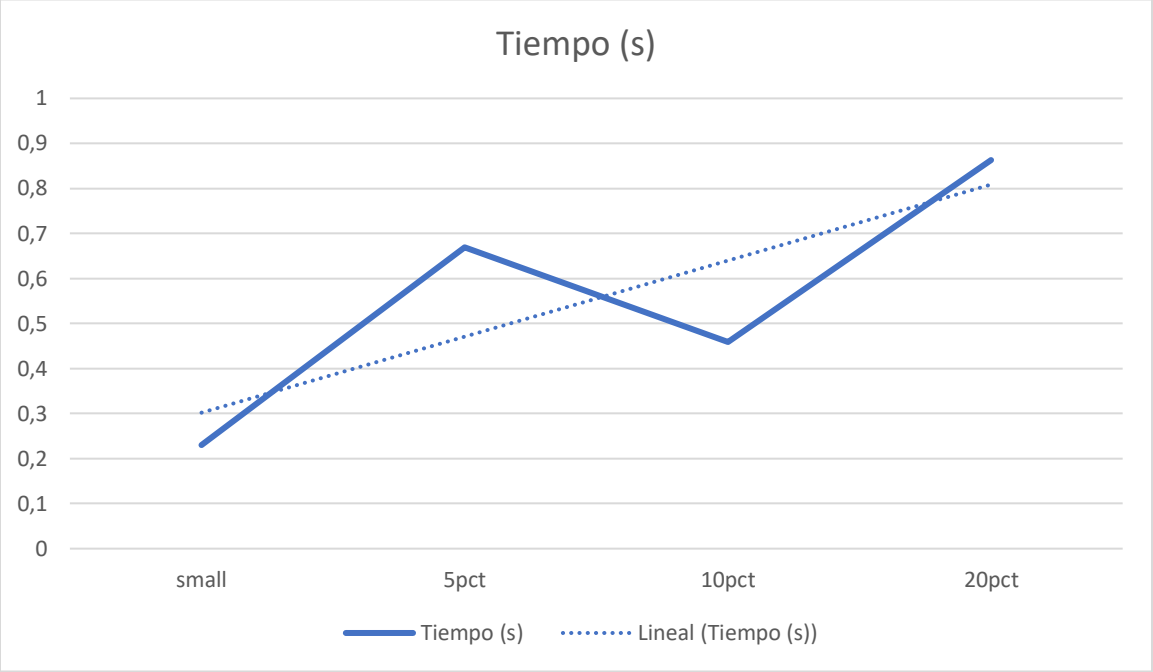
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

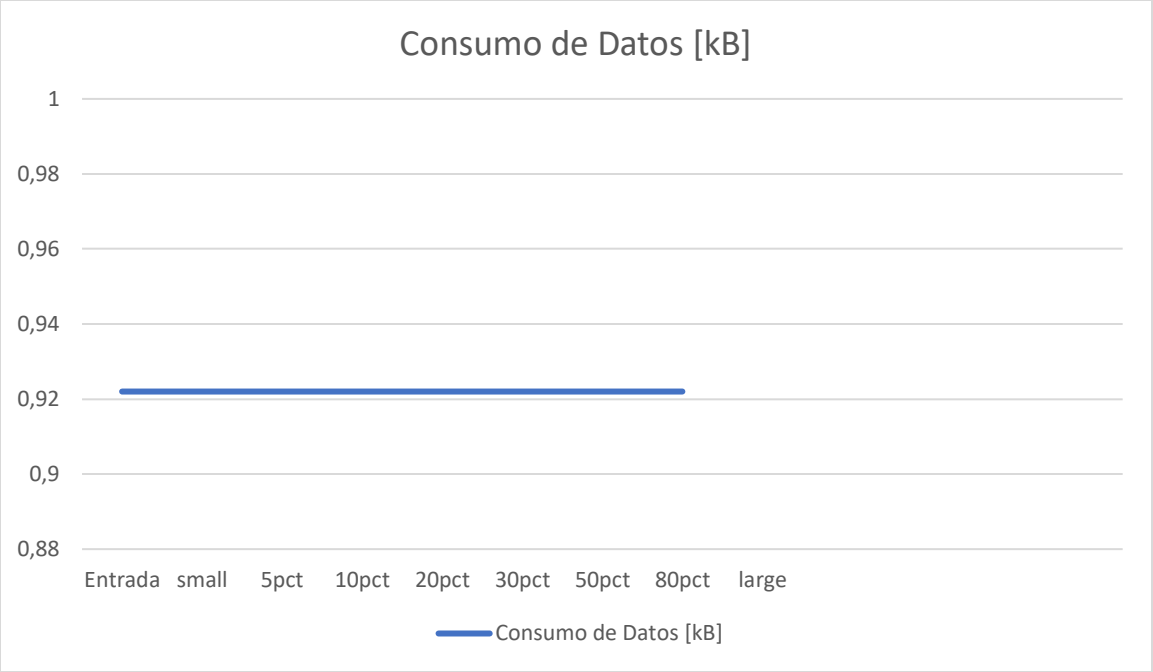
Entrada		**Tiempo (s)**	**Consumo de Datos [kB]**
small		0.230	0.922
5pct		0.669	0.922

10pct		0.459	0.922
20pct		0.863	0.922
30pct		1.190	0.922
50pct		2.954	0.922
80pct		3.359	0.922
large		4.112	0.922

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

La complejidad que se encuentra dentro de este requerimiento es concorde a lo planteado gracias a que según los resultados de la complejidad temporal encontramos un cambio de tipo NlogN y un uso de memoria constante a lo largo de toda la ejecución.

Realizando un acercamiento al reto pasado, se puede ver una diferencia notable dentro de los tiempos de ejecución:

RETO 1	**RETO 2**
0,48	0.230
0,5	0.669
3,71	0.459
10,81	0.863
16,64	1.190
30,93	2.954
32,97	3.359
36,24	4.112

Requerimiento <<2>>

Descripción

```python
def req_2(catalog, goleador_nom, n):
    """
    Función para obtener los primeros N goles anotados por un jugador especifico

    Args:
        catalog (dict): Catálogo con todas las estructuras que organizan la información de partidos
        goleador_nom (str): Nombre del goleador a consultar
        n (int): El número de n goles anotados por el jugador

    Returns:
        resultado (ARRAY_LIST): La lista de los top partidos de ese jugador
        suma (int): El número total de jugadores
        total (int): El número total de partidos
        penal (int): El número total de penales
    """

    jugadores = catalog['goleadores']
    suma = mp.size(jugadores)
    #total de jugadores = size del mapa
    jugador_seleccionado = mp.get(jugadores, goleador_nom)
    if jugador_seleccionado:
        partidos = me.getValue(jugador_seleccionado)['info']

    total = lt.size(partidos)

    if n >= total:
        resultado = lt.subList(partidos, 1, total)
    else:
        resultado = lt.subList(partidos, 1, n)

    penal = me.getValue(jugador_seleccionado)['total_pen']
    return resultado, suma, total, penal
```

Dentro de esta función, se toma el mapa de goleadores para poder tomar el jugador que busca el usuario y luego tomar la lista de partidos que ha jugado este jugador por medio del getValue. Luego de esto se realizará una sublista para poder sacar el top de partidos que busca el usuario. Además de estas operaciones se realizará la obtención de los totales que se presentarán en el view. De esta manera, con el catálogo, el nombre del goleador y el número para el top, se sacará una lista de partidos y los totales necesarios.

