ANÁLISIS DEL RETO

1. *William Bayona Vergara, 202011494,* w.bayona@uniandes.edu.co

2. *Juan Esteban Africano, 202311163,* j.africano@uniandes.edu.co

3. *Daniel Sebastian Caro Ochoa, 202117080,* d.caro@uniandes.edu.co

Carga de Datos

Carga de Catálogo PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	132614.333	1121.295
0.5	102623.852	919.811
0.7	90910.458	1010.607
0.9	99273.005	1319.743

TABLA 1. COMPARACIÓN DE CONSUMO DE DATOS Y TIEMPO DE EJECUCIÓN PARA CARGA DE CATÁLOGO CON EL ÍNDICE POR CATEGORÍAS UTILIZANDO PROBING EN LA MAQUINA 1.

Carga de Catálogo CHAINING

Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	104449.414	1560.826
4.00	103079.914	1391.338
6.00	102625.258	1180.198
8.00	102398.445	915.723

TABLA 2. COMPARACIÓN DE CONSUMO DE DATOS Y TIEMPO DE EJECUCIÓN PARA CARGA DE CATÁLOGO CON EL ÍNDICE POR CATEGORÍAS UTILIZANDO CHAINING EN LA MAQUINA 1.

Requerimiento 1

Listar los últimos N partidos de un equipo según su condición

Descripción

```python
def req_1(teams,NPartidos,equipo,condicion):
    """
    Función que soluciona el requerimiento 1
    """
    team = me.getValue(mp.get(teams,equipo))
    merg.sort(team["results"],compare_date_home_away)
    totalequippos = mp.size(teams)
    totalpartidos = lt.size(team['results'])
    home_list = lt.newList('ARRAY_LIST')
    away_list = lt.newList('ARRAY_LIST')
    total_home = 0
    total_away = 0

    for partido in lt.iterator(team['results']):
        if partido['home_team'] == equipo:
            lt.addLast(home_list,partido)
            total_home += 1
        elif partido['away_team'] == equipo:
            lt.addLast(away_list,partido)
            total_away += 1

    if condicion == 'local':
        merg.sort(home_list,comparedate)
        return totalequippos, totalpartidos, total_home, home_list
    elif condicion == 'visitante':
        merg.sort(away_list,comparedate)
        return totalequippos, totalpartidos, total_away, away_list
    elif condicion == 'indiferente':
        merg.sort(team['results'],comparedate)
        return totalequippos, totalpartidos, totalpartidos, team['results']
```

Entrada	Estructura de datos de los Partidos (ARRAYLIST) Numero de partidos de consulta Nombre del equipo (selección nacional) Condición del equipo (Local, visitante, indiferente)
Salidas	Lista de los N últimos partidos de un equipo según la condición. Dentro de esta lista se menciona: Fecha del partido, equipo local, equipo visitante, puntaje equipo local, puntaje equipo visitante, torneo, ciudad y país. Tamaño de la lista
Implementado (Sí/No)	Si se implementó. Hecho por el grupo en trabajo colaborativo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Organizar la lista (Merge Sort)	O(NLogN)
Filtrar por Equipos (FiltroEquipos)	O(N)

Size (lt.size)	O(N)
Añadir elemento a la lista (addLast) [Dentro de un for]	O(#Partidos*1)
TOTAL	***O(NLogN)***

Pruebas Realizadas

Se realizaron las pruebas en el siguiente computador:

Procesadores	**AMD Ryzen 5 5600X 6-Core Processor**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Los parámetros de entrada fueron:

- Numero de partidos de consulta: 15
- Nombre del equipo (selección nacional): Spain
- Condición del equipo (Local, visitante, indiferente): Indiferente

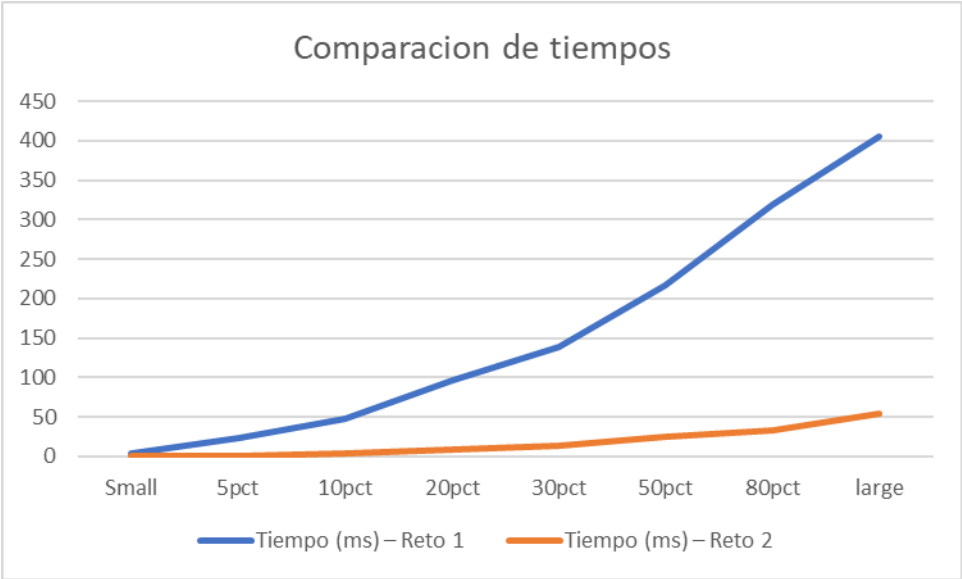
Entrada	**Tiempo (ms) – Reto 1**	**Tiempo (ms) – Reto 2**
small	3.888	0.509
5 pct	22.9799	1.4156
10 pct	47.5248	3.2772
20 pct	97.0256	8.4314
30 pct	138.6793	14.3166
50 pct	217.6976	25.1341
80 pct	319.1685	33.7441
large	406.1635	53.7134

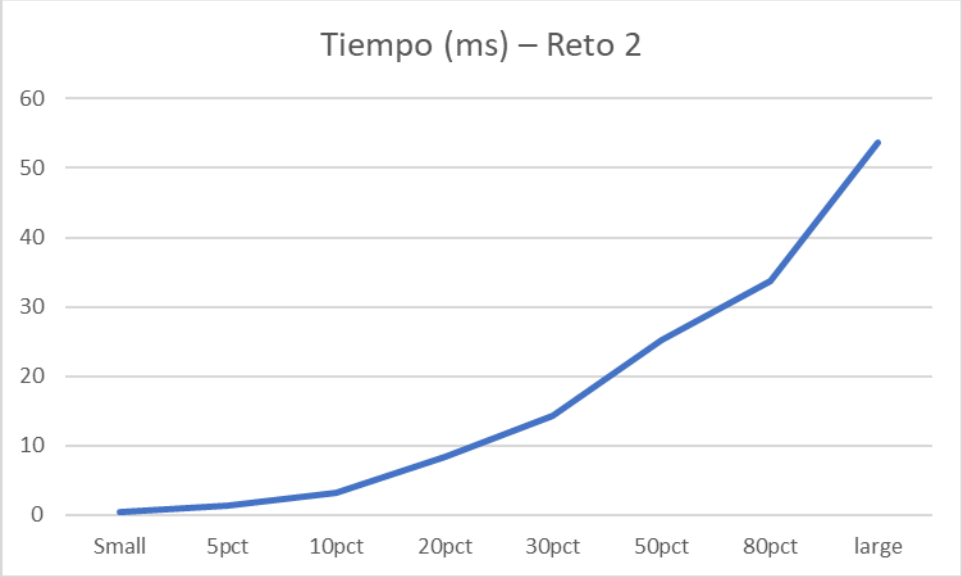
Tablas de datos

Muestra	**Salida**	**Tiempo (ms) -Reto 1**	**Tiempo (ms) – Reto 2**
small	Lista de partidos	3.888	0.509
5 pct	Lista de partidos	22.9799	1.4156
10 pct	Lista de partidos	47.5248	3.2772
20 pct	Lista de partidos	97.0256	8.4314
30 pct	Lista de partidos	138.6793	14.3166
50 pct	Lista de partidos	217.6976	25.1341
80 pct	Lista de partidos	319.1685	33.7441
large	Lista de partidos	406.1635	53.7134

