ANÁLISIS DEL RETO

Jhonny Armando Hortua Oyola, 202111749, j.hortuao@uniandes.edu.co

Gabriel Esteban González Carrillo, 202014375, g.gonzalezc@uniandes.edu.co

Adrian Esteban Velasquez Solano - 202222737, a.velasquezs@uniandes.edu.co

Requerimiento 1

```python
# REQ 1
def first_last(informacion):

    if lt.size(informacion) < 6:
        return informacion
    primeros = lt.subList(informacion,1,3)
    ultimos = lt.subList(informacion,lt.size(informacion)-2,3)
    respuesta = lt.newList('ARRAY_LIST')
    for i in lt.iterator(primeros):
        lt.addLast(respuesta,i)
    for i1 in lt.iterator(ultimos):
        lt.addLast(respuesta,i1)
    return respuesta
```

```python
def print_games_played(data_structs,games,team,condition):
    """
    Función que soluciona el requerimiento 1
    """

    req1 = data_structs['req1']
    results = data_structs['results']
    list1 = lt.newList("SINGLE_LINKED")
    total_teams = 0
    total_matches_team = 0
    total_matches_condition = 0
    for val in lt.iterator(results):
        total_teams +=1
        if val["home_team"] == team or val["away_team"] == team:
            total_matches_team +=1
        if condition == "home":
            if val["home_team"] == team:
                total_matches_condition +=1
                lt.addLast(list1,val)
        if condition == "away":
            if val["away_team"] == team:
                total_matches_condition +=1
                lt.addLast(list1,val)
        if condition == "indiferent":
            if val["home_team"] == team:
                total_matches_condition +=1
                lt.addLast(list1,val)
            if val["away_team"] == team:
                total_matches_condition +=1
                lt.addLast(list1,val)

    list2 = lt.newList("SINGLE_LINKED")
    count = 0
    for num in lt.iterator(list1):
        if count < games:
            lt.addLast(list2,num)
            count += 1
```

Descripción

En este requerimiento se reciben como parámetros de entrada el número de partidos, Nombre del equipo, Condición, dado estos parámetros se filtran los mapas con los datos y se retorna un tad lista con la información organizada y filtrada

Entrada	Numero de partidos, Nombre del equipo, Condicion
Salidas	Lista de N partidos jugados por un equipo en una condicion
Implementado (Sí/No)	Si, Jhonny Hortua

Análisis de complejidad

Pasos	**Complejidad**
Paso 1	O(n)
Paso 2	O(n^2)
Paso ….	O(n)
TOTAL	O(n^2)

Pruebas Realizadas

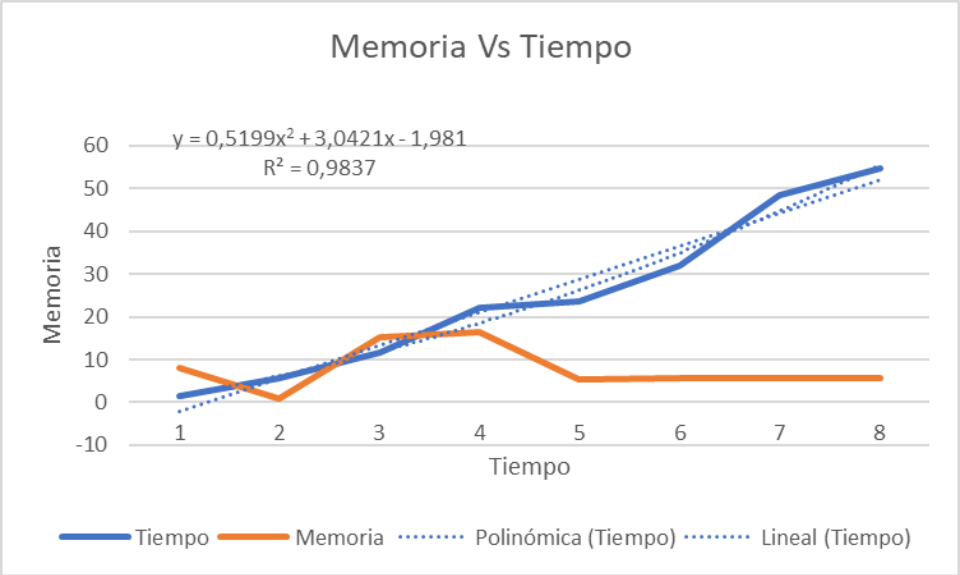
Las pruebas fueron realizadas con una máquina de las siguientes especificaciones. Del estudiante 2

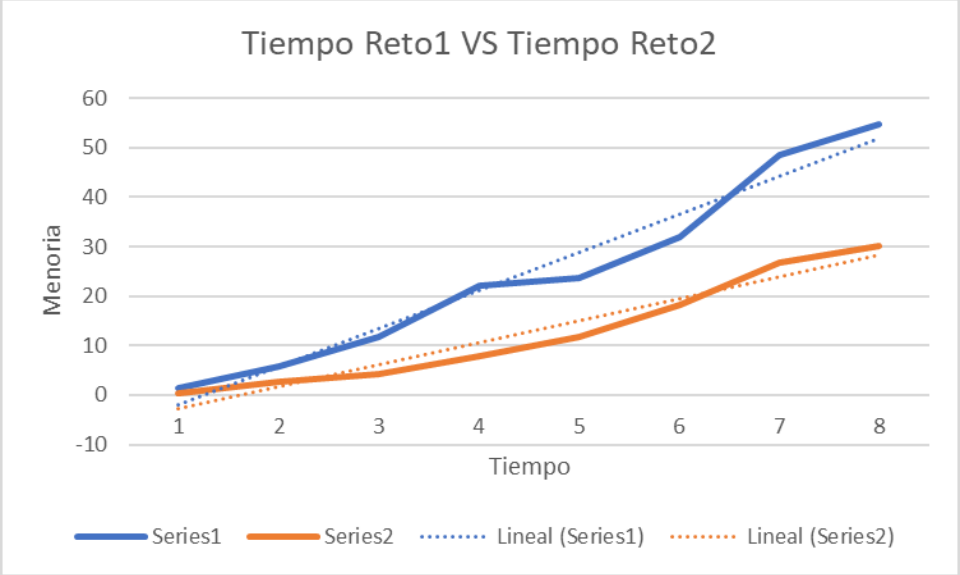
Procesadores	**AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

	Reto 2		**Reto 1**
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	1.467	8.023	0.485
5%	5.751	0.890	2.611
10%	11.659	15.140	4.272
20%	22.010	16.380	7.778
30%	23.726	5.484	11.802
50%	32.003	5.515	18.234
80%	48.471	5.618	26.729
LARGE	54.646	5.507	30.021

Graficas





Análisis

En el requerimiento 1 del reto 2 como se muestra es mas eficiente y demora menos tiempo que el algoritmo anterior debido a sus modificaciones para aumentar su eficiencia

Requerimiento 2

```
def first_last(informacion):

    if lt.size(informacion) < 6:
        return informacion
    primeros = lt.subList(informacion,1,3)
    ultimos = lt.subList(informacion,lt.size(informacion)-2,3)
    respuesta = lt.newList('ARRAY_LIST')
    for i in lt.iterator(primeros):
        lt.addLast(respuesta,i)
    for i1 in lt.iterator(ultimos):
        lt.addLast(respuesta,i1)
    return respuesta
```

```
def Goals_for_player(data_structs,n_goals,player):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2

    goalscorers = data_structs['goalscorers']
    list1 = lt.newList("SINGLE_LINKED")
    list2 = lt.newList("SINGLE_LINKED")
    total_goals = 0
    goals_penalty = 0
    for i in lt.iterator(goalscorers):
        total_goals += 1
        if i["scorer"] == player:
            if i["penalty"] == "True":
                goals_penalty += 1
            lt.addLast(list1,i)

    size_goals_player = lt.size(list1)
    count = 0
    for num in lt.iterator(list1):
        if count < n_goals:
            lt.addLast(list2,num)
            count += 1

    return first_last(list2),total_goals,size_goals_player,goals_penalty
```

El requerimiento 2 nos pide encontrar el numero de goles anotados por un jugador especifico, teniendo en cuenta los parametros dados

Entrada	Datos,Los goles,El nombre del jugador
Salidas	Los N goles que hizo el jugador
Implementado (Sí/No)	Si, Jhonny Hortua

Análisis de complejidad

Pasos	Complejidad
Paso 1	O(1)
Paso 2	O(n)
Paso ….	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

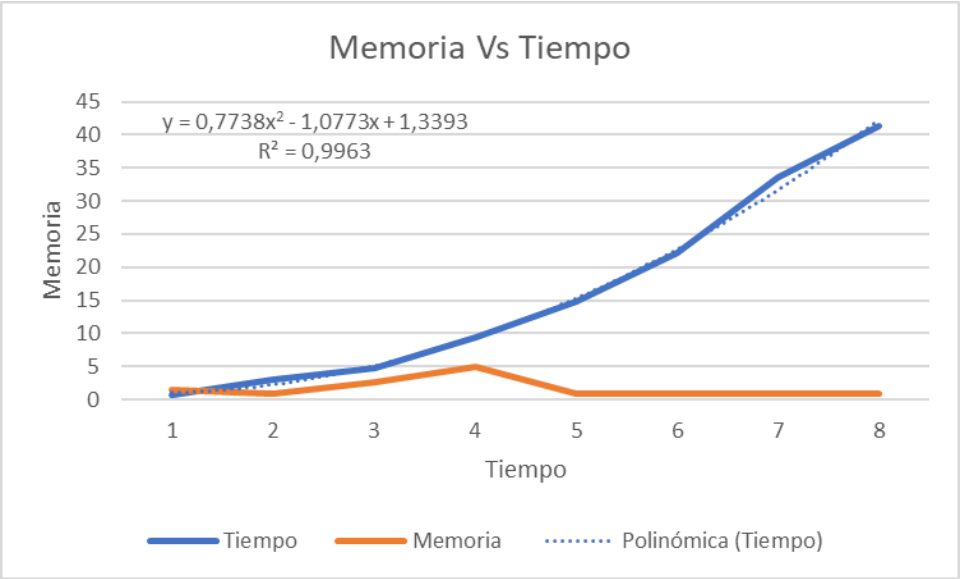
Las pruebas fueron realizadas con una máquina de las siguientes especificaciones. Del estudiante 2