Entrada	- Catálogo con todas las estructuras de datos del reto - Nombre del jugador - Los “n” partidos que se desean buscar
Salidas	- La lista de las “n” anotaciones con su respectiva información ordenada por fecha y minuto. - El total de anotaciones del jugador encontradas. - El total de penaltis del jugador
Implementado (Sí/No)	Sí. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación variable “jugadores”	O (1)
Asignación variable “suma”, equivalente al tamaño del mapa “jugadores”	O (1)
Retorno de la pareja llave – valor a partir del nombre del goleador	O(N)
Asignación variable “partidos” a partir del valor proveniente de la llave (getValue)	O (1)
Tamaño de la lista partidos asignada a la variable “total”	O (1)
Creación de la sublista a partir de los n partidos solicitados	O (1)
TOTAL	**O(N)**

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Para elaborar las pruebas, se mantuvo constante el equipo que entraba por parámetro y para evidenciar mejores resultados se tomó como referencia al jugador con una mayor cantidad de anotaciones (Cristiano Ronaldo) y a medida que aumentaba la cantidad de datos también lo hacia las “N” que se pedían.

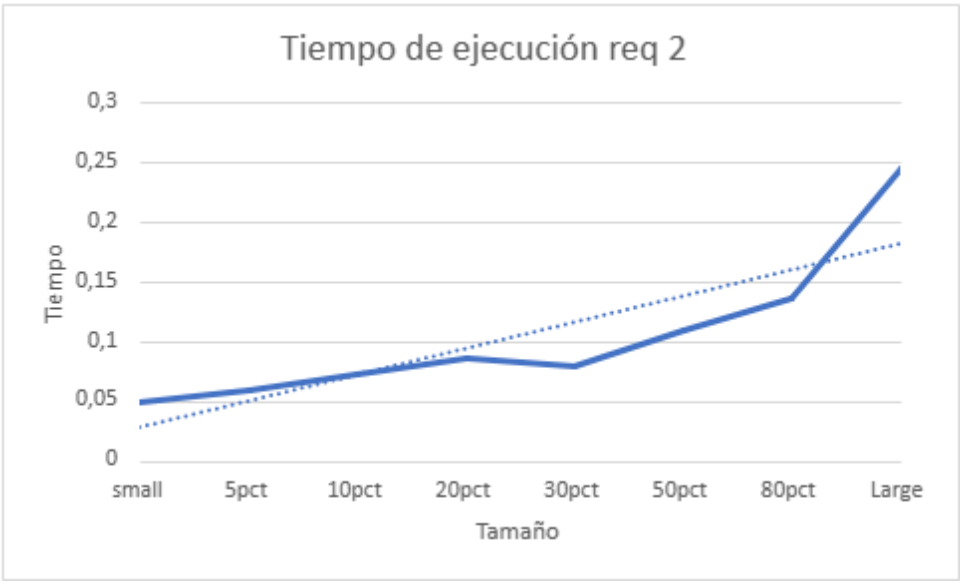
Procesadores	11th Gen Intel(R) Core(TM) i7-11375H @ 3.30GHz 3.30 GHz
Memoria RAM	16 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador x64 Windows 11

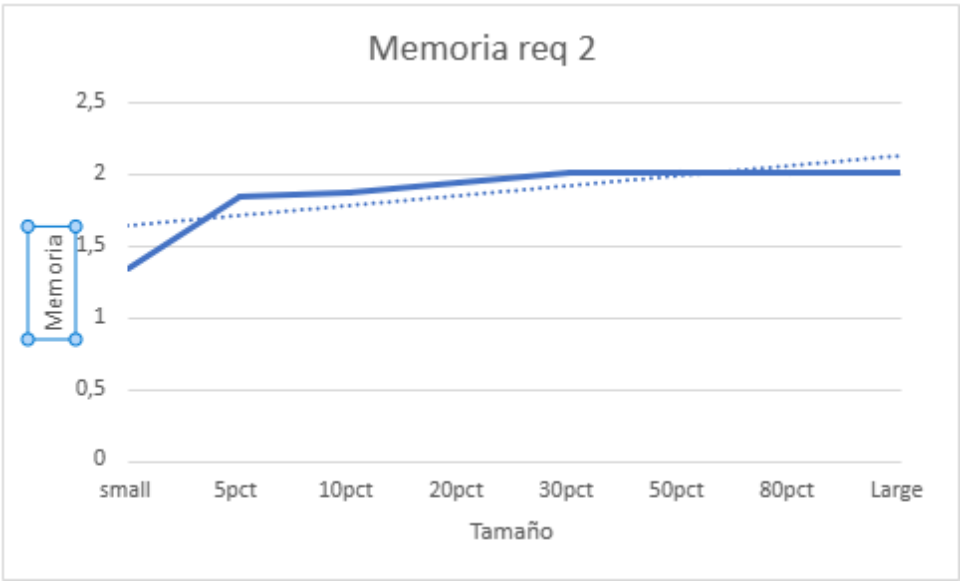
Tablas de datos

Entrada	**Tiempo (s)**	**Consumo de Datos [kB]**
small	0.05	1.352
5pct	0.06	1.852
10pct	0.073	1.883
20pct	0.087	1.945

30pct	0.08	2.016
50pct	0.11	2.016
80pct	0.136	2.016
Large	0.245	2.016

Graficas





Comparación Reto 1 y Reto 2

Tamaño	Tiempo en Reto 1	Tiempo en Reto 2
small	6,26	0.05
5 pct	45,98	0.06
10 pct	76,26	0.073
20 pct	162,84	0.087
30 pct	288,12	0.08
50 pct	423,64	0.11
80 pct	738,22	0.136
large	1035,38	0.245

Se puede evidenciar un cambio bastante significativo en el tiempo de ejecución, siendo el reto 2 mucho más eficaz que el reto 1.