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

Este requerimiento tiene una complejidad de NlogN, debido al mergeSort, pese a la implementación de mapas, la complejidad sigue definida por aquella función de ordenamiento. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.

.

Requerimiento 2:

Este requerimiento se encarga de listar los primeros N goles anotados por un jugador especifico.

```
def req_2(goleadores,Ngoles,jugador):

    Scorer=me.getValue(mp.get(goleadores,jugador))
    merg.sort(Scorer["results"],compare_date_minute)
    total_jogadores=mp.size(goleadores)
    total_anotaciones=lt.size(Scorer["results"])
    sub_penal=0

    for goleador in lt.iterator(Scorer["results"]):
        if goleador["penalty"]=="True":
            sub_penal+=1
    return total_jogadores,total_anotaciones,sub_penal,Scorer["results"]
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Goalscorers:Estructura de datos perteneciente a los goleadores (ARRAY LIST) Ngoles:es el número de goles a consultar Jugador: el jugador del que se quiere conocer la marca de goles
Salidas	Total anotaciones: Retorna el total de anotaciones obtenidas por el jugador Jugador_prime:una lista tipo ARRAY que contiene para cada anotacion la siguiente info: Fecha del partido. o Equipo local. o Equipo visitante. o Equipo del jugador. o Minuto en el que se marcó el gol. o Tipo de anotación, si fue por falta desde el penal. o Tipo de anotación, si fue autogol.
Implementado (Sí/No)	Si, en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Merge sort(organizar por fecha)	O(NLog(N)
Ciclo For	O(N)
Añadir elemento a la lista (addLast) [Dentro de un for]	O(1)
TOTAL	***O(NLogN)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

Procesadores	**AMD Ryzen 7 5700U with Radeon Graphics, 1801 Mhz, 16 procesadores lógicos**

Memoria RAM	16 GB
Sistema Operativo	Windows 11

Los parametros de entra asignados fueron los siguientes:

- Ngoles: los 7 primeros Goles
- Jugador:Lionel Messi

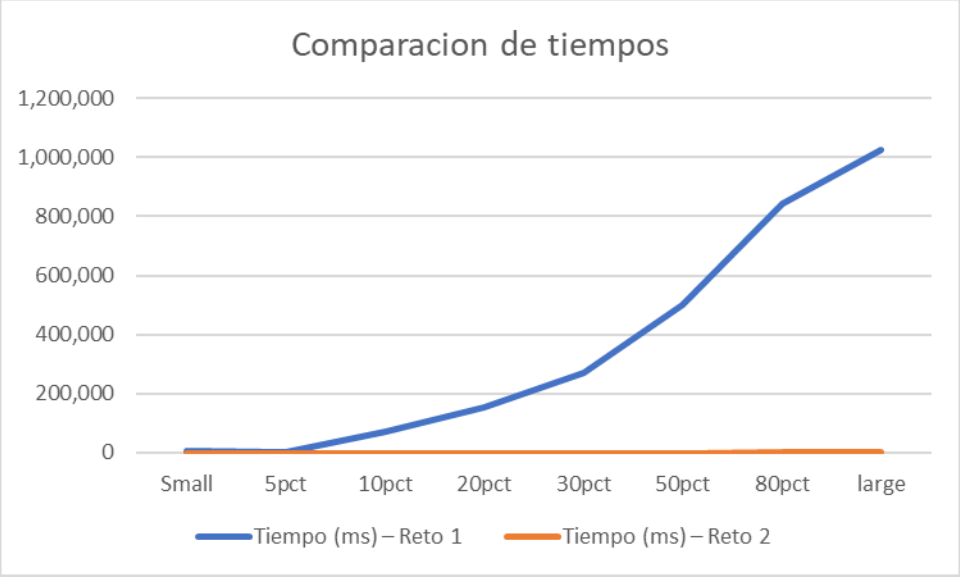
Entrada	**Tiempo (ms) – Reto 1**	**Tiempo (ms) – Reto 2**
small	4,555	0.0773
5 pct	36.760	0.0094
10 pct	71,271	0.1104
20 pct	155,138	0.1880
30 pct	269,998	0.2243
50 pct	498,810	0.2811
80 pct	844,865	0.4903
large	1026,46	1,0829

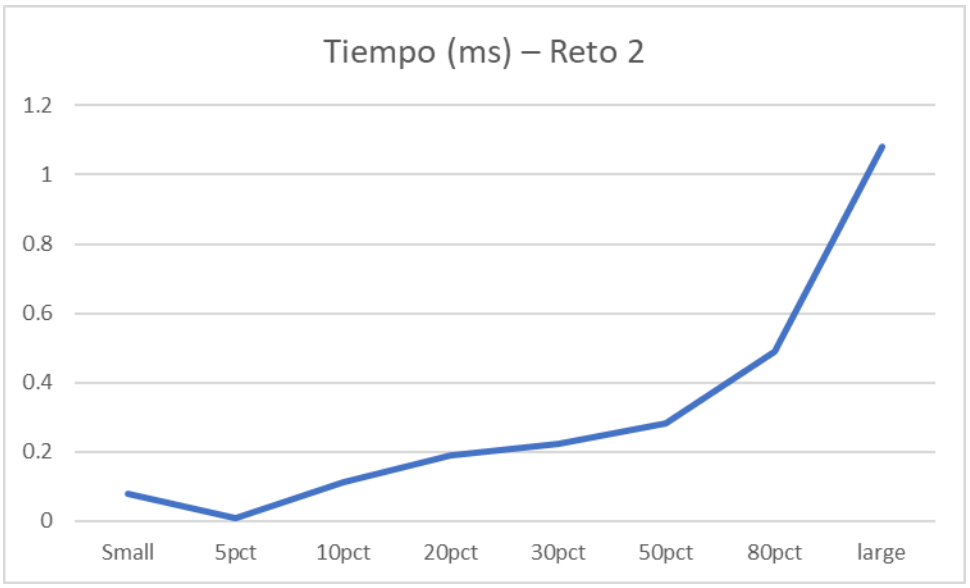
Tablas de datos

Muestra	**Salida**	**Tiempo (ms) -Reto 1**	**Tiempo (ms) – Reto 2**
small	Total de anotaciones, y Lista de los N goles seleccionado	4,555	0.0773
5 pct	Total de anotaciones, y Lista de los N goles seleccionado	36.760	0.0094
10 pct	Total de anotaciones, y Lista de los N goles seleccionado	71,271	0.1104
20 pct	Total de anotaciones, y Lista de los N goles seleccionado	155,138	0.1880

30 pct	Total de anotaciones, y Lista de los N goles seleccionado	269,998	0.2243
50 pct	Total de anotaciones, y Lista de los N goles seleccionado	498,810	0.2811
80 pct	Total de anotaciones, y Lista de los N goles seleccionado	844,865	0.4903
large	Total de anotaciones, y Lista de los N goles seleccionado	1026,46	1,0829

Graficas





Análisis

Este requerimiento tiene una complejidad de NlogN, debido al mergeSort, pese a la implementación de mapas, la complejidad sigue definida por aquella función de ordenamiento. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.

Requerimiento 3:

Este requerimiento se encarga de consultar los partidos que disputó un equipo durante un periodo especifico.

Descripción