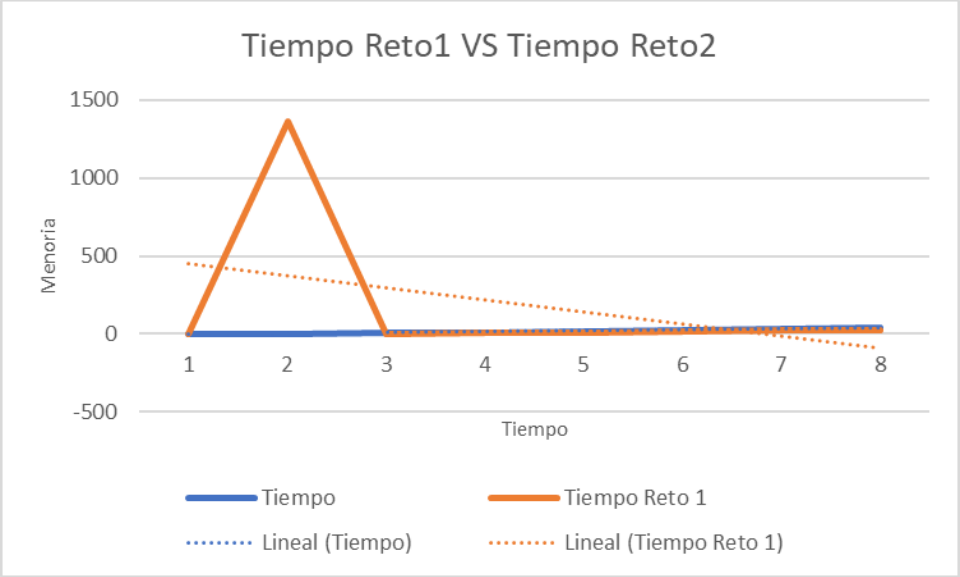
Procesadores	**AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

	Reto 2		**Reto 1**
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	0.764	1.484	0.350
5%	3.051	0.828	1.366
10%	4.687	2.562	2.839
20%	9.451	4.968	5.348
30%	14.735	0.810	7.867
50%	22.153	0.890	13.965
80%	33.589	0.859	22.120
LARGE	41.351	0.863	27.434

Graficas





Análisis

En el requerimiento 2 del reto 2 como se muestra es más eficiente y demora menos tiempo que el algoritmo anterior debido a sus modificaciones para aumentar su eficiencia, como se ve en el requerimiento 2 del reto 1 tiene un pico en el que su tiempo de ejecución aumenta notoriamente, lo que se logró corregir para esta nueva versión del algoritmo

Requerimiento 3

```python
def first_last(informacion):

    if lt.size(informacion) < 6:
        return informacion
    primeros = lt.subList(informacion,1,3)
    ultimos = lt.subList(informacion,lt.size(informacion)-2,3)
    respuesta = lt.newList('ARRAY_LIST')
    for i in lt.iterator(primeros):
        lt.addLast(respuesta,i)
    for i1 in lt.iterator(ultimos):
        lt.addLast(respuesta,i1)
    return respuesta
```

```python
def filter_date(list,fecha_inicial,fecha_final):
    filter = lt.newList("SINGLE_LINKED")
    for p in lt.iterator(list):
        if(p["date"]) <= fecha_final and (p["date"]) >= fecha_inicial:
            lt.addLast(filter,p)
    return filter
```

```python
def Consult_Period_Matches(data_structs,team,start_date,end_date):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    results = data_structs['results']
    lst1 = lt.newList("SINGLE_LINKED")
    lst2 = lt.newList("SINGLE_LINKED")
    home_games = lt.newList("SINGLE_LINKED")
    away_games = lt.newList("SINGLE_LINKED")
    total_teams = 0
    total_games_team = 0
    for pl in lt.iterator(results):
        total_teams  +=1
    for pl in lt.iterator(filter_date(results,start_date,end_date)):
        if pl["home_team"] == team:
            total_games_team += 1
            lt.addLast(lst1,pl)
            lt.addLast(home_games,pl)
        if pl["away_team"] == team:
            total_games_team += 1
            lt.addLast(lst1,pl)
            lt.addLast(away_games,pl)

    for y in lt.iterator(lst1):
        lt.addFirst(lst2,y)

    return first_last(lst2),total_teams,total_games_team,lt.size(home_games),lt.size(away_games)
```

Descripción

El requerimiento 3 nos pide encontrar los partidos jugados por un equipo en un periodo de tiempo, filtrando por los parametros de etrada dados

Entrada	Estructuras de datos del modelo, equipó, fecha inicial, fecha final

Salidas	Partidos disputados por un equipo en un periodo de tiempo especifico
Implementado (Sí/No)	Si, Jhonny Hortua

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(n)
Paso 2	O(n log n)
Paso ….	O(n)
TOTAL	***O(n log n)***

Pruebas Realizadas

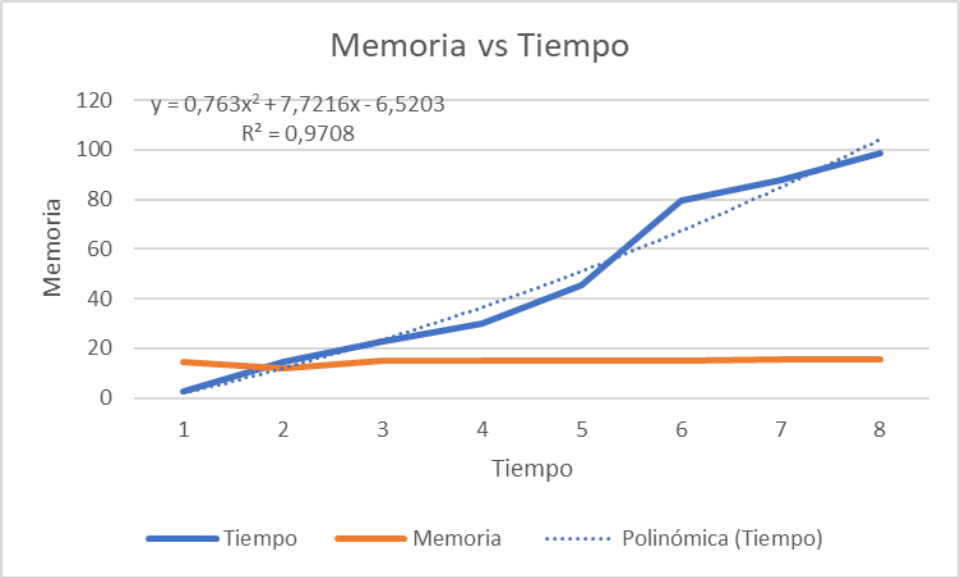
Las pruebas fueron realizadas con una máquina de las siguientes especificaciones. Del estudiante 2

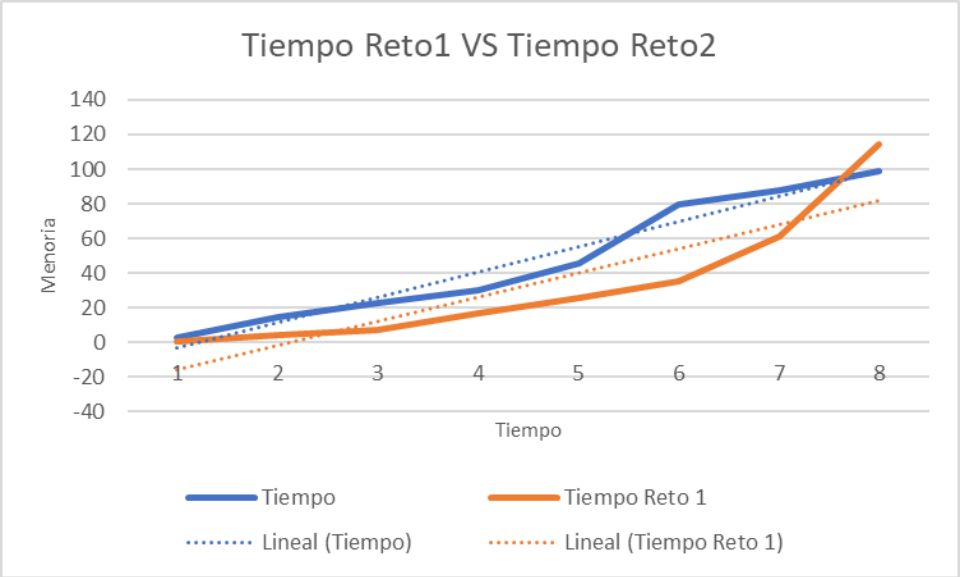
Procesadores	**AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

	Reto 2		**Reto 1**
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	2.754	14.640	0.932
5%	14.315	12.000	4.532
10%	22.717	15.234	6.930
20%	30.112	15.238	16.565
30%	45.373	15.265	25.210
50%	79.677	15.274	35.630
80%	87.740	15.296	61.023
LARGE	98.78	15.632	114.210

Graficas

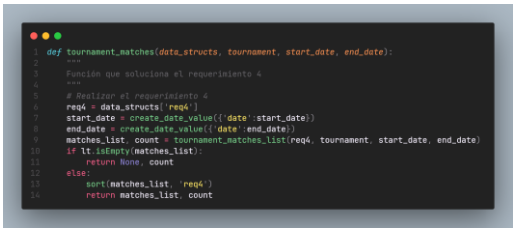




Análisis

En el requerimiento 3 del reto 2 como se muestra es más eficiente y demora menos tiempo que el algoritmo anterior debido a sus modificaciones para aumentar su eficiencia, al principio se ve un poco más lento, pero con los archivos más grandes es más eficiente por lo que para trabajar con datos de gran magnitud muestra ser el más eficiente

Requerimiento 4



```
def tournament_matches(data_structs, tournament, start_date, end_date):
    """
    Función que soluciona el requerimiento 4
    """
    # Realizar el requerimiento 4
    req4 = data_structs['req4']
    start_date = create_date_value({'date':start_date})
    end_date = create_date_value({'date':end_date})
    matches_list, count = tournament_matches_list(req4, tournament, start_date, end_date)
    if lt.isEmpty(matches_list):
        return None, count
    else:
        sort(matches_list, 'req4')
        return matches_list, count
```