Análisis

Se evidencia que este requerimiento tiene un comportamiento lineal lo cual se relaciona con la complejidad mencionada anteriormente O(N), entre mayor sea la cantidad de datos, mayor será el tiempo.

Por otro lado, respecto a la memoria, se evidencia que se mantiene constante, o tiene cambios muy pequeños según aumenta la cantidad de datos.

Requerimiento <<3>>

Descripción

```python
def req_3(catalog, equipo, fi, ff):
    """
    Función para obtener los partidos que disputo un equipo utilizando su nombre
    y un periodo entre dos fechas especificadas

    Args:
        catalog (dict): Catálogo con todas las estructuras que organizan la información de partidos
        equipo (str):  Nombre del equipo a consultar
        fecha_i (datetime): Fecha inicial para la búsqueda
        fecha_f (datetime): Fecha final para la búsqueda

    Returns:
       final (ARRAY_LIST): La lista con los partidos que disputó el equipo
    """

    partidos = catalog['partidos']
    equipos = catalog['equipos']
    total_equ= mp.size(equipos)
    lista = lt.newList("ARRAY_LIST")
    home_games=0
    away_games=0
    for partido in lt.iterator(partidos):
            fecha= partido['date']
            if partido['home_team'] == equipo and fi<=fecha<=ff:
                lt.addLast(lista, partido)
                home_games+=1
            elif partido['away_team'] == equipo and fi<=fecha<=ff:
                lt.addLast(lista, partido)
                away_games+=1
    numero_par = lt.size(lista)
    lista = merg.sort(lista, comparePartidos)
    return lista, numero_par, home_games, away_games,total_equ
```

Esta función toma el mapa de equipos y se seleccionará el equipo del mapa. Luego de esto se filtrará los partidos del equipo dentro de las fechas otorgadas por el usuario. Además, por toda la función se sacarán los diferentes totales que son necesarios para el view. De esta manera, junto con el catálogo, el nombre del equipo y un intervalo de fechas, llegamos a una lista de partidos la cual cuenta con los partidos del equipo en la fecha y los diferentes totales para presentar en el view.

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Sí. Daniel Mancilla

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación de la variable “partidos “(lista con todos los partidos con la información de las 3 estructuras de datos)	O (1)
Asignación de la variable “equipos” (mapa con los equipos)	O (1)

Variable “total_equ”que contiene el tamaño del mapa de equipos, para mostrarlo posteriormente en el print del requerimiento	O (1)
Variable “lista” = Array_list vacío, el cual se va a usar para almacenar la información que va a ser impresa por el tabulate	O (1)
Variables home y away= donde se almacenan los contadores de los partidos visitantes y locales del respectivo equipo	O (1)
Ciclo for en el cual se itera cada partido en la lista “partidos” y a través de condicionales se evalúa la condición(visitante o local) para poder sumarlas a los contadores de away y home, además de la condición del rango de tiempo. Si ambas se cumples finalmente se añade ese partido a la nueva lista creada	O (N)
Se le saca el size a la última lista creada para tener la variable, “numero_par”, del número de partidos disputados por ese equipo en dicho rango de tiempo	O (1)
Se modifica la lista en orden de mostrar primero los partidos más recientes	O(NlogN)
TOTAL	***O(NlogN)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Especificaciones del computador donde se realizaron las pruebas

- 8th Gen Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz 2.21 GHz

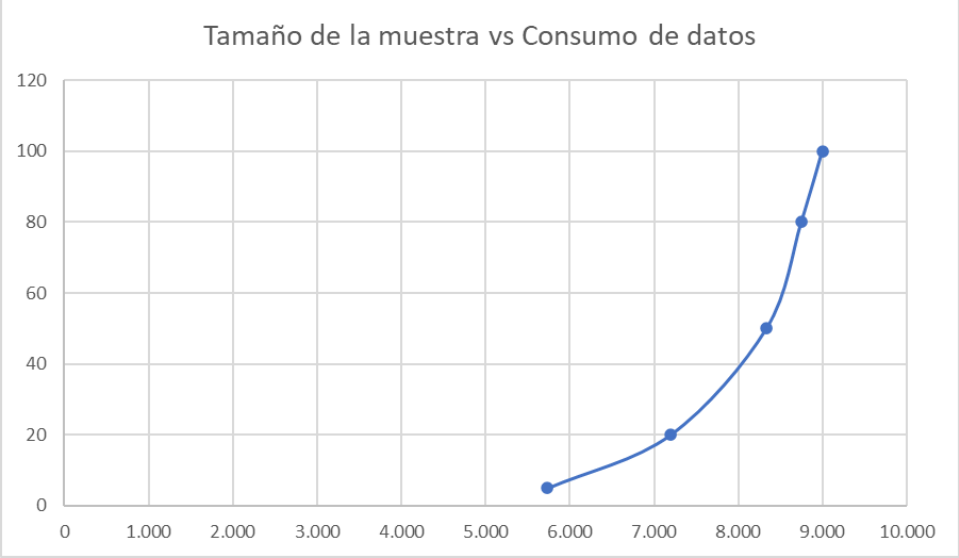
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada: Colombia, 2000-02-01, 2020-11-21	**Tiempo (s)**	**Consumo de Datos [kB]**
Small	1.445	5.734
20pct	15.928	7.195
50pct	33.471	8.336
80pct	40.296	8.750
Large	53.044	9.000

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Respecto al reto nivel 1 podemos identificar que la complejidad disminuye significativamente debido a que a través de los mapas es mucho más fácil realizar recorridos y extraer los datos necesarios para la realización del tabúlate

Requerimiento <<4>>

Descripción

```python
def req_4(catalog, torneo, fecha_i, fecha_f):
    """
    Función para obtener los partidos relacionados con un torneo utilizando
    su nombre y un periodo entre dos fechas especificadas.

    Args:
        catalog (dict): Catálogo con todas las estructuras que organizan la información de partidos
        torneo (str): _description_
        fecha_i (datetime): Fecha inicial para la búsqueda
        fecha_f (datetime): Fecha final para la búsqueda

    Returns:
        torneo_ff (ARRAY_LIST): Una lista ordenada de los partidos del torneo en el periodo de tiempo
        totals (map): Un mapa con todos generales de los partidos
    """
    torneos = catalog['torneos']
    total_torneos = mp.size(torneos)
    torneo_buscado = mp.get(torneos, torneo)
    lista = me.getValue(torneo_buscado)
    #Con la lista de partidos, hallamos los partidos que se encuentran en ese periodo y sus totales
    torneo_ff, totals = get_timeframe(lista['partidos'], fecha_i, fecha_f)
    mp.put(totals, 'total_torneos', total_torneos)

    return merg.sort(torneo_ff, compare_date_name_city), totals
```