```
def req_3(teams,goleadores_team,equipo,inicio,final):
    """
    Función que soluciona el requerimiento 3
    """
    totalequippos = mp.size(teams)
    team = me.getValue(mp.get(teams,equipo))
    merg.sort(team["results"],compare_date_home_away)
    partidos = FiltroFechas(team["results"],inicio,final)

    goleadores = me.getValue(mp.get(goleadores_team,equipo))
    merg.sort(goleadores["results"],compare_date_minute)
    goleadores = FiltroFechas(goleadores["results"],inicio,final)

    for partido in lt.iterator(partidos):
        penalty = "Unknown"
        owngoal = "Unknown"
        for goleador in lt.iterator(goleadores):
            if goleador["date"]== partido["date"] and goleador["team"]==equipo:
                penalty = goleador["penalty"]
                owngoal = goleador["own_goal"]
        partido["penalty"]=penalty
        partido["own_goal"]=owngoal

    partidos = merg.sort(partidos, comparedate2)
    total =lt.size(partidos)
    totalhome = 0
    totalaway = 0

    for partido in lt.iterator(partidos):
        if partido["home_team"] == equipo:
            totalhome+=1
        if partido["away_team"] == equipo:
            totalaway+=1

    return partidos,total,totalhome,totalaway, totalequippos
```

Entrada	Estructura de datos de los Partidos (ARRAYLIST) Estructura de datos de los goleadores (ARRAYLIST) Nombre del equipo a consultar Fecha de inicio de consulta Fecha final de consulta
Salidas	Lista de los partidos del equipo consultado, en el periodo especifico. Incluyendo los datos: Fecha del partido, marcador local, marcador visitante, equipo local, equipo visitante, país del partido, ciudad del partido, nombre del torneo asociado, anotación si el partido tuvo goles desde punto penal, anotación si el partido presento autogoles. Número total de partidos Numero total de partidos disputados como local

	Número total de partidos disputados como visitante
Implementado (Sí/No)	Si se implementó. Hecho por William Bayona

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Organizar las listas (Merge Sort)	O(NLogN)
Filtrar por Equipos (FiltroEquipos)	O(N)
Filtrar por Fechas (FiltroEquipos)	O(N)
Comparar datos (Dentro de dos for, uno por goleadores y otro por partidos)	O(N*M) Donde N son los datos de partidos y M son los datos de goleadores
Tamaño de la lista lt.size()	O(N)
Evaluar si elemento está en lista (Dentro de un for que itera por partidos)	O(N)
TOTAL	***O(NLogN)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

Procesadores	**AMD Ryzen 5 4500U (with Radeon Graphics) CPU @ 2.38 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Los parámetros de entrada fueron:

- Nombre del equipo: Alemania (“Germany”)
- Fecha de inicio de consulta: 1939-01-01
- Fecha final de consulta: 1980-12-31

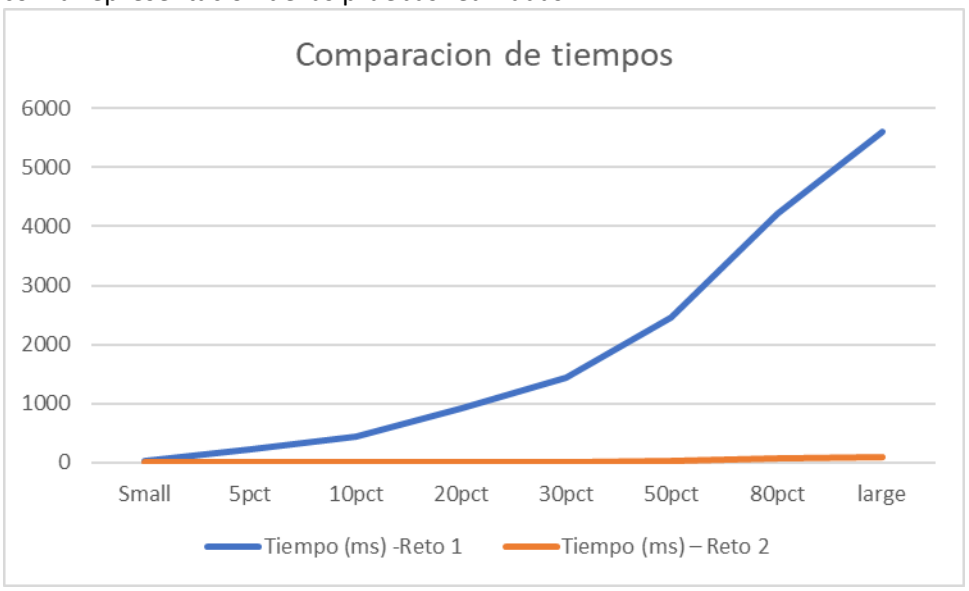
Entrada	**Tiempo (ms) – Reto 1**	**Tiempo (ms) – Reto 2**
small	31.92	0.8461
5 pct	224.97	2.4521
10 pct	440.19	4.6051
20 pct	926.90	8.5332
30 pct	1435.56	14.5639
50 pct	2459.69	26.0567
80 pct	4205.39	75.8497
large	5592.90	92.0525

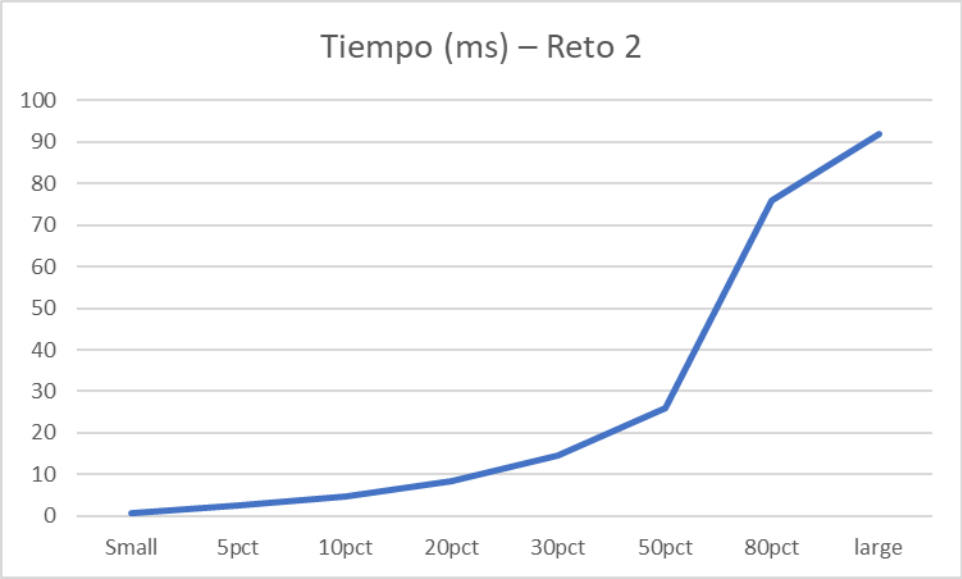
Tablas de datos

Muestra	Salida	Tiempo (ms) -Reto 1	Tiempo (ms) – Reto 2
small	Lista con partidos filtrados	31.92	0.8461
5 pct	Lista con partidos filtrados	224.97	2.4521
10 pct	Lista con partidos filtrados	440.19	4.6051
20 pct	Lista con partidos filtrados	926.90	8.5332
30 pct	Lista con partidos filtrados	1435.56	14.5639
50 pct	Lista con partidos filtrados	2459.69	26.0567
80 pct	Lista con partidos filtrados	4205.39	75.8497
large	Lista con partidos filtrados	5592.90	92.0525

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

Este requerimiento tiene una complejidad de NlogN, debido al mergeSort, pese a la implementación de mapas, la complejidad sigue definida por aquella función de ordenamiento. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.

Requerimiento 4:

Consultar los partidos relacionados con un torneo durante un periodo especifico

Descripción