4.1: tournament_matches()

```
def tournament_matches_list(req4, tournament, start_date, end_date):
    r_req4 = lt.getElement(req4, 1) # mapa de torneos
    # determinar si existe o no el torneo en el mapa
    exists_tournament = mp.contains(r_req4, tournament)
    if not exists_tournament:
        empty_list = lt.newList()
        return empty_list, None
    # si existe, retornar la lista
    entry = mp.get(r_req4, tournament)
    tournament_list = me.getValue(entry)['matches']
    # obtener el mapa de shootouts
    s_req4 = lt.getElement(req4, 3) # mapa de shootouts por fecha
    # crear la lista respuesta
    matches_list = lt.newList('ARRAY_LIST')
    # adquirir la cantidad total de torneos en el mapa
    tournaments_in_map = mp.keySet(r_req4)
    total_tournaments = lt.size(tournaments_in_map)
    # adquirir la cantidad de partidos del torneo
    tournament_matches = lt.size(tournament_list)
    # initializar los contadores
    count = {'total_tournaments':total_tournaments,
             'total_tournament_matches':tournament_matches,
             'total_matches_in_range':0,
             'total_involved_countries':[],
             'total_involved_cities':[],
             'total_penalty_matches':0
             }
    # iterar la lista del torneo
    for match in lt.iterator(tournament_list):
        match_date_value = create_date_value(match)
        # verificar si la partida está en el rango de fechas
        if (match_date_value > start_date) and (match_date_value < end_date):
            count['total_matches_in_range'] += 1
            # crear el elemento de la partida
            element = tournament_matches_element(match, s_req4, count)
            # añadir el elemento de la partida a la lista respuesta
            lt.addFirst(matches_list, element)
    # cambiar contadores
    count['total_involved_countries'] = len(count['total_involved_countries'])
    count['total_involved_cities'] = len(count['total_involved_cities'])
    # retornar la lista respuesta y el contador
    return matches_list, count
```

4.2: tournament_matches_list()

```
def tournament_matches_element(match, s_req4, count):
    # modificar contadores
    match_country = match['country']
    match_city = match['city']
    if match_country not in count['total_involved_countries']:
        count['total_involved_countries'].append(match_country)
    if match_city not in count['total_involved_cities']:
        count['total_involved_cities'].append(match_city)
    # crear el elemento vacio
    element = ['Unkown']*10
    # añadir los valores pertinentes del partido
    # al elemento
    i = 0
    for value in match.values():
        if i < len(match)-1:
            element[i] = value
            i += 1
    # determinar si la fecha se encuentra en el mapa
    # de shootouts
    match_date = match['date']
    exists_date = mp.contains(s_req4, match_date)
    # si se encuentra, añadir los valores pertientes
    # al elemento
    found = False
    if exists_date:
        # adquirir la lista de partidos de la fecha
        entry = mp.get(s_req4, match_date)
        date_list = me.getValue(entry)['matches']
        # adquirir los valores pertinentes del partido
        match_home_team = match['home_team']
        match_away_team = match['away_team']
        # buscar el partido en la lista de la fecha
        for shootout_match in lt.iterator(date_list):
            count['total_penalty_matches'] += 1
            shootout_home_team = shootout_match['home_team']
            shootout_away_team = shootout_match['away_team']
            if match_home_team == shootout_home_team and match_away_team == shootout_away_team:
                found = True
                element[8] = True
                element[9] = shootout_match['winner']
                return element
    if not found:
        element[8] = False
    return element
```

4.3: tournament_matches_element()

Descripción

Este requerimiento pretende encontrar todos los partidos de un torneo en específico y retornar una lista compuesta de estos. Para esto se deben recorrer dos estructuras de datos, un mapa de resultados y un mapa de goleadores. El mapa de resultados utiliza el índice de torneo, donde cada torneo contiene una lista con la información de todas las partidas del torneo. Por otro lado, el mapa de goleadores utiliza el índice de fecha, donde cada fecha tiene una lista de los goleadores que anotaron en esa fecha en específico. Una vez se recolecta la información de todas las partidas del torneo en el rango de fechas, se ordena la lista respuesta y se retorna.

Entrada	Estructuras de datos del modelo, nombre del torneo, fecha inicial y final de búsqueda.
Salidas	Una lista con la información de todos los partidos del torneo de entrada entre las fechas especificadas.
Implementado (Sí/No)	Sí. Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
4.1	Sacar la estructura de datos del modelo	O(1)
4.1	Calcular el valor de la fecha (x2)	O(1)
4.1	Invocar 4.2	O(1)
4.2	Determinar si existe el torneo en el mapa de torneos de resultados	O(1)
4.2	Obtener la lista de partidos	O(1)
4.2	Operaciones de tamaño y contadores	O(1)
4.2	Recorrer el iterator de la lista de partidos del torneo	O(t)
4.2	Invocar 4.3	O(1)
4.3	Modificación de contadores y demás operaciones aritméticas	O(1)
4.3	Recorrer los valores de la partida y añadirlos a la lista respuesta	O(1)
4.3	Determinar si existe la fecha de la partida en el mapa de fechas de shootouts	O(1)
4.3	Recorrer la lista de la fecha de la partida en el mapa de fechas de shootouts	O(f)
4.1	Ordenar	O(k)
4.1	Retornar la lista respuesta y el contador	O(1)
Total		O(tf + k)

En comparación a la complejidad del reto 1, es claro que la implementación de mapas reduce significativamente la complejidad del requerimiento, debido a que los recorridos son mucho más pequeños.

Pruebas Realizadas

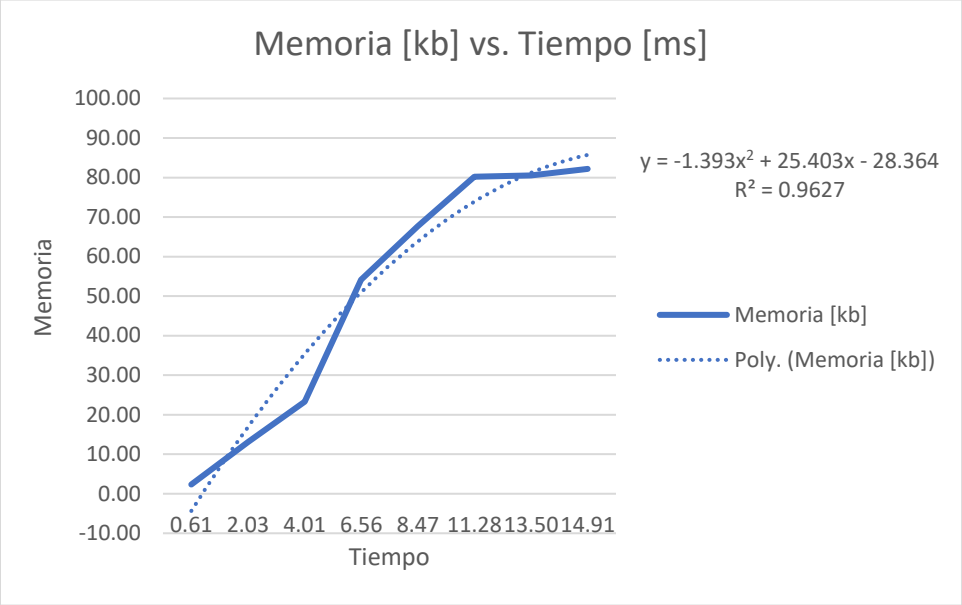
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el torneo **Copa América,** fecha inicial de **1955-06-01**, y fecha final de **2022-06-30.**

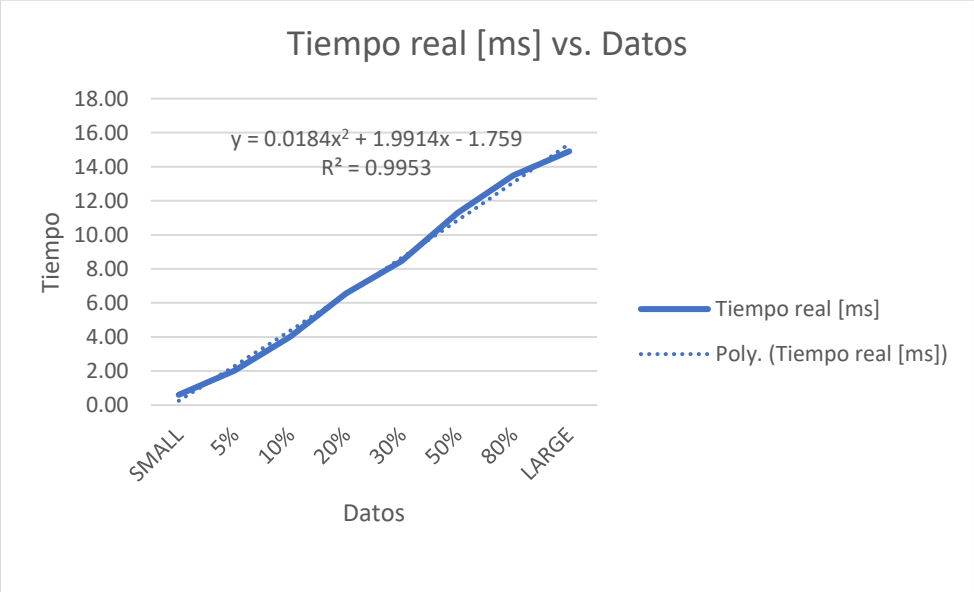
Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