Para obtener una lista ordenada de partidos relacionados con un torneo específico en un período de tiempo definido. Para lograr esto, la función accede a un catálogo que las estructuras de partidos y busca el torneo deseado. Luego, obtiene la lista de partidos asociados con ese torneo. Utiliza una función llamada get_timeframe para filtrar los partidos que se encuentran dentro del rango de fechas especificado, desde fecha_i hasta fecha_f. Los partidos que cumplen con este criterio se almacenan en torneo_ff, que es una lista ordenada de partidos. Además, la función calcula totales relacionados con los partidos, como la cantidad total de partidos, la cantidad de autogoles, los países en los que se jugaron los partidos, las ciudades de los partidos, los equipos ganadores y la cantidad de equipos en el torneo. Estos totales se almacenan en un mapa llamado totals. Finalmente, la función retorna torneo_ff y totals, proporcionando así una lista de partidos ordenada y totales generales relacionados con los mismos.

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Sí. María Lucía Benavides

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación y acceso de datos con parametros	O (1)
Llamar a la función get_timeframe	O(N)
Ordenamiento con merg.sort	O(NlogN)
Inserción dentro del mapa	O (1)
TOTAL	***O(NlogN.)***

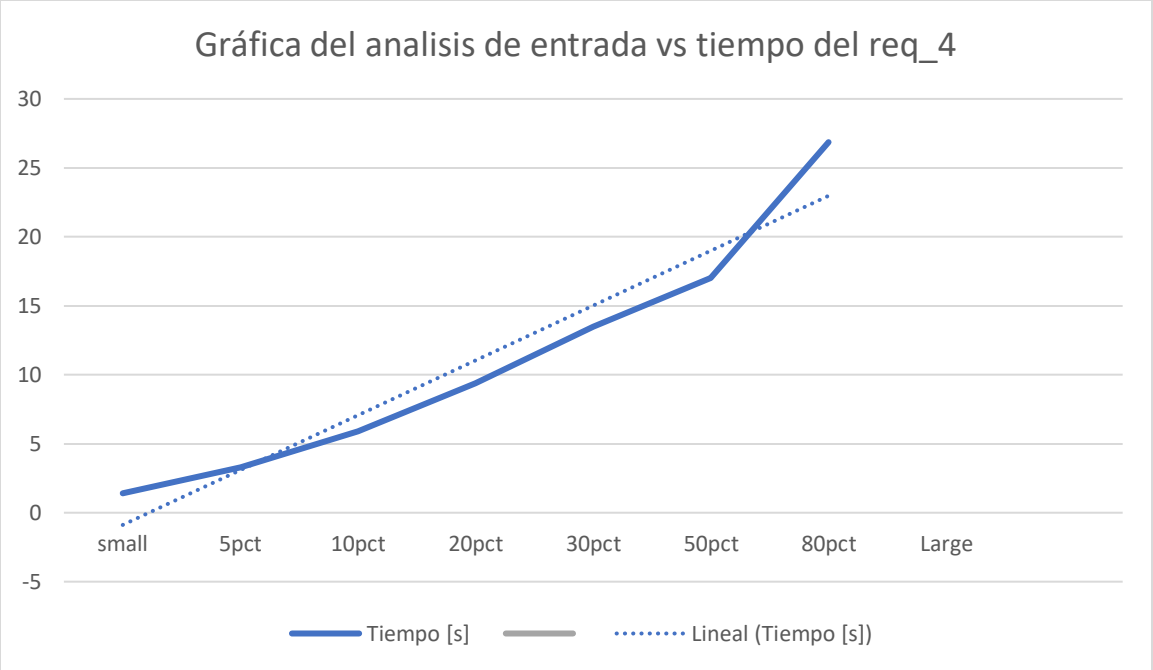
Pruebas Realizadas

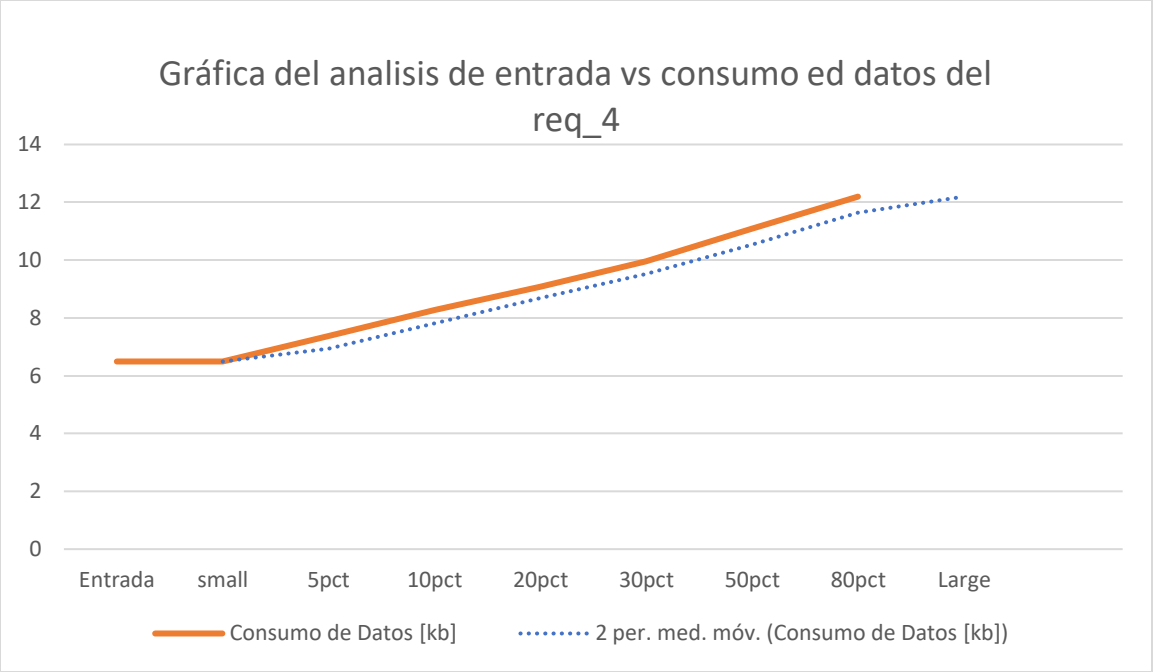
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (s)	Consumo de Datos [kB]
small	1.407	6.492
5pct	3.304	6.492
10pct	5.887	7.367
20pct	9.382	8.273
30pct	13.464	9.086
50pct	17.008	9.961
80pct	26.863	11.086
Large	26.234	12.195

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

Según las gráficas podemos entender el aumento de complejidad que se explica gracias a las diferentes funciones utilizadas y aplicadas dentro del código. Tomando en cuenta lo resultados, podemos asumir un carácter lineal dentro del requerimiento demostrado, que expresa el comportamiento que se entendió dentro del análisis de complejidad gracias a su aproximación logarítmica.