```
def req_4(torneos,penales_team,torneo,inicio,final):
    """
    Función que soluciona el requerimiento 4
    """
    total_torneos = mp.size(torneos)
    partidos = me.getValue(mp.get(torneos,torneo))
    req4 = FiltroFechas(partidos['results'], inicio, final )
    merg.sort(req4,compare_date_home_away)
    total_partidos=lt.size(req4)
    totalshoot = 0
    totalcountry = 0
    countries = []
    totalcity = 0
    cities = []

    for partido in lt.iterator(req4):

        if partido["country"] not in countries:
            countries.append(partido["country"])
            totalcountry+=1

        if partido["city"] not in  cities:
            cities.append(partido["city"])
            totalcity+=1

        partido["winner"] = "Unknown"
        equipo = partido["home_team"]
        existpenales = mp.contains(penales_team,equipo)
        if existpenales == True:
            penales_posibles  = me.getValue(mp.get(penales_team,equipo))
            penales_posibles = FiltroFechas(penales_posibles["results"],inicio,final)

            for penal in lt.iterator(penales_posibles):
                if penal["date"] == partido["date"] and partido["home_team"] == penal["home_team"]:
                    partido["winner"]=penal["winner"]
                    totalshoot+=1

    merg.sort(req4,comparedate)
    return total_torneos,total_partidos, totalcountry,totalcity,totalshoot,req4
```

Entrada	Estructura de datos de los Partidos (ARRAYLIST) Estructura de datos de los Partidos definidos por Penales (ARRAYLIST) Nombre del torneo a consultar Fecha de inicio de consulta Fecha final de consulta
Salidas	Lista de los partidos del torneo, en el periodo especifico. En dicha lista se menciona la fecha, equipo local, equipo visitante, puntuación equipo local, puntuación equipo visitante, torneo, ciudad, país y ganador (en caso de que el partido se haya definido por penales) Total de partidos del torneo, total de países involucrados en el torneo, total de ciudades donde se jugaron los partidos del torneo, total de partidos definidos por penales.

Implementado (Sí/No)	Si se implementó. Hecho por William Bayona

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Organizar las listas (Merge Sort)	O(NLogN)
Filtrar por Equipos (FiltroEquipos)	O(N)
Filtrar por Fechas (FiltroEquipos)	O(N)
Comparar datos (Dentro de dos for, uno por penales y otro por partidos)	O(N*M) Donde N son los datos de partidos y M son los datos de penales
Tamaño de la lista lt.size()	O(N)
Evaluar si elemento esta en lista (Dentro de un for que itera por partidos)	O(N)
TOTAL	***O(NLogN)***

Pruebas Realizadas

Se realizaron las pruebas en el siguiente computador:

Procesadores	**AMD Ryzen 5 5600X 6-Core Processor**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Los parámetros de entrada fueron:

- Nombre del torneo: Copa América
- Fecha de inicio de consulta: 1999-01-01
- Fecha final de consulta:2020-10-10

Entrada	**Tiempo (ms) – Reto 1**	**Tiempo (ms) – Reto 2**
small	7.6875	0.7030
5 pct	22.4796	1.0653
10 pct	46.3555	2.0955
20 pct	92.6187	3.6054
30 pct	143.5854	7.7720
50 pct	212.0635	8.4131
80 pct	318.9102	10.0895
large	388.3736	23.7606

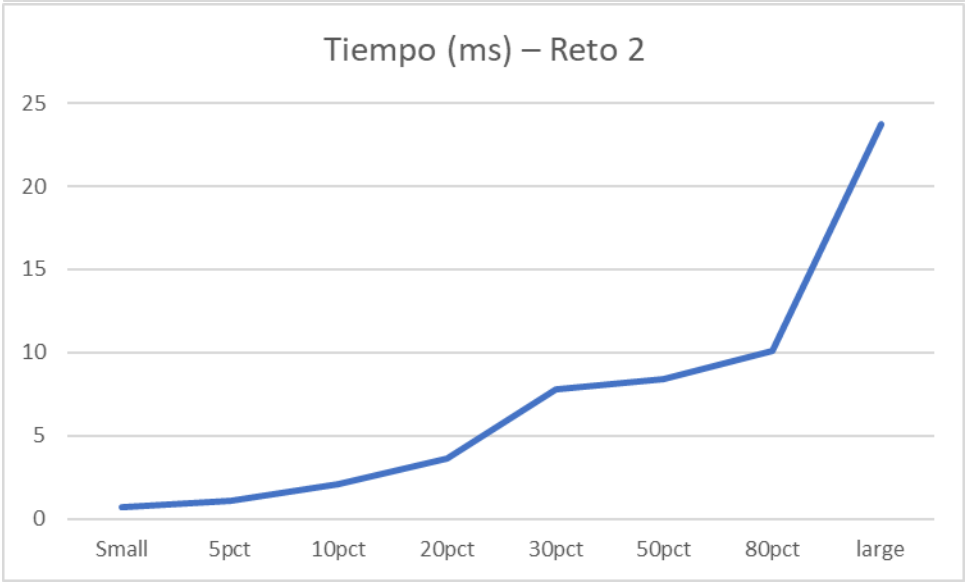
Tablas de datos

Muestra	**Salida**	**Tiempo (ms)- Reto 1**	**Tiempo (ms) – Reto 2**

small	Lista con partidos filtrados	7.6875	0.7030
5 pct	Lista con partidos filtrados	22.4796	1.0653
10 pct	Lista con partidos filtrados	46.3555	2.0955
20 pct	Lista con partidos filtrados	92.6187	3.6054
30 pct	Lista con partidos filtrados	143.5854	7.7720
50 pct	Lista con partidos filtrados	212.0635	8.4131
80 pct	Lista con partidos filtrados	318.9102	10.0895
large	Lista con partidos filtrados	388.3736	23.7606

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento tiene una complejidad de NlogN, debido al mergeSort, pese a la implementación de mapas, la complejidad sigue definida por aquella función de ordenamiento. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.

Requerimiento 5:

Consultar las anotaciones de un jugador durante un periodo especifico