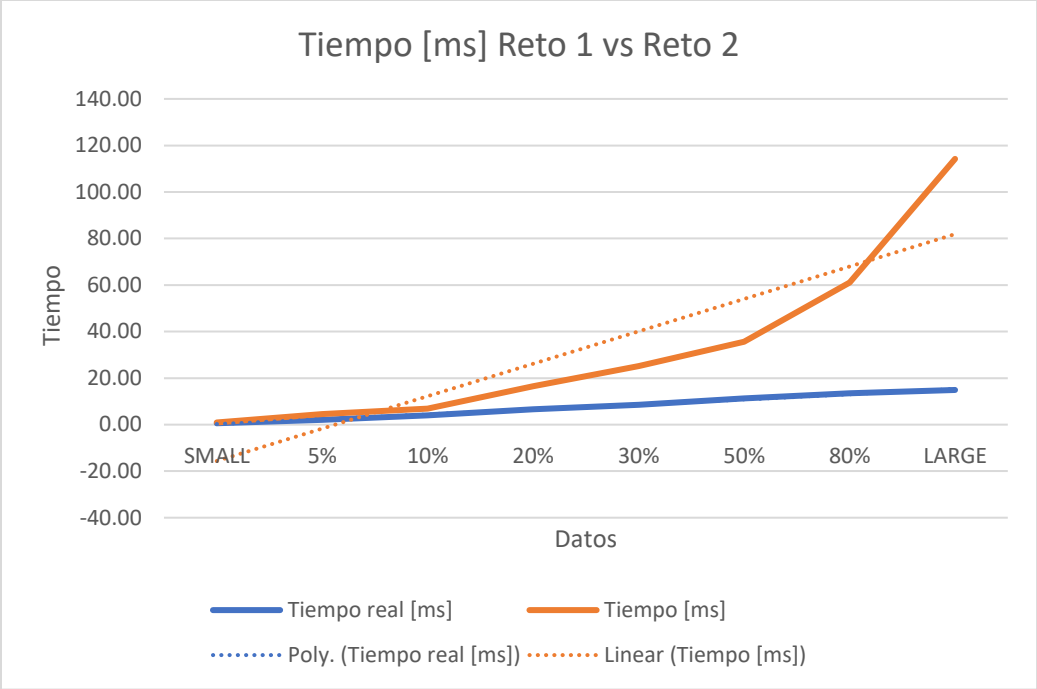
Tablas de datos

	Reto 2		**Reto 1**
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	0.61	2.34	0.932
5%	2.03	13.08	4.532
10%	4.01	23.26	6.93
20%	6.56	54.11	16.565
30%	8.47	67.68	25.21
50%	11.28	80.21	35.63
80%	13.50	80.57	61.023
LARGE	14.91	82.18	114.21

Graficas


Memoria [kb] vs. Tiempo [ms]
100.00
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
-10.00
Memoria
0.61 2.03 4.01 6.56 8.47 11.28 13.50 14.91
Tiempo
y = -1.393x² + 25.403x - 28.364
R² = 0.9627
Memoria [kb]
Poly. (Memoria [kb])


Tiempo real [ms] vs. Datos
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Tiempo
y = 0.0184x² + 1.9914x - 1.759
R² = 0.9953
SMALL
5%
10%
20%
30%
50%
80%
LARGE
Datos
Tiempo real [ms]
Poly. (Tiempo real [ms])



Análisis

Como se ve en las tablas y en las gráficas, en cuanto al tiempo transcurrido, la implementación del reto 2 es mucho más rápida en comparación al reto 1. Esto confirma la hipótesis que, debido al cambio significativo en cómo se construyen las estructuras de datos del requerimiento, la complejidad de las funciones y, por ende, el tiempo de ejecución resulta menor en el reto 2. De hecho, al comparar los gráficos se puede predecir un comportamiento cercano al lineal en el reto 2, mientras que en el reto 1, el comportamiento era más cercano al cuadrático. Por otro lado, implementando separate chaining, se puede ver que el uso de memoria tiene un comportamiento cercano al logarítmico, puesto que a pesar de con pocos datos el crecimiento es alto, se estabiliza a medida que crece la cantidad de datos.

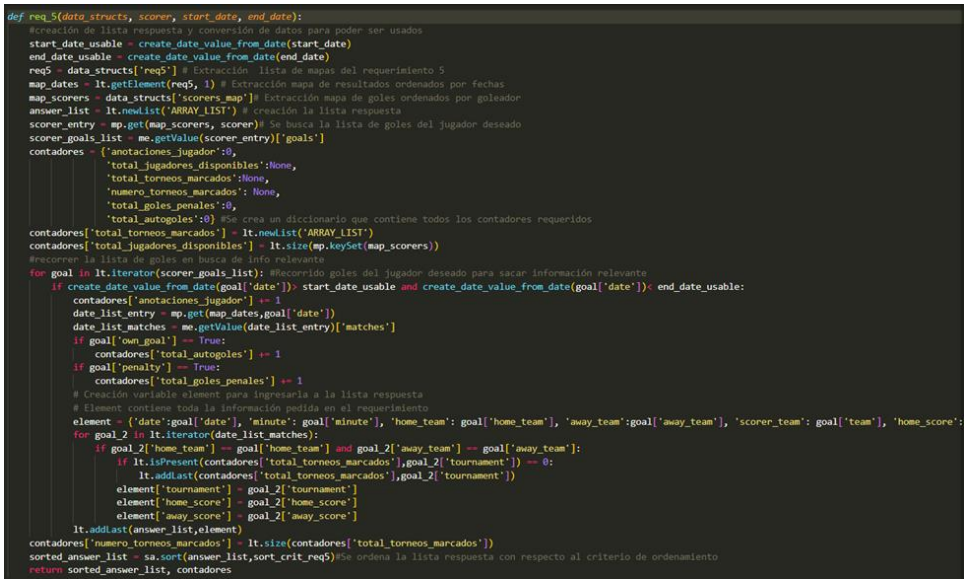
Requerimiento 5

```
def map_by_scorers(data_structs, data):
    # sacar el mapa de los scorers
    scorers = data_structs['scorers_map']
    # guardar el nombre del scorer
    data_scorer_name = data['scorer']
    # determinar si existe o no el scorer
    exists_scorer = mp.contains(scorers, data_scorer_name)
    # si existe, obtener el valor asociado
    if exists_scorer:
        entry = mp.get(scorers, data_scorer_name)
        scorer = me.getValue(entry)
    # si no existe, crear el valor y la llave del scorer
    # en el mapa de scorers
    else:
        scorer = {'scorer':data_scorer_name,
                  'goals': None}
        scorer['goals'] = lt.newList('ARRAY_LIST')
        mp.put(scorers, data_scorer_name, scorer)
    # añadir el gol la lista del scorer en el mapa de
    # scorers
    lt.addLast(scorer['goals'], data)
```

5.1 : Creación de la estructura de datos para archivo ‘Goalscorers’

```
def req5_results_map(data_structs, data):
    req5 = data_structs['req5']
    #revisar si la lista está vacia para agregar el mapa
    if lt.isEmpty(req5):
        map_results = mp.newMap(1000,maptype='CHAINING',loadfactor=5)
        lt.addFirst(req5,map_results)
    #si existe entonces se sapa el mapa para usarlo
    map_results = lt.getElement(req5,1)
    #Se procede a crear el codigo que agrega información
    date_info = data['date']
    date_exist = mp.contains(map_results,date_info)
    if date_exist:
        #si existe se agrega la info a dicha llave existente
        entry = mp.get(map_results, date_info)
        date = me.getValue(entry)
    #si no existe se crea la pareja llave valor en el mapa
    else:
        date = {'date':date_info,
                'matches':None}
        date['matches'] = lt.newList('ARRAY_LIST')
        mp.put(map_results, date_info, date)
    lt.addLast(date['matches'], data)
```

5.2: Creación de la estructura de datos para archivo ‘Results’



```
def req_5(data_structs, scorer, start_date, end_date):
    #creación de lista respuesta y conversión de datos para poder ser usados
    start_date_usable = create_date_value_from_date(start_date)
    end_date_usable = create_date_value_from_date(end_date)
    req5 = data_structs['req5'] # Extracción  lista de mapas del requerimiento 5
    map_dates = lt.getElement(req5, 1) # Extracción mapa de resultados ordenados por fechas
    map_scorers = data_structs['scorers_map']# Extracción mapa de goles ordenados por goleador
    answer_list = lt.newList('ARRAY_LIST') # creación la lista respuesta
    scorer_entry = mp.get(map_scorers, scorer)# Se busca la lista de goles del jugador deseado
    scorer_goals_list = me.getValue(scorer_entry)['goals']
    contadores = {'anotaciones_jugador':0,
                  'total_jugadores_disponibles':None,
                  'total_torneos_marcados':None,
                  'numero_torneos_marcados': None,
                  'total_goles_penales':0,
                  'total_autogoles':0} #Se crea un diccionario que contiene todos los contadores requeridos
    contadores['total_torneos_marcados'] = lt.newList('ARRAY_LIST')
    contadores['total_jugadores_disponibles'] = lt.size(mp.keySet(map_scorers))
    #recorrer la lista de goles en busca de info relevante
    for goal in lt.iterator(scorer_goals_list): #Recorrido goles del jugador deseado para sacar información relevante
        if create_date_value_from_date(goal['date'])> start_date_usable and create_date_value_from_date(goal['date'])< end_date_usable:
            contadores['anotaciones_jugador'] += 1
            date_list_entry = mp.get(map_dates,goal['date'])
            date_list_matches = me.getValue(date_list_entry)['matches']
            if goal['own_goal'] == True:
                contadores['total_autogoles'] += 1
            if goal['penalty'] == True:
                contadores['total_goles_penales'] += 1
            # Creación variable element para ingresarla a la lista respuesta
            # Element contiene toda la información pedida en el requerimiento
            element = {'date':goal['date'], 'minute': goal['minute'], 'home_team': goal['home_team'], 'away_team':goal['away_team'], 'scorer_team': goal['team'], 'home_score':
            for goal_2 in lt.iterator(date_list_matches):
                if goal_2['home_team'] == goal['home_team'] and goal_2['away_team'] == goal['away_team']:
                    if lt.isPresent(contadores['total_torneos_marcados'],goal_2['tournament']) == 0:
                        lt.addLast(contadores['total_torneos_marcados'],goal_2['tournament'])
                    element['tournament'] = goal_2['tournament']
                    element['home_score'] = goal_2['home_score']
                    element['away_score'] = goal_2['away_score']
            lt.addLast(answer_list,element)
    contadores['numero_torneos_marcados'] = lt.size(contadores['total_torneos_marcados'])
    sorted_answer_list = sa.sort(answer_list,sort_crit_req5)#Se ordena la lista respuesta con respecto al criterio de ordenamiento
    return sorted_answer_list, contadores
```

5.3: Búsqueda de información en las estructuras de datos creadas.

Descripción

La solución del requerimiento se puede resumir en dos procesos generales, la creación de las estructuras de datos adecuadas para el requerimiento y la búsqueda de información en dicha estructura. El primer proceso se muestra en las figuras 5.1 y 5.2. y dichas funciones tienen el objetivo de crear mapas adecuados para el requerimiento. La figura 5.1 crea un mapa en el cual ordena la información del archivo ‘Goalscorers’ usando como índice el nombre del jugador que marcó el gol y sus valores son la información completa de los goles marcados por dicho jugador. La figura 5.2 crea un mapa el cual ordena la información del archivo ‘Results’ usando como índice la fecha del partido y su valor contiene toda la información de los partidos disputados en esa fecha. Por otro lado, la figura 5.3 se muestra cómo se recorren las diferentes estructuras de datos con el fin de extraer la información necesaria para agregarla a una lista respuesta que entrega la función como respuesta a el requerimiento.