RETO 1	**RETO 2**
16,77	1.407
125,66	3.304
246,81	5.887
429,41	9.382
611,84	13.464
732,72	17.008
960,36	26.863
1066,91	26.234

Requerimiento <<5>>

Descripción

```python
def req_5(catalog, goleador_nom, fecha_i, fecha_f):
    """
    Función que soluciona el requerimiento 5
    """
    jugadores = catalog['goleadores']
    penalty = 0
    auto = 0
    torneos = lt.newList('ARRAY_LIST')
    suma = mp.size(jugadores)
    jugador_seleccionado = mp.get(jugadores, goleador_nom)
    lista = lt.newList('SINGLE_LINKED')
    if jugador_seleccionado:
        partidos = me.getValue(jugador_seleccionado)['info']
        for valor in range(1, lt.size(partidos) + 1):
            golea = lt.getElement(partidos, valor)
            if fecha_i <= golea['date'] <= fecha_f:
                lt.addFirst(lista, golea)
            if golea['penalty'] == 'True':
                penalty += 1
            if golea['own_goal'] == 'True':
                auto += 1
            if lt.isPresent(torneos, golea['tournament']) == 0:
                lt.addLast(torneos, golea['tournament'])

    total_torneos = lt.size(torneos)

    total = lt.size(partidos)

    return lista, suma, total, total_torneos, penalty,auto
```

Dentro de esta función, se toma el mapa de goleadores para poder tomar el jugador que busca el usuario y luego tomar la lista de partidos que ha jugado este jugador por medio del getValue. Luego se analizarán los partidos que estén en el rango de fechas asignado por el usuario. Además de estas operaciones se realizará la obtención de los totales que se presentarán en el view. De esta manera, con el catálogo, el nombre del goleador y el número para el top, se sacará una lista de partidos y los totales necesarios.

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Para elaborar las pruebas, se mantuvo constante la fecha inicial y la final que entraban por parámetro (fecha incial: 2006-05-14, fecha final: 2023-04-21) y para evidenciar mejores resultados se hicieron varias pruebas con tres goleadores famosos (Cristiano Ronaldo, Messi, Neymar)

Procesadores	11th Gen Intel(R) Core(TM) i7-11375H @ 3.30GHz 3.30 GHz
Memoria RAM	16 GB
Sistema Operativo	Sistema operativo de 64 bits, procesador x64 Windows 11

Tabla de datos

Entrada	- Catálogo con todas las estructuras de datos del reto - Nombre del jugador - Fecha inicial - Fecha Final
Salidas	- La lista de los goles encontrados - El total de jugadores disponibles - El total de goles del jugador - El total de torneos en los que participó el jugador - El total de penaltis. - El total de autogoles.
Implementado (Sí/No)	Sí. Jessica Garay

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

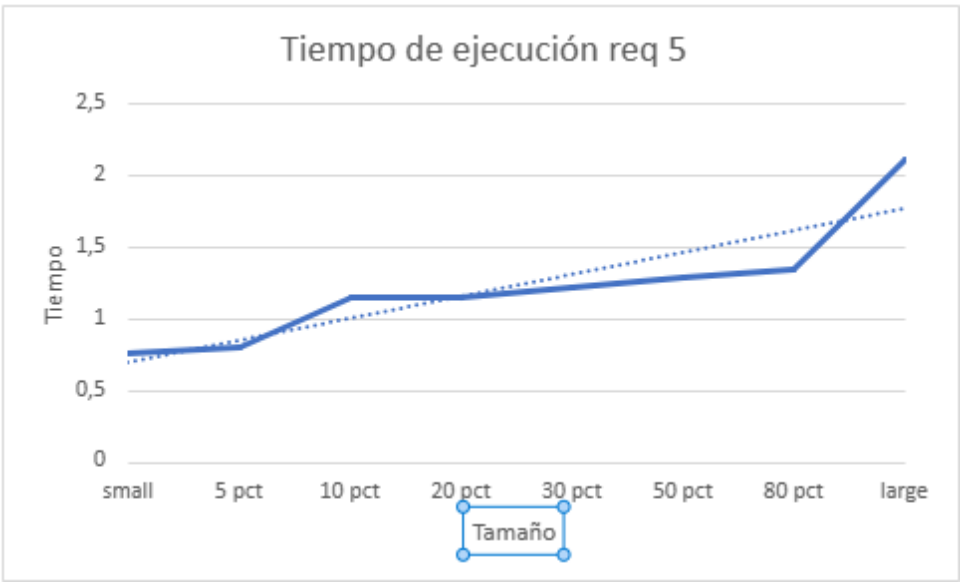
Pasos	**Complejidad**
Asignación variable “jugadores”	O (1)
Asignación variable “penalty”	O (1)
Asignación variable “auto”	O (1)
Creación nueva “ARRAY_LIST” = Variable “Torneos”	O (1)
Asignación variable “suma” equivalente al tamaño del mapa jugadores	O (1)
Retorno de la pareja llave – valor a partir del nombre del goleador	O(N)
Nueva Single_Linked list equivalente a la variable “lista”	O (1)
Asignación variable “partidos” a partir del valor proveniente de la llave (getValue)	O (1)
Recorrido por la lista “partidos”	O(M), siendo M el número de partidos pertenecientes a la variable “partidos”
TOTAL	***O(N)***