```
def req_5(Top,jugador,fecha_ini,fecha_fin):
    """
    Función que soluciona el requerimiento 5
    """
    Scorer=me.getValue(mp.get(Top,jugador))
    merg.sort(Scorer["results"],compare_date_minute)
    total_anotaciones=lt.size(Scorer["results"])
    sub_penal=0
    sub_torneos=0
    sub_autogol=0
    torneos=[]
    date_list=lt.newList('ARRAY_LIST')
    for goleador in lt.iterator(Scorer["results"]):
        if dt.strptime(fecha_ini , "%Y-%m-%d").date() <= goleador["date"] <=  dt.strptime(fecha_fin , "%Y-%m-%d").date():
            if goleador["penalty"]=="True":
                sub_penal+=1
            if goleador["own_goal"]=="True":
                sub_autogol+=1
            if goleador["tournament"] not in torneos:
                torneos.append(goleador["tournament"])
            sub_torneos += 1

            lt.addLast(date_list,goleador)
    merg.sort(date_list, comparedate)
    return total_anotaciones,sub_torneos,sub_penal,sub_autogol,date_list
```

Descripción

Entrada	Estructura de datos de los Partidos (ARRAYLIST) Estructura de datos de los Goleadores (ARRAYLIST) Nombre del Jugador para consultar Fecha de inicio de consulta Fecha final de consulta
Salidas	Cantidad de goles anotados. (En el periodo de tiempo) Cantidad de torneos jugados. (Sin repetir) Cantidad de penales anotados. Cantidad de autogoles. El listado de las anotaciones del jugador ordenadas cronológicamente por fecha y minuto en que se marcó el gol en el encuentro. Donde cada uno de los elementos resultantes

	contendrá la siguiente información: Fecha del partido, minuto en el que se marcó el gol, equipo local, equipo visitante.,Equipo del jugador, Marcador del equipo local (goles del local), Marcador del equipo visitante (goles del visitante), Nombre del torneo donde se marcó el gol, Tipo de anotación, si fue por falta desde el penal, Tipo de anotación, si fue autogol.
Implementado (Sí/No)	Si se implementó. Hecho por Juan Esteban Africano

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Organizar las listas (Merge Sort)	O(NLogN)
Filtrar la estructura de datos (resultados) por límite de fecha (FiltroEquipos)	O(M) #de resultados
Ciclo For que recorra la estructura de datos(Goalscorers) filtrando por fecha y anexe un diccionario un diccionario para cada anotación del jugador	O(N)
Filtrar por Fechas (FiltroEquipos)	O(N)
Comparar datos para añadir los valores restantes los diccionarios de la lista (un for dentro de otro for , el primero son las anotaciones metidas por el jugador y el segundo se encarga de iterar los resultados para anexar los valores restantes)	O(N*M) Donde N son los datos de cada anotación del jugador s y M son los datos de resultados
Evaluar si elemento esta en lista (Dentro de un For que itera por partidos)	O(N)
TOTAL	***O(NLogN)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

Procesadores	**AMD Ryzen 7 5700U with Radeon Graphics, 1801 Mhz, 16 procesadores lógicos**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

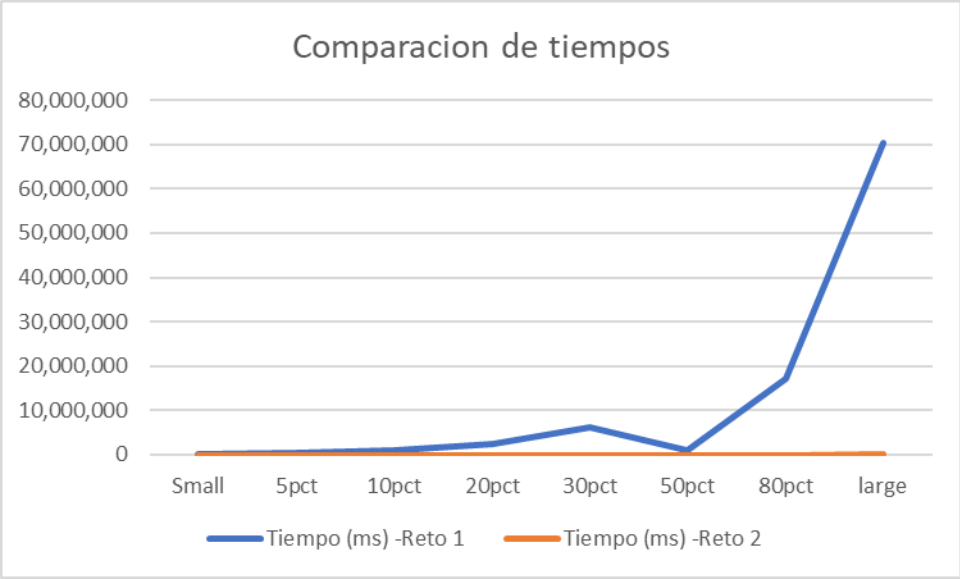
Parámetros de entrada definidos

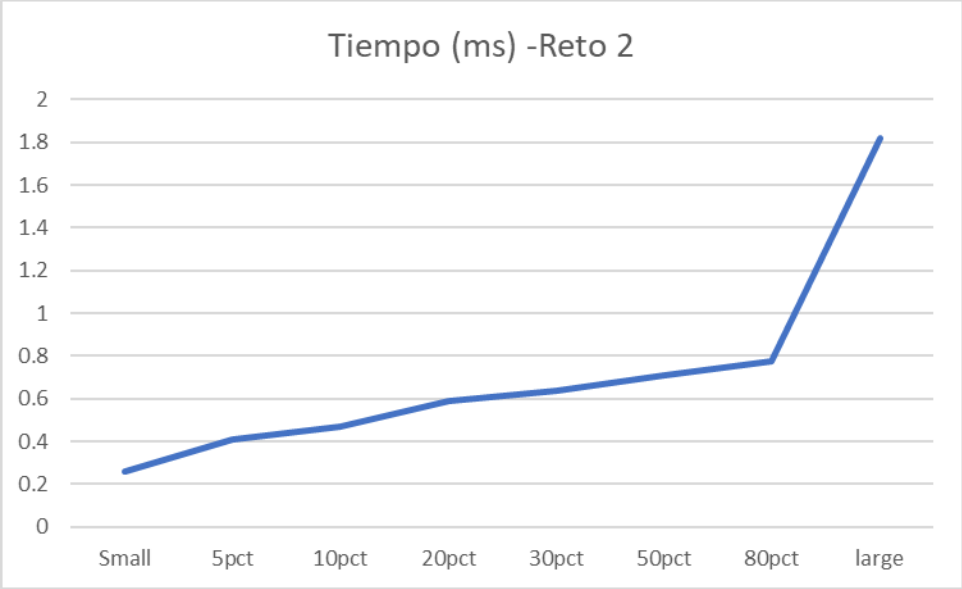
- Nombre del torneo: Lionel Messi
- Fecha de inicio de consulta: 2004-04-04
- Fecha final de consulta:2016-10-10

Entrada	Tiempo (ms) -Reto 1	Tiempo (ms) -Reto 2
small	9,2887	0.2599
5 pct	35,7654	0.4075
10 pct	90,8765	0.4671
20 pct	241,7734	0.5874
30 pct	626,7542	0.6403
50 pct	977,975	0.7098
80 pct	1710,8652	0.7748
large	7042,8753	1,0820

Graficas

Las gráficas con la representación de las pruebas realizadas.





Análisis

Este requerimiento tiene una complejidad de NlogN, debido al mergeSort, pese a la implementación de mapas, la complejidad sigue definida por aquella función de ordenamiento. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.

Requerimiento 6:

Clasificar los N mejores equipos de un torneo en un periodo especifico

Descripción