Entrada	Estructura de datos del modelo, nombre del jugador, fecha inicial de la búsqueda y la fecha final de la búsqueda.
Salidas	Un ADT lista con los goles obtenidos por el jugador. Se agregan otras estadísticas secundarias.
Implementado (Sí/No)	Si – Gabriel Esteban González Carrillo - 202014375

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Declaración de variables y obtención de información de la carga de datos.	O(1)
Recorrido estructura de datos completa	O(N^2)

Usar lt.addlast()	O(n) en ARRAY_LIST y O (1) en SINGLE_LINKED
TOTAL	***O(N^2)***

Pruebas Realizadas

Las pruebas realizadas se realizaron en una máquina con las siguientes especificaciones utilizando estos datos de entrada: Jugador 'Ali Daei' entre las fechas '1999-03-25' y '2021-11-23'.

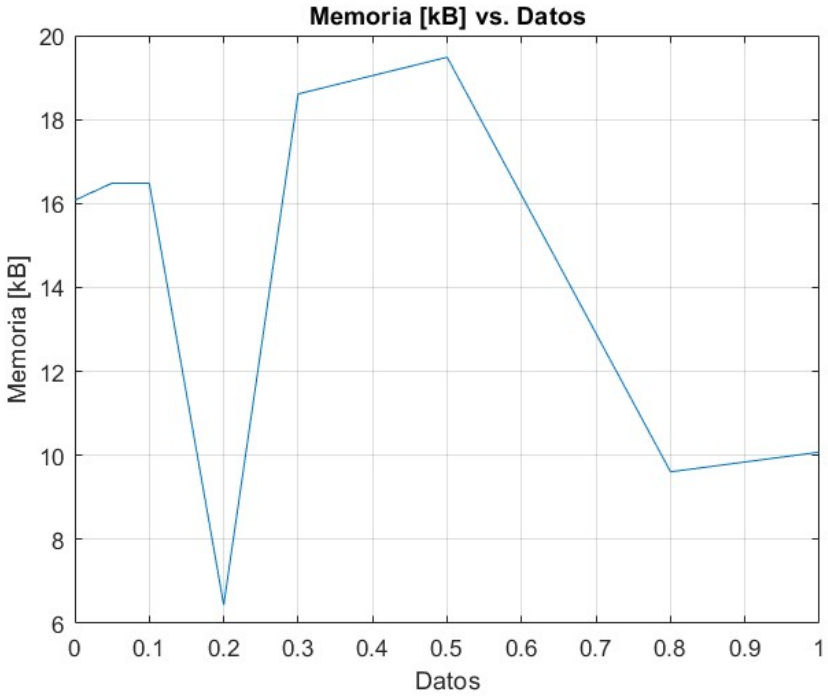
Procesadores	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.20 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 11

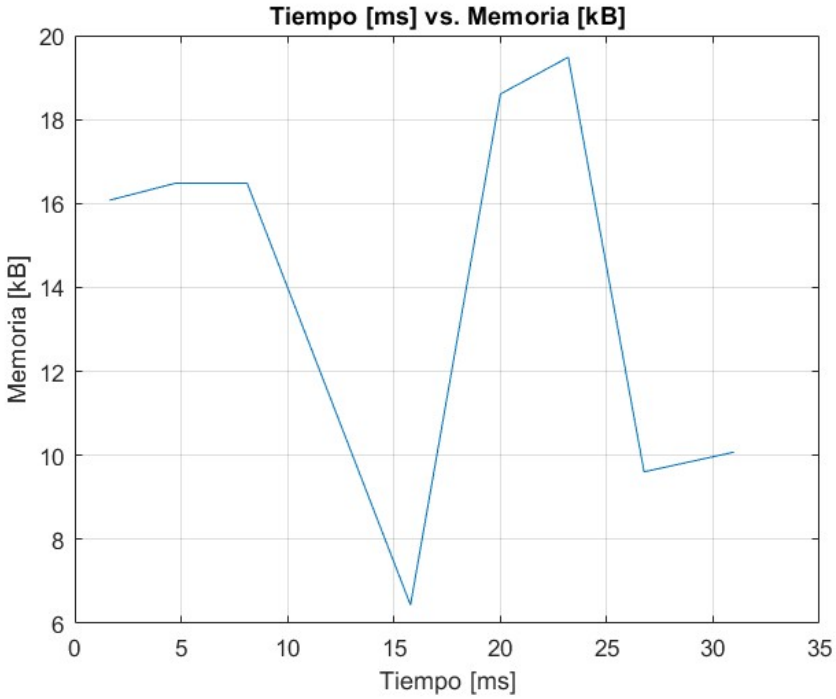
Tablas de datos

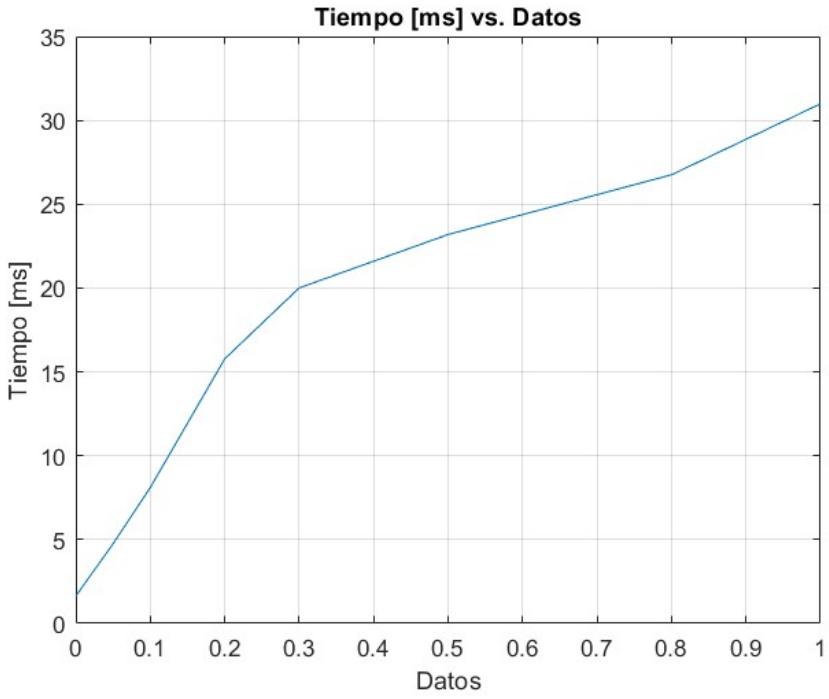
	Reto 2		Reto 1
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	1.632	16.078	0.61
5%	4.747	16.484	2.80
10%	8.1	16.484	5.58
20%	15.778	6.438	12.46
30%	20.007	18.609	17.99
50%	23.193	19.484	29.76
80%	26.755	9.609	48.31
LARGE	30.986	10.078	60.78

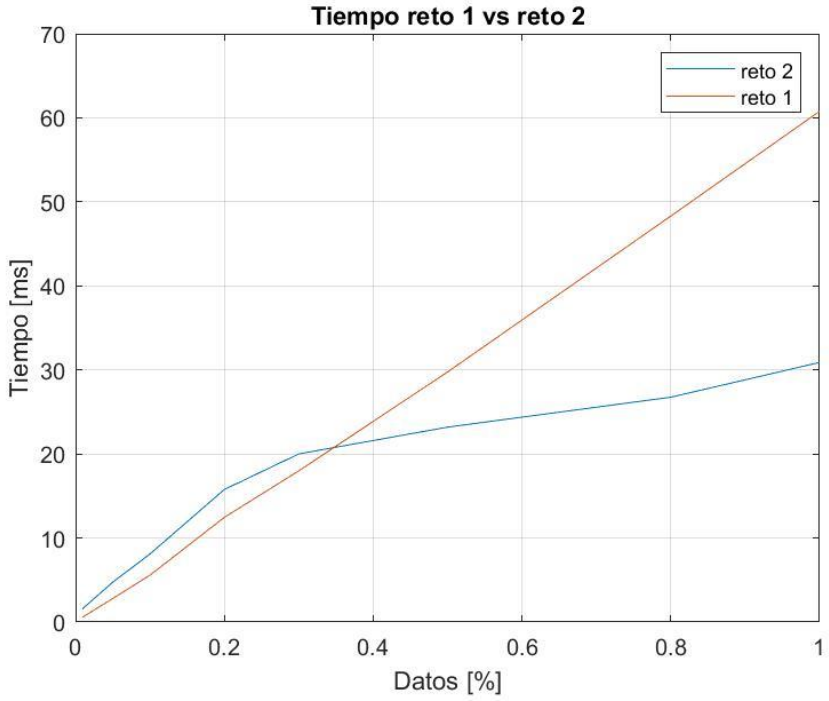
Gráficas

Las gráficas de la anterior tabla se muestran a continuación:









Análisis

Como se puede evidenciar en las anteriores figuras, los tiempos del reto 2 son mucho mejores que los del reto 1 a partir del 30% de uso de datos, esto índica como el uso de mapas a la hora de la implementación de los requerimientos tiene un muy buen rendimiento tomando el tiempo como factor

diferenciador. Esta mejora se relaciona con la creación de mapas óptimos y exclusivos para la búsqueda de información del requerimiento. Por otro lado, también se puede apreciar como el comportamiento de la memoria con respecto a la cantidad de datos ingresados tiene un comportamiento muy irregular. En la tabla memoria vs tiempo se puede ver como no necesariamente a medida que se aumenta el tamaño de los datos también aumenta la memoria utilizada.