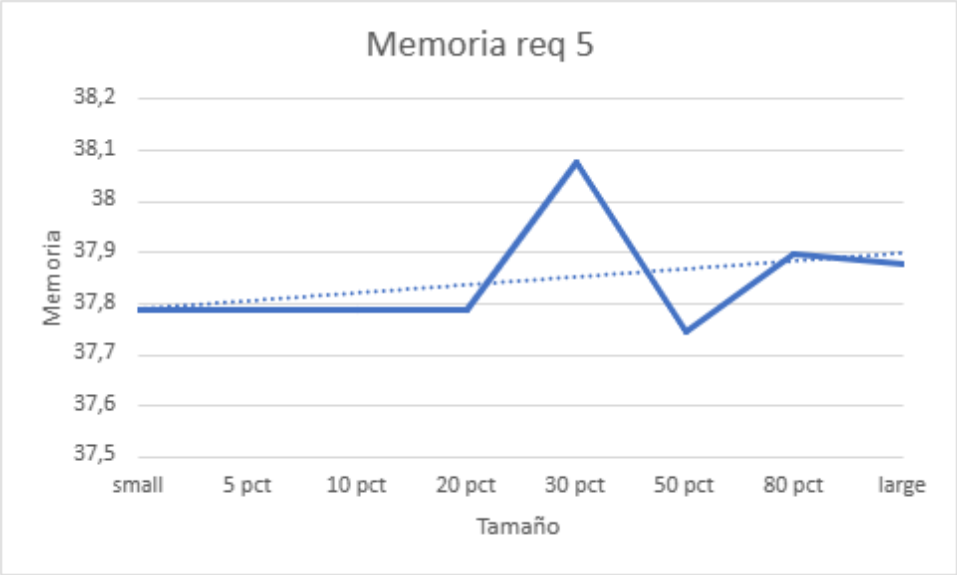
Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (ms)	Consumo de Datos [kB]
small	0.76	36,788
5pct	0.8	36,788
10pct	1,16	36,788
20pct	1,15	36,788
30pct	1,225	38,077
50pct	1,3	37,745
80pct	1,345	38,134
Large	2, 12	38,129

Graficas





Comparación Reto 1 y Reto 2

Tamaño	Tiempo en Reto 1	Tiempo en Reto 2
small	0,4	0.76
5 pct	2,09	0.8
10 pct	3,63	1,16
20 pct	9	1,15
30 pct	17,48	1,225
50 pct	33,24	1,3
80 pct	41,6	1,345
large	45,16	2, 12

Se puede evidenciar un cambio bastante significativo en el tiempo de ejecución, siendo el reto 2 mucho más eficaz que el reto 1.

Análisis

Se evidencia que este requerimiento tiene un comportamiento lineal lo cual se relaciona con la complejidad mencionada anteriormente O(N), entre mayor sea la cantidad de datos, mayor será el tiempo.

Por otro lado, respecto a la memoria, se evidencia que generalmente esta se mantiene constante. En este caso, los valores de la memoria se mantienen en un rango entre 36,000 y 39,000.

Requerimiento <<6>>

Descripción

```python
def req_6(catalog, n, torneo, year):
    """
    Función para obtener los N mejores equipos de una liga o torneo dentro de un periodo de
    tiempo. Esto puede entenderse como el TOP ranking de cierta cantidad de equipos en el torneo

    Args:
        catalog (dict): Catálogo con todas las estructuras que organizan la información de partidos
        n (int): El número de equipos a consultar
        torneo (str): El torneo de búsqueda
        year (int): El año de búsqueda

    Returns:
       equipo_list ('ARRAY_LIST): La lista ordenada de los N equipos con sus totales del torneo dentro del año
       totales (map): Un mapa con todos generales de los partidos
    """
    #Se va sacando totales generales según el tamaño del mapa
    total_years = mp.size(catalog['years'])
    entry = mp.get(catalog['years'], year)
    año = me.getValue(entry)
    total_matches= año['total_partidos']

    total_torneos = mp.size(año['torneos'])
    entry_torneo = mp.get(año['torneos'], torneo)
    torneos = me.getValue(entry_torneo)
    total_equipos = mp.size(torneos['equipos'])
    #Se obtiene todos los equipos que se encuentran con las especificaciones
    equipos = mp.keySet(torneos['equipos'])
    total_encuentros = 0
    total_paises = lt.newList('ARRAY_LIST')
    total_ciudades = {'total':0}
    equipo_list = lt.newList('ARRAY_LIST')

    for equipo in lt.iterator(equipos):
        #Se itera cada equipo para hallar los totales de este dentro del mapa
        entryeq = mp.get(torneos['equipos'], equipo)
        totales_equipo, total_paises, total_ciudades, total_encuentros= totales_por_equipos(equipo, entryeq, total_paises, total_ciudades, total_encuentros)
        lt.addLast(equipo_list, totales_equipo)
```

```python
    #Los partidos se ordenan segun puntos dentro de la lista
    merg.sort(equipo_list, compare_points)
    if n< lt.size(equipo_list):
        equipo_list = lt.subList(equipo_list, 1, n)


    total_country = lt.size(total_paises)
    total_cities = total_ciudades['total']
    del total_ciudades['total']
    mayor_ciudad = max(total_ciudades, key=total_ciudades.get)
    #Se completan los totales

    totales_generales =(total_years, total_torneos, total_equipos, total_matches, total_country, total_cities, mayor_ciudad)


    return equipo_list, totales_generales
```

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si. Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Asignación de variables	O(1)
Recorrido que itera por la lista de jugadores con **lt.iterator**	O(e) (cantidad de equipos que es menor a la cantidad de partidos)
Llamada a la función de **totales de equipos**	O(p) (lineal según la cantidad de partidos por equipos)
Ordenamiento por medio de **merg.sort()**	O(NlogN)
Creación de una sublista con **lt.sublist()**	O (1)
TOTAL	***O(NlogN)***