```python
def req_6 (torneos,years,goleadores_team,Nequipos,torneo,year):
    total_years = mp.size(years)
    partidos_year = me.getValue(mp.get(years,year))
    partidos_year = partidos_year["results"]

    partidos_torneo = me.getValue(mp.get(torneos,torneo))
    partidos_torneo = partidos_torneo['results']
    inicio = year + "-01-01"
    final = year + "12-31"
    partidos_torneo = FiltroFechas(partidos_torneo,inicio,final)
    total_partidos = lt.size(partidos_torneo)

    equipos = mp.newMap(800,
                        maptype='PROBING',
                        loadfactor=0.5,
                        )

    totalcountry = 0
    countries = []
    totalcity = 0
    cities = []

    for partido in lt.iterator(partidos_torneo):
        addbyteam(equipos,partido)

        if partido["country"] not in countries:
            countries.append(partido["country"])
            totalcountry+=1
        if partido["city"] not in  cities:
            cities.append(partido["city"])
            totalcity+=1
```

```
total_equipos = mp.size(equipos)
equipos_list = mp.valueSet(equipos)
req6 = lt.newList('ARRAY_LIST')

for equipo in lt.iterator(equipos_list):
    equipo = equipo['Equipo']
    partidos_equipo = me.getValue(mp.get(equipos,equipo))
    partidos_equipo = partidos_equipo['results']
    goleadores_equipo = me.getValue(mp.get(goleadores_team,equipo))
    goleadores_equipo = goleadores_equipo['results']

    equipo_dict = {
        "team": equipo,
        "total_points":0,
        "goal_difference":0,
        "penalty_points":0,
        "matches":0,
        "own_goal_points":0,
        "wins":0,
        "draws":0,
        "losses":0,
        "goals_for":0,
        "goals_againist":0,
        "top_scorer":{}
        }
```

```
    equipo_dict["matches"] = lt.size(partidos_equipo)
    wins = 0
    draws = 0
    losses = 0

    for partido in lt.iterator(partidos_equipo):

        if equipo == partido["home_team"]:
            goals_team_scorer = int(partido["home_score"])
            goals_team_other = int(partido["away_score"])
        elif equipo == partido["away_team"]:
            goals_team_scorer = int(partido["away_score"])
            goals_team_other = int(partido["home_score"])

        equipo_dict["goal_difference"] = goals_team_scorer - goals_team_other

        if goals_team_scorer > goals_team_other:
            wins += 1
        elif goals_team_scorer == goals_team_other:
            draws += 1
        elif goals_team_scorer < goals_team_other:
            losses += 1
```

```
    equipo_dict["wins"] = wins
    equipo_dict["draws"] = draws
    equipo_dict["losses"] = losses
    equipo_dict["total_points"] = wins*3 + draws

    top_scorer = lt.subList(goleadores_equipo,0,1)
    top_scorer = tabulate(lt.iterator(top_scorer),headers="keys",tablefmt="grid")
    equipo_dict["top_scorer"] = top_scorer

    lt.addLast(req6,equipo_dict)

#merg.sort(req6,comparedate)