Requerimiento 6

```python
def classify_teams(data_structs, n, tournament, year):
    """
    Función que soluciona el requerimiento 6
    """
    # Realizar el requerimiento 6
    req6 = data_structs['req6']
    year = float(year)
    classified_teams, count = classify_teams_list(req6, tournament, year)

    if count != None:
        p = 0
        popular_city = 'Unkown'
        for city in count['cities']:
            if count['cities'][city] > p:
                p = count['cities'][city]
                popular_city = city
        count['total_teams'] = len(count['total_teams'])

    if lt.isEmpty(classified_teams):
        return None, None, None
    else:
        sort(classified_teams, 'req6')
        top_n_classified_teams = top_n_classified_teams_list(classified_teams, n)
        return top_n_classified_teams, count, popular_city
```

6.1: classify_teams()

```python
def classify_teams_list(req6, tournament, year):
    classified_teams = lt.newList('ARRAY_LIST')
    # sacar las estructuras de datos organizadas para el Req 6 del ADT List req6
    r_req6 = lt.getElement(req6, 1) # mapa de torneos, cada uno con un diccionario de equipos
    # retornamos el existe o no el torneo
    exists_tournament = mp.contains(r_req6, tournament)
    if not exists_tournament:
        return classified_teams, None
    # encontrar la cantidad total de torneos para contarlos
    tournaments_keys = mp.keySet(r_req6)
    # encontrar el mapa de estadios del torneo
    entry = mp.get(r_req6, tournament)
    teams_map = me.getValue(entry)['teams']
    # sacar las llaves de los equipos en el mapa
    team_keys = mp.keySet(teams_map)
    total_teams = lt.size(team_keys)
    count = {'total_tournaments': lt.size(tournaments_keys), # cantidad total de torneos
             'total_teams': [], # lista con los equipos del torneo en el rango
             'matches_in_range':0, # partidos evaluados
             'countries':[], # cada valor es el nombre del país
             'cities':{}} # ciudades totales, cada valor es el número de ocurrencias
    # obtener el mapa de fechas de goalscorers
    g_req6 = lt.getElement(req6, 2)
    # sacar el recorrido de los países del torneo
    for team_name in lt.iterator(team_keys):
        # crear las variables respuesta
        elem = []
        team_scorers = {}
        team_values = {'total_points':0,
                       'matches':0,
                       'goal_difference':0,
                       'penalty_points':0,
                       'own_goal_points':0,
                       'wins':0,
                       'draws':0,
                       'losses':0,
                       'goals_for':0,
                       'goals_against':0}
        # encontrar el id's lista de partidos del torneo
        entry = mp.get(teams_map,team_name)
        matches_list = me.getValue(entry)['matches']
        # sacar las llaves de los equipos en el mapa
        for match in lt.iterator(matches_list):
            match_date_value = create_date_value(match)
            # revisar el partido si está en el rango de fechas
            if (match_date_value>year) and (match_date_value<year+1):
                # contar los países
                count_countries_and_cities(match, count)
                # contar los partidos en el rango el igual que los equipos
                count['matches_in_range'] += 1
                if match['home_team'] not in count['total_teams']:
                    count['total_teams'].append(match['home_team'])
                if match['away_team'] not in count['total_teams']:
                    count['total_teams'].append(match['away_team'])
                # contar contadores del equipo
                classify_teams_match_values(match, team_name, team_values, team_scorers, g_req6)
        # calcular diferencia de goles del equipo
        team_values['goal_difference'] = team_values['goals_for'] - team_values['goals_against']
        # encontrar el mejor goleador
        top_team_scorer = top_scorer(team_scorers)
        # crear el elemento para agregar a la lista respuesta
        classify_teams_element(elem, team_name, team_values, top_team_scorer)
        # añadir el elemento a la lista respuesta
        lt.addFirst(classified_teams, elem)

    return classified_teams, count
```

6.2: classify_teams_list()

```python
def count_countries_and_cities(match, count):
    """Evaluates the city and country of the match


    Args:
        match (dict): dictionary with the match's information
        count (dict): dictionary with all the pertinent counters
    """
    match_country = match['country']
    match_city = match['city']
    if match_country not in count['countries']:
        count['countries'].append(match_country)
    if match_city not in count['cities']:
        count['cities'][match_city] = 1
    else:
        count['cities'][match_city] += 1
```

6.3: count_countries_and_cities

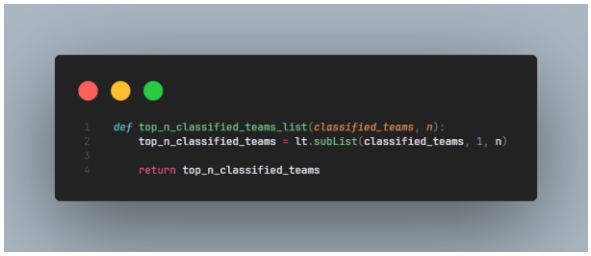
```python
def classify_teams_match_values(match, team_name, team_values, team_scorers, g_req6):
    # sacar información pertinente de los resultados del partido
    match_date = match['date']
    match_home_team = match['home_team']
    match_away_team = match['away_team']
    match_home_score = int(match['home_score'])
    match_away_score = int(match['away_score'])
    team_values['matches'] += 1
    # contar los puntos en las categorías pertinentes de acuerdo a los resultados
    # (goals_for, goals_against, wins, total_points)
    if team_name==match_home_team:
        team_values['goals_for'] += match_home_score
        team_values['goals_against'] += match_away_score
        if match_home_score > match_away_score:
            team_values['wins'] += 1
            team_values['total_points'] += 3
        elif match_home_score < match_away_score:
            team_values['losses'] += 1
        else:
            team_values['draws'] += 1
            team_values['total_points'] += 1
    elif team_name==match_away_team:
        team_values['goals_for'] += match_away_score
        team_values['goals_against'] += match_home_score
        if match_home_score > match_away_score:
            team_values['losses'] += 1
        elif match_home_score < match_away_score:
            team_values['wins'] += 1
            team_values['total_points'] += 3
        else:
            team_values['draws'] += 1
            team_values['total_points'] += 1
    ### contar información a los goleadores (penalty, own_goal y scorers)
    # verificar que exista la fecha
    exists_date = mp.contains(g_req6, match_date)
    if exists_date:
        # recorrer el mapa de equipos del torneo
        entry = mp.get(g_req6, match_date)
        goalscorers_list = me.getValue(entry)['matches']
        # recorrer la lista
        for goalscorer in lt.iterator(goalscorers_list):
            goalscorer_team = goalscorer['team']
            if goalscorer_team == team_name:
                # añadir o modificar goleador en team_scorers
                goalscorer_name = goalscorer['scorer']

                goalscorer_minute = goalscorer['minute']
                if goalscorer_minute == '':
                    goalscorer_minute = 0
                else:
                    goalscorer_minute = float(goalscorer_minute)

                if goalscorer_name not in team_scorers:
                    team_scorers[goalscorer_name] = {'name': goalscorer_name,
                                                     'goals': 1,
                                                     'matches': 1,
                                                     'total_minutes': goalscorer_minute,
                                                     'played_matches': [match_date]}

                else:
                    team_scorers[goalscorer_name]['goals'] += 1
                    team_scorers[goalscorer_name]['total_minutes'] += goalscorer_minute
                    if match_date not in team_scorers[goalscorer_name]['played_matches']:
                        team_scorers[goalscorer_name]['played_matches'].append(match_date)
                        team_scorers[goalscorer_name]['matches'] += 1
                # determinar si hubo o no penalty/own_goal y cambiar el contador
                goalscorer_own_goal = str(goalscorer['own_goal']).lower()
                goalscorer_penalty = str(goalscorer['penalty']).lower()
                if goalscorer_own_goal == 'true':
                    team_values['own_goal_points'] += 1
                if goalscorer_penalty == 'true':
                    team_values['penalty_points'] += 1
```

6.4: classify_teams_match_values

```python
def top_scorer(scorers):
    top_scorer = ['Unknown', 0, 1, 100*100]
    for scorer in scorers.values():
        if scorer['goals'] > top_scorer[1]:
            if (scorer['total_minutes']/scorer['matches']) < (top_scorer[3]/top_scorer[2]):
                top_scorer[0] = scorer['name']
                top_scorer[1] = scorer['goals']
                top_scorer[2] = scorer['matches']
                top_scorer[3] = scorer['total_minutes']
    if top_scorer[2] != 0:
        top_scorer[3] = round(top_scorer[3]/top_scorer[2], 2)

    if top_scorer[0] == 'Unknown':
        top_scorer = ['N/D','N/D','N/D','N/D']

    return top_scorer
```

6.6: top_scorer()

6.7: classify_teams_element()



6.8: top_n_classified_teams_list()

Descripción

Este requerimiento pretende encontrar los mejores n equipos de un torneo específico en un año específico. Para esto se recorren dos estructuras de datos, un mapa de resultados que utiliza los índices de torneo, donde cada torneo es a su vez un mapa organizado por equipos. Cada equipo de este mapa es una lista con la información de los resultados de dicho torneo. La otra estructura de datos es un mapa de goleadores que utiliza las fechas como índice, donde cada fecha es una lista de todos los goleadores que anotaron en esa fecha. Al final, se retorna una lista que incluye los n mejores equipos del torneo en el año especificado.

Entrada	Estructuras de datos del modelo, nombre del torneo, cantidad de equipos a evaluar, y el año.
Salidas	Una lista con la información de los n mejores equipos del torneo en el año especificado.
Implementado (Sí/No)	Sí. Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
6.1	Obtener estructuras de datos y convertir el año a float	O(1)
6.1	Invocar 6.2	O(1)
6.2	Determinar si existe el torneo e inicalización de contadores	O(1)
6.2	Recorrer todos los equipos del torneo	O(e)

6.2	Recorrer todas las partidas del equipo	O(p)
6.2	Invocar 6.3	O(1)
6.3	Modificar contadores	O(1)
6.2	Modificar contadores	O(1)
6.2	Invocar 6.4	O(1)
6.4	Modificar contadores y determinar si existe la fecha	O(1)
6.4	Recorrer todos los goleadores	O(g)
6.4	Modificar contadores	O(1)
6.2	Invocar 6.5	O(1)
6.5	Recorrer parcialmente los goleadores	O(g)
6.2	Invocar 6.6	O(1)
6.6	Crear el elemento	O(1)
6.1	Ordenar	O(k)
6.1	Obtener los n primeros	O(n)
Total		O(epg^2 + k + n)

En comparación a la complejidad del reto 1, O(), es claro que la implementación de mapas reduce significativamente la complejidad del requerimiento, debido a que los recorridos son mucho más pequeños.