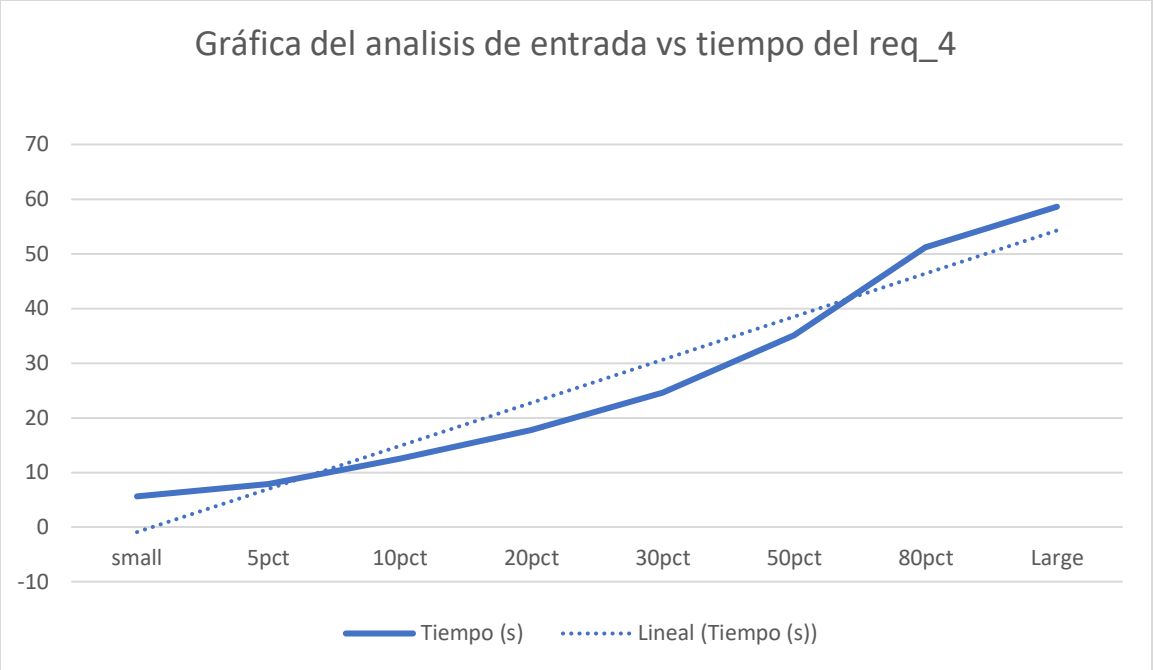
Pruebas Realizadas

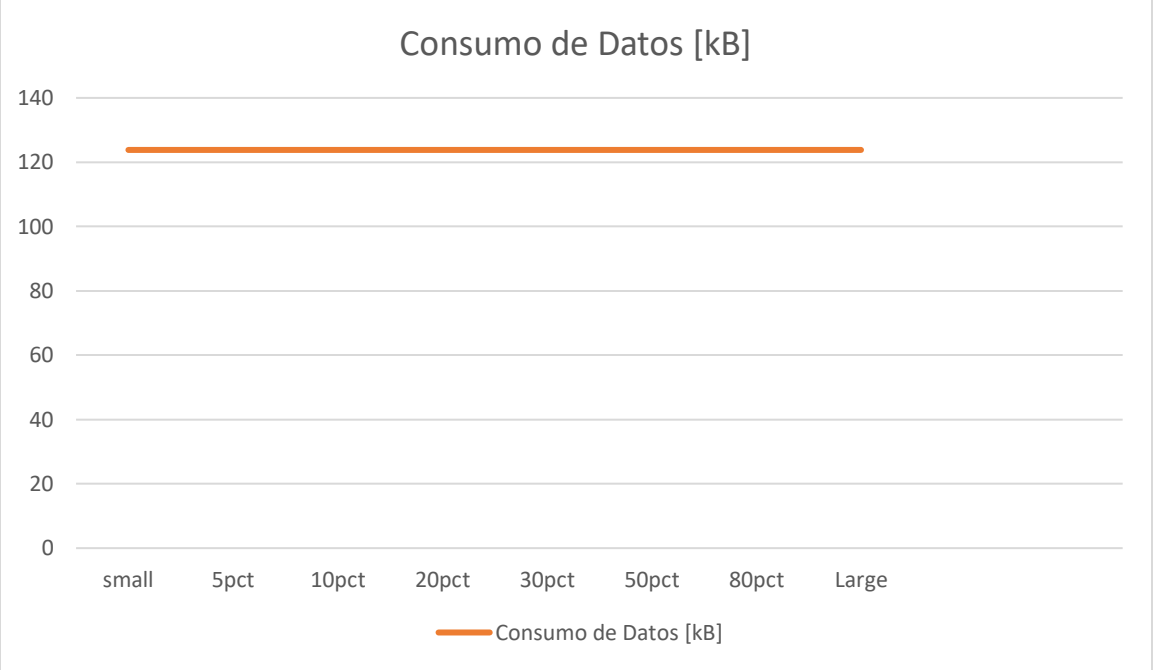
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (s)	Consumo de Datos [kB]
small	5.610	123.820
5pct	7.923	123.820
10pct	12.456	123.820
20pct	17.789	123.820
30pct	24.568	123.820
50pct	35.123	123.820
80pct	51.234	123.820
Large	58.611	123.820

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

Al analizar este requerimiento podemos apreciar un comportamiento similar a lo planteado en su análisis de complejidad. Si tomamos en cuenta factores como la utilización de la función de totales, podemos asumir y entender este tipo de comportamiento dentro de la gráfica temporal. Así mismo, gracias a las diferentes implementaciones, se asume constante la segunda gráfica gracias a diversas características.

Requerimiento <<7>>

Descripción

```
def req_7(catalog, torneo, n):
    """
    Función para obtener los jugadores de futbol con N puntos dentro de una competencia especifica.

    Args:
        catalog (dict): Catálogo con todas las estructuras que organizan la información de partidos
        torneo (srt): El torneo de búsqueda
        n (int): El puntaje que deben tener los goleadores

    Returns:
        lista_totales_jugadores (ARRAY_LIST): La lista con los totales de los jugadores con los n puntos del torneo
        totales_generales (tuple): La tupla contiene todos los totales generales de los partidos
    """
    #Se sacan totales generales según el tamaño del mapa
    total_torneos = mp.size(catalog['golea_torneos'])
    entry_torneo = mp.get(catalog['golea_torneos'], torneo)
    total_goleadores = mp.size(catalog['golea_torneos'])
    torneo_pair = me.getValue(entry_torneo)
    #Se toman los partidos con el puntaje seleccionado
    map_puntaje = torneo_pair['puntajes']
    puntaje = mp.get(map_puntaje, n)
    jugadores = me.getValue(puntaje)
    map_jugadores = jugadores['jugadores']
    #Se completan totales
    total_jugadores_torneo = mp.size(map_jugadores)
    lista_jugadores = mp.keySet(map_jugadores)
    lista_totales_jugadores = lt.newList('ARRAY_LIST')
    total_goles = 0
    total_penal = 0
    total_autogoles = 0
    for jugador in lt.iterator(lista_jugadores):
        #Por cada jugadorse hallan y almacenan los totales individuales
        totales_jug = me.getValue(mp.get(map_jugadores, jugador))
        lt.addLast(lista_totales_jugadores, totales_jug)
        #Se hallan más totales generales
        jug_total_goles = me.getValue(mp.get(totales_jug, 'total_goals'))
        jug_total_penal = me.getValue(mp.get(totales_jug, 'penalty_goals'))
        jug_total_autogoles = me.getValue(mp.get(totales_jug, 'own_goals'))
```

```
        total_goles += jug_total_goles
        total_penal += jug_total_penal
        total_autogoles += jug_total_autogoles
    lista_provicional = merg.sort(lista_totales_jugadores, compare_points_jug)
    totales_generales = (total_torneos, total_goleadores, total_jugadores_torneo,
                         total_goles, total_penal, total_autogoles)
    return lista_totales_jugadores, totales_generales
```