return total_years, total_equipos, total_partidos, totalcountry, totalcity, req6
```

Entrada	Estructura de datos de los Partidos (ARRAYLIST) Estructura de datos de los Partidos definidos por Penales (ARRAYLIST) Estructura de datos de los Goleadores (ARRAYLIST) Numero de equipos para consultar Nombre del torneo a consultar Año del torneo a consultar
Salidas	Total de equipos involucrados en el torneo Total de partidos en el periodo de tiempo Total de países en el torneo Total de ciudades en el torneo Nombre de la ciudad donde mas partidos se han hecho Lista de equipos que conforman el torneo. Donde se resaltan los siguientes datos: Nombre del equipo Puntos obtenidos Diferencia de goles Total de partidos Total de puntos desde línea penal Total de puntos por autogol Total victorias Total Empates Total derrotas Total goles obtenidos por jugadores Total goles recibidos Jugador con mas anotaciones, con los siguientes datos: Nombre, goles anotados, partidos jugados donde marco un gol, promedio de tiempo para anotar un gol
Implementado (Sí/No)	Si, se implemento en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Evaluar si elemento está en lista (Dentro de un for que itera por partidos)	O(N)
Comparar elementos (Dentro de dos for, uno que itera por partidos y otro que itera por penales)	O(M*N) Donde M es el numero de datos de los penales y N es el numero de datos de los partidos
Declaración de variables (Dentro de un for que itera sobre los equipos)	O(T) Donde T es el numero de datos que respecta a los equipos (Numero de equipos)
Comparar elementos (Dentro de tres for, uno que itera por partidos, otro que itera por penales y otro que itera por equipos)	O(M*N*T) Donde M es el numero de datos de los penales, N es el numero de datos de los partidos y T es el numero de equipos
Filtrar por Torneo (FiltroEquipos)	O(N)
Filtrar por Fechas (FiltroEquipos)	O(N)
TOTAL	***O(N)***

Pruebas Realizadas

Se realizaron las pruebas en el siguiente computador:

Procesadores	**AMD Ryzen 5 5600X 6-Core Processor**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Los parámetros de entrada fueron:

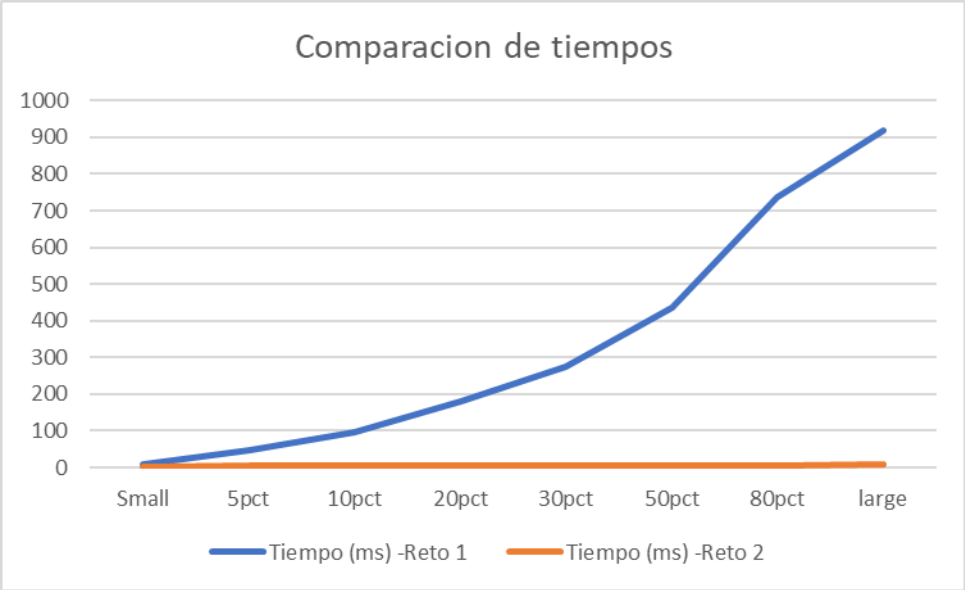
- Nombre del torneo: Copa América
- Numero de Equipos: 11
- Año:2021

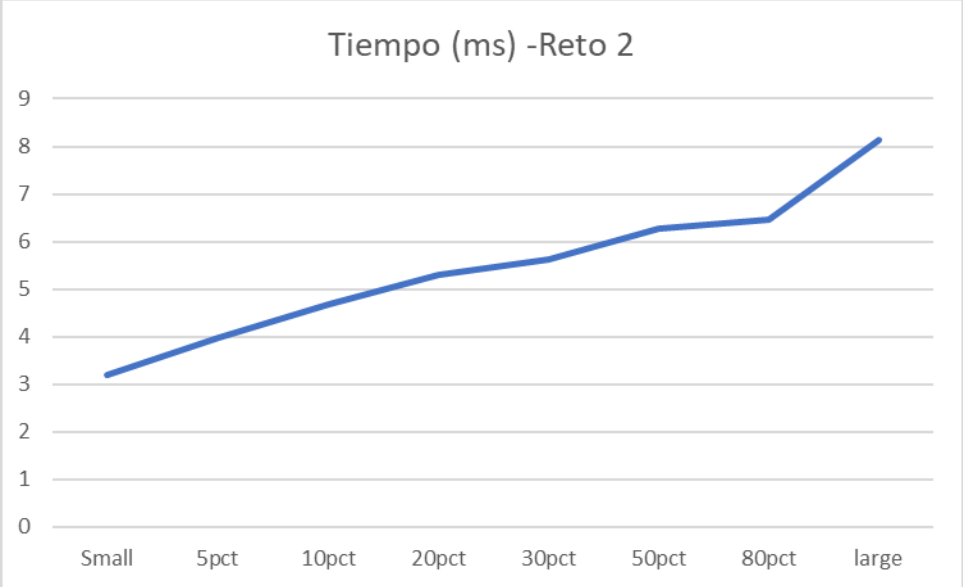
Entrada	**Tiempo (ms) -Reto 1**	**Tiempo (ms) -Reto 2**
small	10.4067	3.2023
5 pct	46.8925	3.9641
10 pct	95.0243	4.6767
20 pct	180.2674	5.2893
30 pct	274.0722	5.6261
50 pct	435.7599	6.2686
80 pct	737.0959	6.4551
large	919.3295	8.1448

Tablas de datos

Muestra	Salida	Tiempo (ms) -Reto 1	Tiempo (ms) -Reto 2
small	Listado de anotadores	10.4067	3.2023
5 pct	Listado de anotadores	46.8925	3.9641
10 pct	Listado de anotadores	95.0243	4.6767
20 pct	Listado de anotadores	180.2674	5.2893
30 pct	Listado de anotadores	274.0722	5.6261
50 pct	Listado de anotadores	435.7599	6.2686
80 pct	Listado de anotadores	737.0959	6.4551
large	Listado de anotadores	919.3295	8.1448

Graficas





Análisis

Este requerimiento tiene una complejidad de N, debido a un iterador sobre los partidos. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.

Requerimiento 7:

Este requerimiento se encarga de clasificar los N mejores anotadores en partidos oficiales dentro de un periodo especifico.

Descripción

```
def req_7(torneos,goleadores_team,torneo, NPuntos):
    """
    Función que soluciona el requerimiento 7
    """
    total_torneos = mp.size(torneos)
    partidos = me.getValue(mp.get(torneos,torneo))
    partidos = partidos["results"]
    merg.sort(partidos,compare_date_home_away)
    total_partidos = lt.size(partidos)
    total_goles = 0
    total_penales = 0
    total_owngoales = 0
    anotadores = mp.newMap(800,
                           maptype='PROBING',
                           loadfactor=0.5,
                           )

    for partido in lt.iterator(partidos):
        total_goles += int(partido["home_score"]) + int(partido["away_score"])
        equipo = partido['home_team']
        scorer_team = me.getValue(mp.get(goleadores_team,equipo))
        scorer_team = scorer_team['results']

        for goleador in lt.iterator(scorer_team):
            if goleador["date"] == partido["date"] and goleador["home_team"] == partido["home_team"]:
                goleador['home_score'] = partido['home_score']
                goleador['away_score'] = partido['away_score']
                addbyscorer(anotadores,goleador)

    anotadores_list = mp.valueSet(anotadores)
    req7 = lt.newList('ARRAY_LIST')

    for scorer in lt.iterator(anotadores_list):
        scorer = scorer['scorer']
        anotaciones = me.getValue(mp.get(anotadores,scorer))
        anotaciones = anotaciones['results']
```

```
    scorer_torneo = {
        "scorer": scorer,
        "total_points":0,
        "total_goals":0,
        "penalty_goals":0,
        "own_goals":0,
        "avg_time (min)":0,
        "scored_in_wins":0,
        "scored_in_losses":0,
        "scored_in_draws":0,
        "last_goal":{}
    }
    sum_prom = 0
    n_prom = 0
    wins = 0
    draws = 0

    for anotacion in lt.iterator(anotaciones):
        scorer_torneo["total_goals"] += 1
        sum_prom += float(anotacion["minute"])
        n_prom += 1

        if anotacion["penalty"]=="True":
            scorer_torneo["penalty_goals"] += 1
            total_penales += 1
        if anotacion["own_goal"]=="True":
            scorer_torneo["own_goals"] += 1
            total_owngoales += 1
```

