ANÁLISIS DEL RETO 2

Andres Chaparro Diaz, 202111146, a.chaparrod

Edward Camilo Sánchez Nova, 202113020, e.sanchezn

Juan Esteban Rojas, 202124797, je.rojasc1

Requerimiento 1

Descripción

```
def req_1(data_structs,num,equipo,condicion):
    """
    Función que soluciona el requerimiento 1
    """

    mapa = data_structs["map_TeamResults"]
    total_equipos = mp.size(mapa)
    entry = mp.get(mapa, equipo)
    lista_equipo = me.getValue(entry)["datos"]
    neutral = lt.newList('ARRAY_LIST')
    home = lt.newList('ARRAY_LIST')
    away = lt.newList('ARRAY_LIST')

    for match in lt.iterator(lista_equipo):
        if match['neutral'] == 'True':
            lt.addLast(neutral, match)
        if match['neutral'] == 'False' and match['home_team'] == equipo:
            lt.addLast(home, match)
        if match['neutral'] == 'False' and match['away_team'] == equipo:
            lt.addLast(away, match)

    if condicion.lower() == 'local' or condicion.lower() == 'home':
        x = sa.sort(home, sort_criteria_req1)
        if lt.size(x) > num:
            SubListN = lt.subList(x, 1,num)
        else:
            SubListN = x
    if condicion.lower() == 'indiferente' or condicion.lower() == 'neutral':
        x = sa.sort(neutral, sort_criteria_req1)
        if lt.size(x) > num:
            SubListN = lt.subList(x, 1,num)
        else:
            SubListN = x

    if condicion.lower() == 'visitante' or condicion.lower() == 'away':
        x = sa.sort(away, sort_criteria_req1)
        if lt.size(x) > num:
            SubListN = lt.subList(x, 1,num)
        else:
            SubListN = x

    total_partidos = lt.size(lista_equipo)
    total_condicion = lt.size(SubListN)

    if lt.size(SubListN) <= 6:
        return SubListN, total_equipos, total_partidos, total_condicion
    else:
        return FirstandALst(SubListN), total_equipos, total_partidos, total_condicion
```

Para el desarrollo del requerimiento 1, se utilizó el mapa ‘map_teamResults’ del catálogo. Para empezar, se itero sobre la lista que contiene todos los partidos de un equipo, y se agregaron a una lista dependiendo su condición.

Luego de tener todos los partidos clasificados en tres listas, se ordena la lista que corresponde a la condición elegida por el usuario y se encuentra una sublista dependiendo el número de elementos solicitados por parámetro.

Entrada	data_structs, num, equipo, condicion
Salidas	SubListN, total_equipos, total_partidos, total_condicion
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

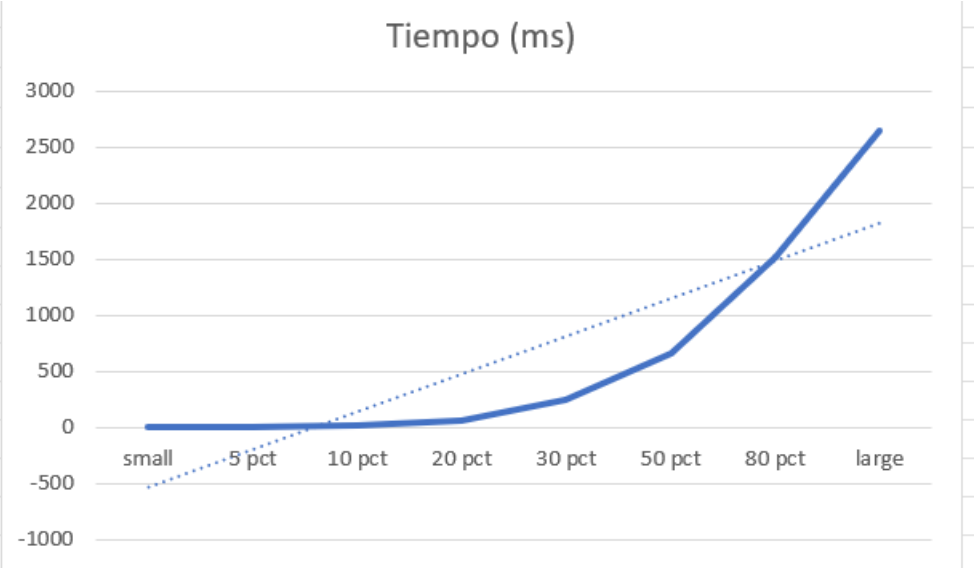
Pasos	**Complejidad**
Iteración por Lista_equipo	O(n)
lt.addLast	O (1)
Shell Sort	O (n log n)
SubList	O(n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesadores	**Intel(R) Core(TM) i7-10750H CPU**
Memoria RAM	32 GB