Breve descripción de como abordaron la implementación del requerimiento

Entrada	catalog: Catálogo con todas las estructuras que organizan la información de partidos torneo: El torneo de búsqueda n: El puntaje que deben tener los goleadores

Salidas	Lista_totales_jugadores: Lista que contiene todos los datos que se almacenarán dentro de la tabla, incluyendo total de puntos, total_de goles etc. Totales_generales : una tupla que contiene todos los totales que se le presentarán al usuario dentro del view, como preámbulo a los partidos
Implementado (Sí/No)	Sí.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación y acceso de variables	O (1)
Búsqueda de puntajes dentro del mapa con mp.get()	O (1)
Bucle con KeySet()	O(M) según la cantidad de jugadores con un puntaje especifico
Ordenamiento con merg.sort()	O(NlogN)
TOTAL	***O (Nlog N)***

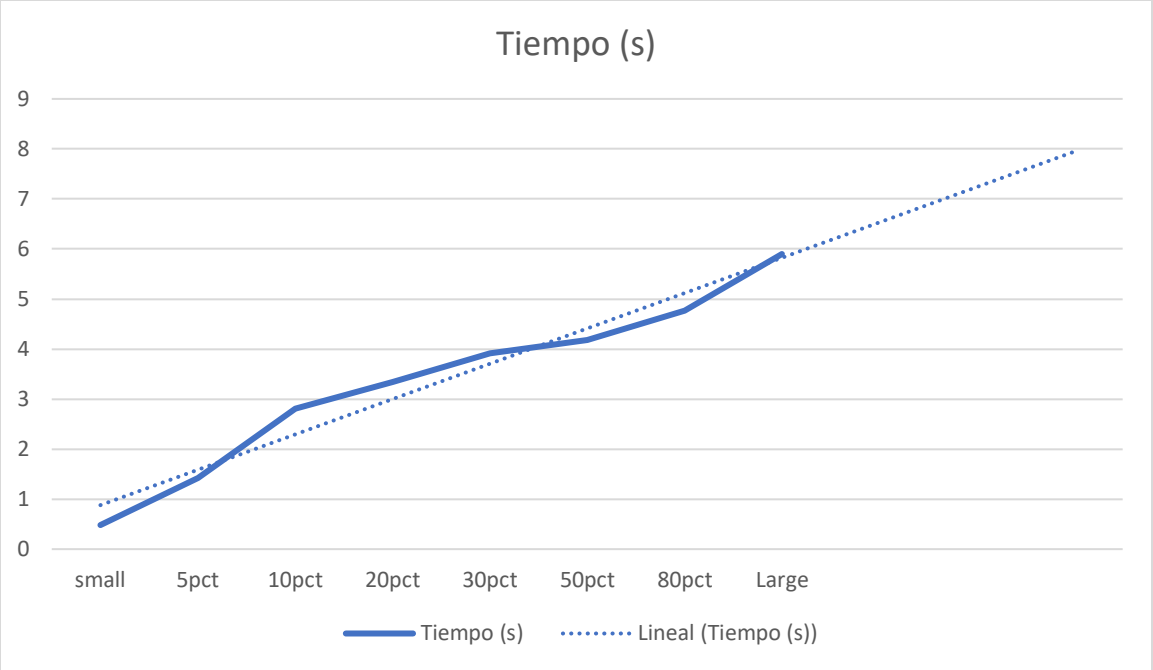
Pruebas Realizadas

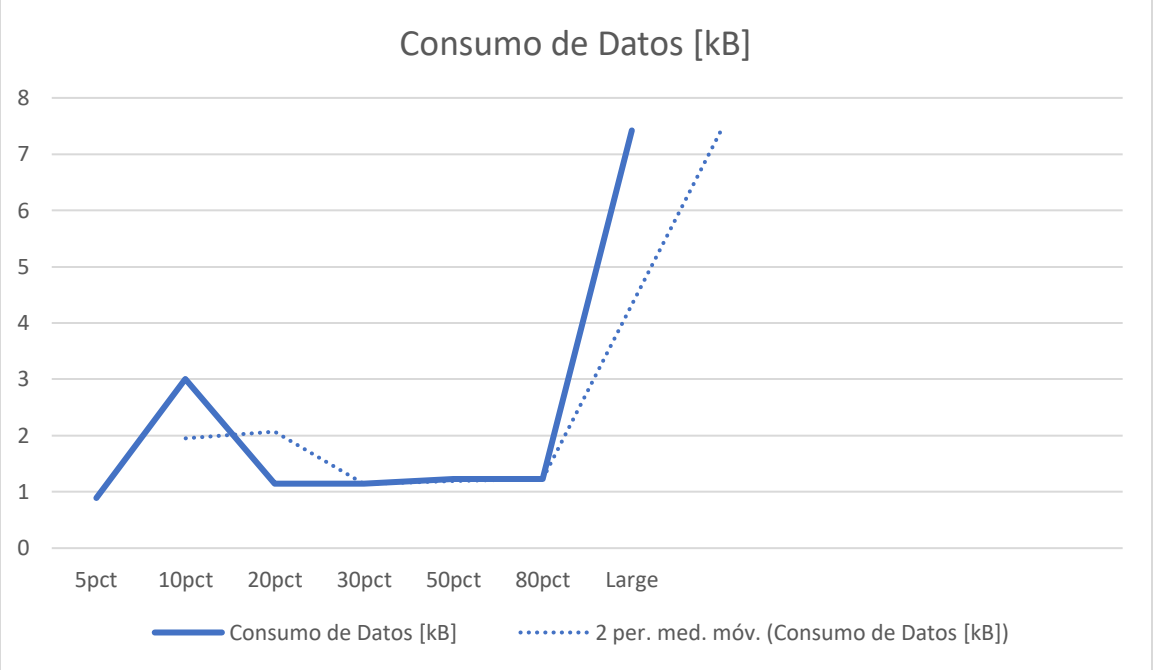
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**	**Consumo de Datos [kB]**
small	0.481	0.891
5pct	1.423	3.008
10pct	2.804	1.141
20pct	3.342	1.141
30pct	3.918	1.234
50pct	4.185	1.234
80pct	4.762	7.422
Large	5.899	1.328

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

Las gráficas demuestran de manera adecuada el análisis otorgado dentro del análisis de complejidad y reflejan la efectividad de usar las diferentes estructuras en comparación a otras. Gracias al uso transversal de mapas dentro de la creación de una estructura de datos, se puede inferir que la buena y clara complejidad se debe a este factor.