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el torneo **FIFA World Cup qualification, 11** equipos en el año **2021.**

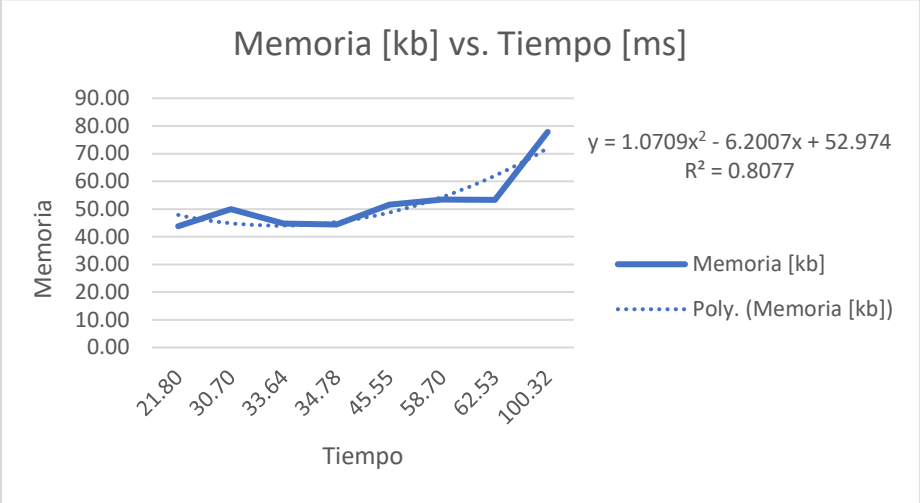
Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

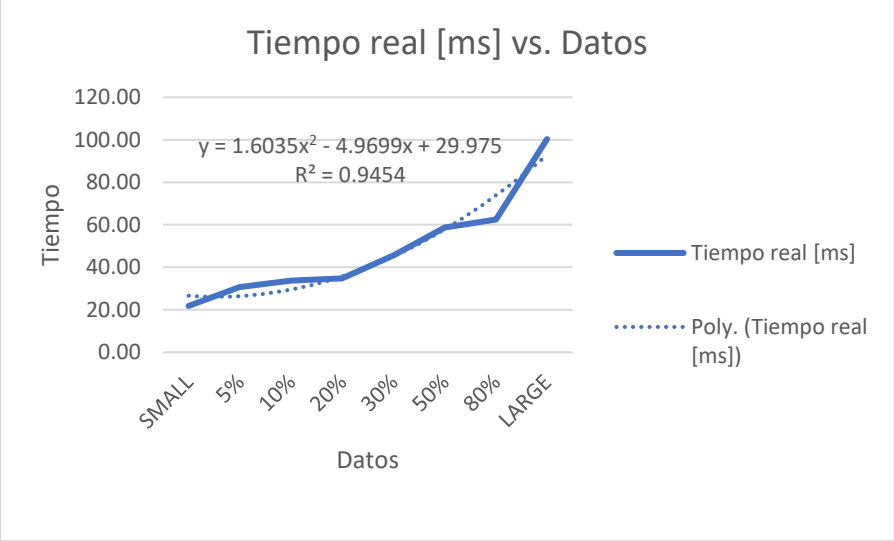
Tablas de datos

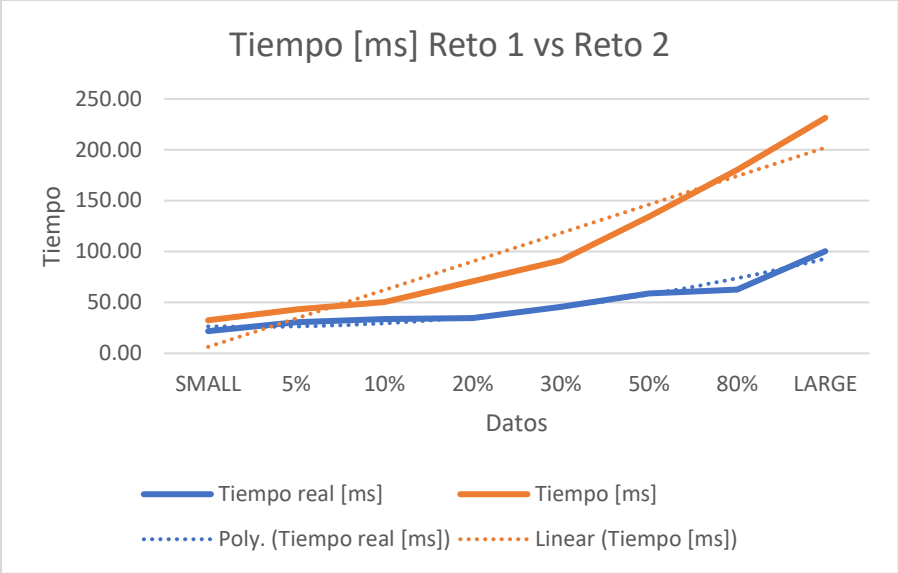
	Reto 2		**Reto 1**
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	21.80	43.79	32.45
5%	30.70	49.90	43.21
10%	33.64	44.71	50.23
20%	34.78	44.45	70.85
30%	45.55	51.60	91.32
50%	58.70	53.44	134.21

80%	62.53	53.29	180.23
LARGE	100.32	77.85	231.34

Graficas







Análisis

En este requerimiento, a pesar de que se puede ver una gran diferencia en cuanto a los tiempos de ejecución entre el reto 1 y el reto 2, siendo el reto 2 más rápido que el anterior, el comportamiento sigue siendo relativamente similar. A pesar de que el crecimiento del reto 1 es más pronunciado, se puede ver indudablemente una tendencia cuadrática (aunque mucho menos prominente), en el reto 2. Por otro lado, el uso de memoria también tiene una tendencia cuadrática.

Requerimiento 7

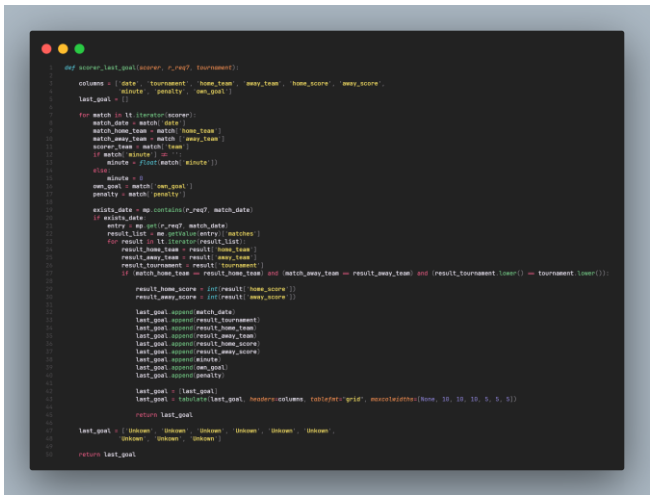
7.1: top_scorers()

```python
def top_scorers_list(req7, req6, tournament):
    scorers = lt.newList('ARRAY_LIST')
    # sacar las estructuras de datos pertinentes
    r_req7 = lt.getElement(req7, 1) # mapa de fechas
    g_req7 = lt.getElement(req7, 2) # mapa de goleadores
    tournaments = lt.getElement(req6, 1) # mapa de torneos
    # sacar la lista de torneos
    tournament_keys = mp.keySet(tournaments)
    # inicializar contadores
    count = {'total_tournaments':lt.size(tournament_keys),
             'total_matches':0,
             'total_scorers': [],
             'total_scorers_in_range':0,
             'total_goals_in_range':0,
             'total_penalty_points':0,
             'total_own_goals':0,}
    # sacar todas las llaves de los goleadores
    scorers_keys = mp.keySet(g_req7)
    # recorrer el mapa de goleadores
    for scorer in lt.iterator(scorers_keys):
        elem = []
        scorer_values = {'total_points':0,
                         'total_goals':0,
                         'penalty_points':0,
                         'own_goals':0,
                         'matches':[],
                         'scorer_minutes':0,
                         'scorer_tournaments':[],
                         'scored_in_wins':0,
                         'scored_in_losses':0,
                         'scored_in_draws':0}
        # obtener la lista de partidos del jugador
        entry = mp.get(g_req7, scorer)
        scorer_matches_list = me.getValue(entry)['matches']
        # recorrer la lista
        for match in lt.iterator(scorer_matches_list):
            # obtener valores del goleador y modificar contador
            scorers_match_value(match, scorer_values, tournament, r_req7, count)
            # obtener el ultimo gol
            last_goal = scorer_last_goal(scorer_matches_list, r_req7, tournament)
            # construir elemento del jugador
            top_scorers_element(elem, scorer, scorer_values, last_goal)
        # agregar elemento del jugador a la lista respuesta
        lt.addFirst(scorers, elem)
    # retornar contador y lista respuesta
    return scorers, count
```

7.2: top_scorers_list()

```python
def scorers_match_value(match, scorer_values, tournament, r_req7, count):
    # sacar los datos pertinentes del match
    match_date = match['date']
    match_home_team = match['home_team']
    match_away_team = match ['away_team']
    scorer_team = match['team']
    minute = match['minute']
    if minute != '':
        minute = float(minute)
    else:
        minute = 0
    own_goal = str(match['own_goal']).lower()
    penalty = str(match['penalty']).lower()
    # determinar si la fecha está en resultados
    exists_date = mp.contains(r_req7, match_date)
    if exists_date:
        # obtener la lista de la fecha
        entry = mp.get(r_req7, match_date)
        date_matches_list = me.getValue(entry)['matches']
        # recorrer la fecha en busca de el resultado de
        for result in lt.iterator(date_matches_list):
            result_home_team = result['home_team']
            result_away_team = result['away_team']
            result_tournament = result['tournament']
            if (match_home_team == result_home_team) and (match_away_team == result_away_team):
                # modificar contadores relacionados al torneo
                match_tourn = result['tournament']
                #if match_tourn not in count['total_tournaments']:
                    #count['total_tournaments'].append(match_tourn)
                if match_tourn not in scorer_values['scorer_tournaments']:
                    scorer_values['scorer_tournaments'].append(match_tourn)
                # verificar si la partida es del torneo en cuestion
                if match_tourn.lower() == tournament.lower():
                    # modificar contador de jugadores del torneo
                    match_scorer = match['scorer']
                    if match_scorer not in count['total_scorers']:
                        count['total_scorers'].append(match_scorer)
                    # modificar sentinela de torneo
                    is_tournament = True
                else:
                    is_tournament = False

                if is_tournament and match_date not in scorer_values['matches']:
                    scorer_values['matches'].append(match_date)
                    count['total_matches'] += 1

                result_home_score = int(result['home_score'])
                result_away_score = int(result['away_score'])
                if is_tournament and scorer_team == result_home_team:
                    if result_home_score > result_away_score:
                        scorer_values['scored_in_wins'] += 1
                    elif result_home_score < result_away_score:
                        scorer_values['scored_in_losses'] += 1
                    else:
                        scorer_values['scored_in_draws'] += 1
                    scorer_values['total_goals'] += 1
                elif is_tournament and scorer_team == result_away_team:
                    if result_home_score > result_away_score:
                        scorer_values['scored_in_losses'] += 1
                    elif result_home_score < result_away_score:
                        scorer_values['scored_in_wins'] += 1
                    else:
                        scorer_values['scored_in_draws'] += 1
                    scorer_values['total_goals'] += 1

    scorer_values['scorer_minutes'] += minute

    if is_tournament:
        count['total_goals_in_range'] += 1
        if own_goal == 'true':
            scorer_values['own_goals'] += 1
            count['total_own_goals'] += 1
            scorer_values['total_points'] -= 1
        if penalty == 'true':
            scorer_values['penalty_points'] += 1
            count['total_penalty_points'] += 1
            scorer_values['total_points'] += 1
```