```
        if anotacion["team"] == anotacion["home_team"]:
            goals_team_scorer = int(anotacion["home_score"])
            goals_team_other = int(anotacion["away_score"])
        elif anotacion["team"] == anotacion["away_team"]:
            goals_team_scorer = int(anotacion["away_score"])
            goals_team_other = int(anotacion["home_score"])

        if goals_team_scorer > goals_team_other:
            scorer_torneo["scored_in_wins"] += 1
            wins += 1
        elif goals_team_scorer == goals_team_other:
            scorer_torneo["scored_in_draws"] += 1
            draws += 1
        elif goals_team_scorer < goals_team_other:
            scorer_torneo["scored_in_losses"] += 1

    scorer_torneo["total_points"] = wins*3 + draws
    scorer_torneo["avg_time (min)"] = sum_prom/n_prom
    last_goal = lt.subList(anotaciones,0,1)
    last_goal = tabulate(lt.iterator(last_goal),headers="keys",tablefmt="grid")
    scorer_torneo["last_goal"] = last_goal


    if scorer_torneo["total_points"] == int(NPuntos):
        lt.addLast(req7,scorer_torneo)

merg.sort(req7,compare_avgminute)
total_scorers = mp.size(anotadores)

return total_torneos,total_scorers,total_partidos,total_goles, total_penales, total_owngoales, req7
```

Breve descripción de como abordaron la implementación del requerimiento

Entrada	NJugadores: Número de jugadores que desean consultar Inicio: La fecha de inicio del periodo a consultar (con formato "%Y-%m-%d") Fin: La fecha final del periodo a consultar (con formato "%Y-%m-%d")
Salidas	total_torneos: Número de torneos donde participaron los partidos en ese periodo. Total_scorers: Número de anotadores encontrados en la consulta. Total_partidos: Número de encuentros en que participaron los anotadores. Total_goles: Número de anotaciones obtenidos durante los partidos de ese periodo. Total_Penal: Número de goles obtenidos desde el punto penal. Total_owngoals: Número de autogoles en que incurrieron los anotadores de ese periodo. Req7: Listado de anotadores ordenados por sus características
Implementado (Sí/No)	Si se implemento, en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Organizar las listas (Merge Sort)	O(NLogN)
Filtrar por Fechas (FiltroEquipos)	O(N)
Evaluar si elemento esta en lista (Dentro de un for que itera por partidos)	O(N)
Recorrer una lista para explorar sus elementos (Dentro de un for que itera en goleadores)	O(N)
Comparar datos (Dentro de dos for, uno por goleadores, uno por jugadores y otro por partidos)	O(N*M*T) Donde N son los datos de partidos y M son datos de goleadore sy T son datos de jugadores
Tamaño de la lista lt.size()	O(N)
TOTAL	***O(NLogN)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones.

Procesadores	**AMD Ryzen 5 4500U (with Radeon Graphics) CPU @ 2.38 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Los parametros de entra asignados fueron los siguientes:

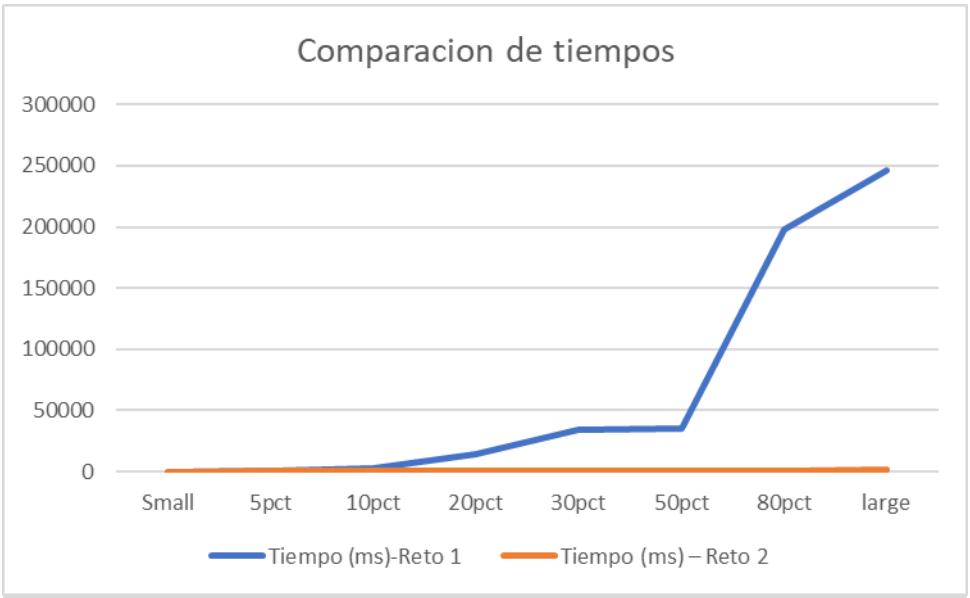
- NPuntaje = 5 puntos
- Torneo= Copa América

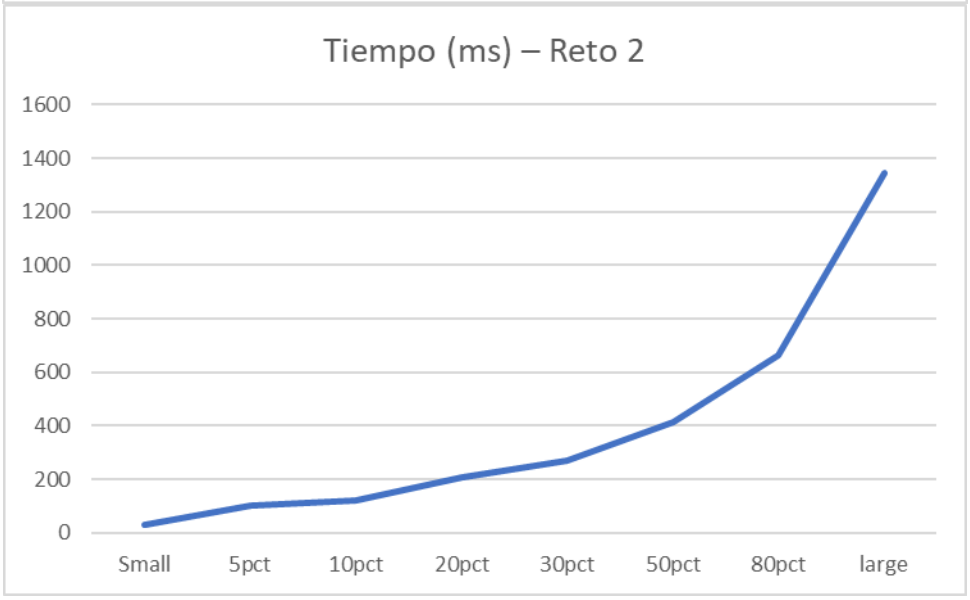
Entrada	**Tiempo (ms)-Reto 1**	**Tiempo (ms) – Reto 2**
small	66.18	30.27
5 pct	774.63	104.2378
10 pct	3111.67	120.8930
20 pct	14730.89	209.5685
30 pct	33803.37	270.4799
50 pct	35444.36	413.1320
80 pct	198278.16	664.3928
large	245732.21	1344.0641

Tablas de datos

Muestra	Salida	Tiempo (ms) -Reto 1	Tiempo (ms) – Reto 2
small	Listado de anotadores	66.18	30.27
5 pct	Listado de anotadores	774.63	104.2378
10 pct	Listado de anotadores	3111.67	120.8930
20 pct	Listado de anotadores	14730.89	209.5685
30 pct	Listado de anotadores	33803.37	270.4799
50 pct	Listado de anotadores	35444.36	413.1320
80 pct	Listado de anotadores	198278.16	664.3928
large	Listado de anotadores	245732.21	1344.0641

Graficas





Análisis

Este requerimiento tiene una complejidad de NlogN, debido al mergeSort, pese a la implementación de mapas, la complejidad sigue definida por aquella función de ordenamiento. Aun asi cuando comparamos con los resultados anteriores, podemos observar que los tiempos de procesamiento son bastante menores a los obtenidos en el reto 1.