Entrada	**Tiempo (ms)**
small	0.3070999999945343
5 pct	0.7292000000015832
10 pct	0.9970000000030268
20 pct	2.0248000000137836
30 pct	2.9363000000012107
50 pct	4.768999999971129
80 pct	7.377900000021327
large	9.936099999933504

Graficas



Análisis

En el requerimiento 1 se encontró una complejidad O (n log n) ya que luego de la iteración que se usa para clasificar los partidos, se hace un shell sort de complejidad O (n log n). Además, esto se puede evidenciar en la gráfica obtenida después de las pruebas realizadas.

Aunque esta función tiene una complejidad similar a la realizada en el reto 1, esta es más eficiente debido a que el valor de n es menor. Esto debido a que no se iteran todos los resultados si no que solo los del equipo entregado como parámetro.

Requerimiento 2

Descripción

```python
def req_2(data_structs,num,jugador):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    datos = data_structs["scorer"]
    total_anotaciones = mp.size(datos)
    lista_jugador = me.getValue(mp.get(datos,jugador))["datos"]
    lista_jugador_ordenada = ordenar_req_2(lista_jugador)
    goles_goleador = lt.size(lista_jugador_ordenada[0])
    penales = lista_jugador_ordenada[1]
    goles_interes = lt.newList("ARRAY_LIST")
    para_printear = lt.newList("ARRAY_LIST")
    if num <= goles_goleador:
        for i in range(0,num):
            dato = lt.getElement(lista_jugador_ordenada[0],i)
            lt.addLast(goles_interes,dato)
    else:
        for i in range(0,goles_goleador):
            dato = lt.getElement(lista_jugador_ordenada[0],i)
            lt.addLast(goles_interes,dato)

    if lt.size(goles_interes)>=6:
        num_tabla = "Goal scorers results hass more than 6 records..."
        for i in range(1, 4):
            lt.addLast(para_printear,lt.getElement(goles_interes,i))
        for i in range(lt.size(goles_interes)-2, lt.size(goles_interes)+1):
            lt.addLast(para_printear,lt.getElement(goles_interes,i))
    else:
        num_tabla = "Goal scorers results hass less than 6 records..."
        for i in range(1, lt.size(goles_interes)+1):
            lt.addLast(para_printear,lt.getElement(goles_interes,i))
    return total_anotaciones , goles_goleador , penales , para_printear, num_tabla
```

El requerimiento 2 extrae datos relevantes del data_structs, en este caso un mapa del tipo:

{nombre Jugador: [gol1] , [gol2] … [gol n]}

Luego, calcula el total de anotaciones y la cantidad máxima de goles anotados por un jugador. Posteriormente, organiza una lista de goles de un jugador específico solicitado por el usuario. Por último, dependiendo la cantidad de elementos que contiene la respuesta, se retornan los primeros y últimos 3 o la lista completa.

Entrada	data_structs, num, jugador
Salidas	total_anotaciones , goles_goleador , penales, para_printear, num_tabla
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