7.3: scorers_match_value()



7.4: scorer_last_goal()

```
def top_scorers_element(elem, name, scorer_values, last_goal):
    elem.append(name)

    total_points = scorer_values['total_points'] + scorer_values['total_goals']
    elem.append(total_points)

    elem.append(scorer_values['total_goals'])
    elem.append(scorer_values['penalty_points'])
    elem.append(scorer_values['own_goals'])

    if len(scorer_values['matches']) != 0:
        avg_time = scorer_values['scorer_minutes'] / len(scorer_values['matches'])
    else:
        avg_time = 'Unkown'
    elem.append(avg_time)

    elem.append(len(scorer_values['scorer_tournaments']))
    elem.append(scorer_values['scored_in_wins'])
    elem.append(scorer_values['scored_in_losses'])
    elem.append(scorer_values['scored_in_draws'])
    elem.append(last_goal)
```

7.5: top_scorers_element()

```
def top_n_scorers_list(scorers, n):
    top_n_scorers = lt.newList('ARRAY_LIST')
    for scorer in lt.iterator(scorers):
        scorer_total_points = scorer[1]
        if scorer_total_points == n:
            lt.addLast(top_n_scorers, scorer)
    return top_n_scorers
```

7.6: top_n_scorers_list()

Descripción

Esta función pretende encontrar todos los jugadores con un puntaje de n en el torneo especificado. Para esto, se utilizan 2 estructuras de datos. En primer lugar, se utiliza un mapa de resultados el cual se organiza con base en la fecha, donde cada fecha tiene una lista de partidos jugados. Segundo, se utiliza un mapa de goleadores organizado por goleador, donde cada goleador tiene una lista de sus goles. Al

final se retorna una lista con todos los goleadores que cumplen el requisito de tener un puntaje total de n.

Entrada	Estructuras de datos del modelo, torneo en cuestión y número de puntos.
Salidas	Una lista con la información de todos los jugadores con el puntaje especificado.
Implementado (Sí/No)	Sí. Adrian Velasquez 202222737

Análisis de complejidad

Función	Paso	Complejidad
7.1	Obtener estructuras de datos y convertir el año a float	O(1)
7.1	Invocar 7.2	O(1)
7.2	Obtener estructuras de datos, inicializar contadores y demás	O(1)
7.2	Recorrer mapa de goleadores	O(g)
7.2	Recorrer lista de partidos	O(p)
7.2	Invocar 7.3	O(1)
7.3	Sacar valores y determinar si existe la fecha en el mapa de resultados	O(1)
7.3	Recorrer la lista de la fecha	O(f)
7.3	Cambiar contadores	O(1)
7.2	Invocar 7.4	O(1)
7.4	Recorrer las partidas de scorer	O(p)
7.4	Recorrer las partidas de la fecha	O(f)
7.2	Invocar 7.5	O(1)
7.5	Crear el elemento de la lista respuesta	O(1)
7.1	Ordenar	O(k)
7.1	Buscar los goleadores con n goles	O(k)
Total		O(gpf + pf + 2k)

En comparación a la complejidad del reto 1, O(), es claro que la implementación de mapas reduce significativamente la complejidad del requerimiento, debido a que los recorridos son mucho más pequeños.

Pruebas Realizadas

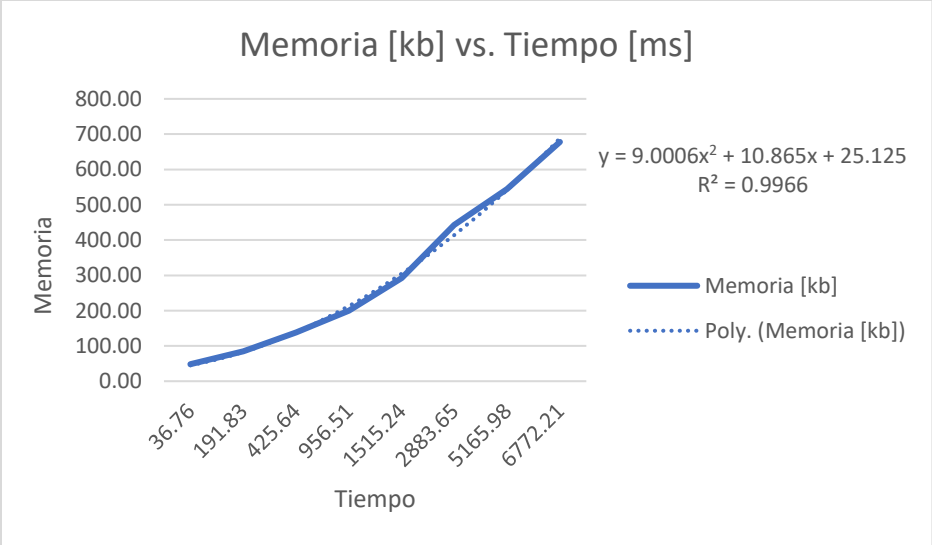
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el torneo **UEFA Euro qualification,** con un mínimo de **2** puntos**.**

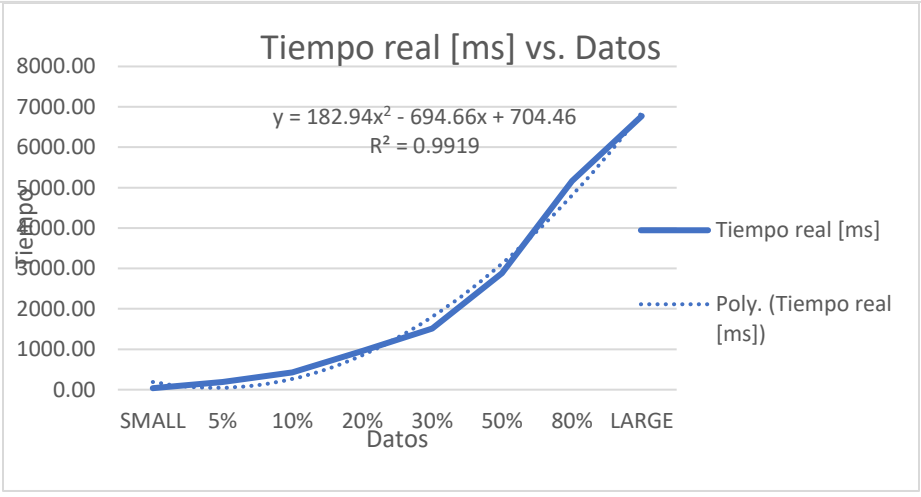
Procesadores	**11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

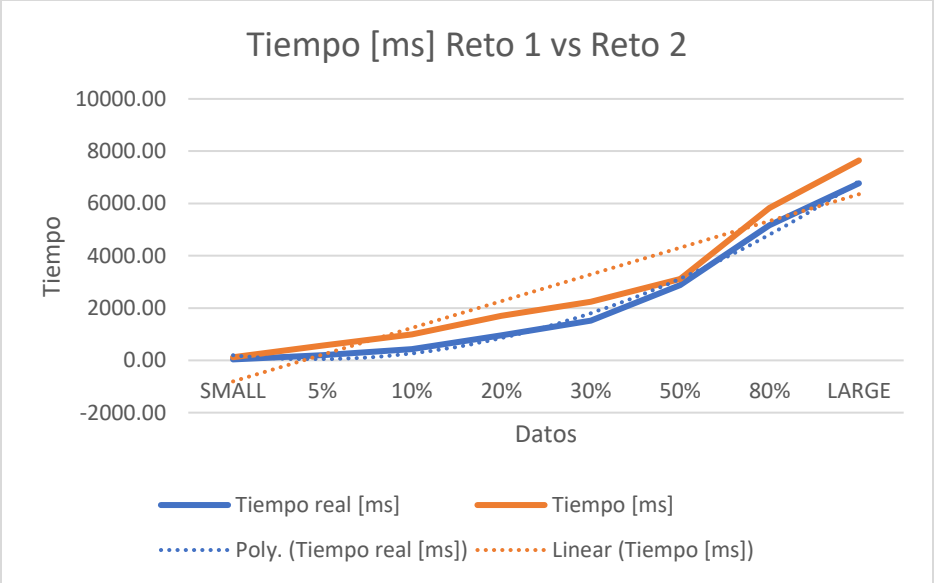
Tablas de datos

	Reto 2		Reto 1
Entrada	**Tiempo real [ms]**	**Memoria [kb]**	**Tiempo [ms]**
SMALL	36.76	48.20	119.99
5%	191.83	84.60	559.23
10%	425.64	137.99	993.82
20%	956.51	199.72	1,696.76
30%	1515.24	291.99	2,245.44
50%	2883.65	442.88	3,108.28
80%	5165.98	544.82	5,823.89
LARGE	6772.21	678.07	7,646.48

Graficas







Análisis

En este requerimiento, el comportamiento en cuanto al tiempo tanto del reto 1 como del reto 2 es muy similar. Esto se debe a que, a pesar de que se implementan los mapas en el reto 2, se sigue teniendo que hacer un recorrido extenso de las estructuras de datos. Debido a esto, ambas curvas muestran un comportamiento cercano al cuadrático. El uso de memoria, por otro lado, también tiene una tendencia cuadrática poco pronunciada.

Requerimiento 8

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

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

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(...)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(...)***

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)**

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

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