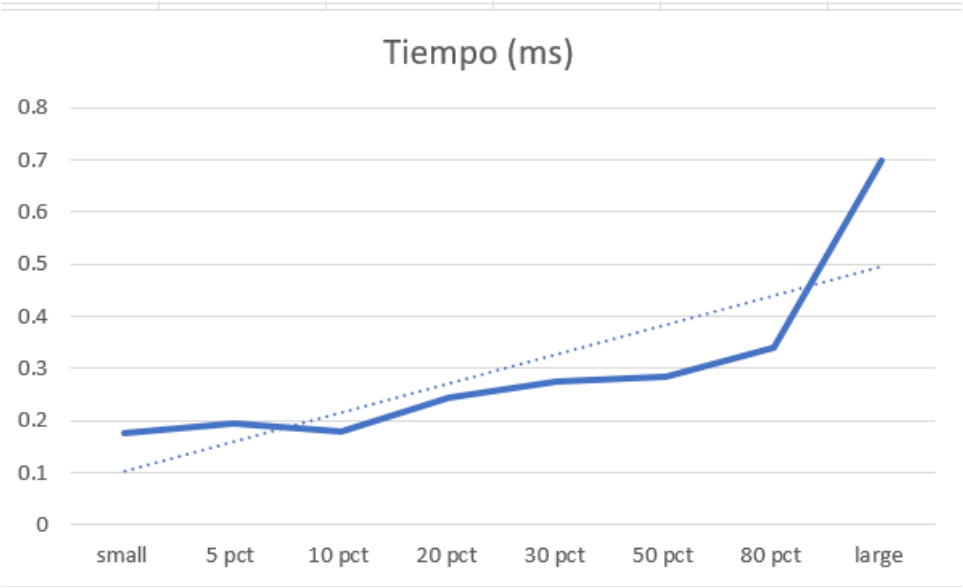
Pasos	**Complejidad**
Ordenar_req_2	O(n)
lt.addLast	O (1)
TOTAL	***O (n)***

Pruebas Realizadas

Procesadores	**Intel(R) Core(TM) i7-10750H CPU**
Memoria RAM	32 GB

Entrada	**Tiempo (ms)**
small	0.17459999999846332
5 pct	0.19369999998889398
10 pct	0.18039999999746215
20 pct	0.24470000001019798
30 pct	0.275800000061281
50 pct	0.2828999999910593
80 pct	0.33929999999236315
large	0.6976999999023974

Graficas



Análisis

En términos de complejidad este requerimiento es O(n) debido a la función ordenar_req_2 que itera sobre la lista con las anotaciones de un jugador. Además, después de hacer las pruebas y graficar los resultados, encontramos que es una función muy eficiente debido a que las iteraciones que requiere son muy cortas debido a la estructura de datos utilizada (mapa).

Requerimiento 3

Descripción

```
def req_3(control, Equipo, Inicio, Final):
    """
    Función que soluciona el requerimiento 3
    """
    local = 0
    visitante = 0
    respuesta = lt.newList("ARRAY_LIST")
    mapa = control['map_TeamResults']
    entry = mp.get(mapa, Equipo)
    SubLista1 = me.getValue(entry)
    SubLista1 = SubLista1["datos"]
    listaOrd1 = merg.sort(SubLista1, sort_criteria_eng)
    entry = mp.get(control["map_TeamGoalScorers"], Equipo)
    SubLista2 = me.getValue(entry)['datos']
    listaOrd2 = merg.sort(SubLista2, sort_criteria_eng)
    for game in lt.iterator(listaOrd1):
        fecha = game["date"]
        if fecha >= Inicio and fecha <= Final:
            Pos  = busqueda_binaria_req3(listaOrd2, fecha)
            diccionario_i = game
            if game["home_team"] == Equipo:
                local += 1
            elif game["away_team"] == Equipo:
                visitante += 1
            if Pos > 0:
                diccionario_gsc = lt.getElement(listaOrd2, Pos)
                diccionario_i["AutoGol"] =  diccionario_gsc["penalty"]
                diccionario_i["Penalty"] =  diccionario_gsc["own_goal"]
                lt.addLast(respuesta, diccionario_i)
            elif Pos == -1:
                diccionario_i["AutoGol"] = "Desconocido"
                diccionario_i["Penalty"] =  "Desconocido"
                lt.addLast(respuesta, diccionario_i)
    return respuesta, local, visitante
```

Para el desarrollo del requerimiento 3, se utilizaron 2 mapas del catálogo. Primero, el mapa ‘map_TeamResults’ que contiene llaves valor de tipo:

{nombre Equipo: [partido1] , [partido 2] … [partido n]}

Y el mapa “scorer” que contiene llaves valor de tipo:

{nombre Jugador: [gol1] , [gol2] … [gol n]}

Para el desarrollo del requerimiento se itero sobre la lista que contiene los partidos de un equipo y se adquirió la información necesaria. Para obtener la información faltante de los partidos y se hizo una búsqueda binaria para encontrar el partido que coincide con el gol anotado. Luego de esto se agrega la información a un formato de presentación y lo agrega a la lista de solución que posteriormente se retorna.

Entrada	control, Equipo, Inicio, Final
Salidas	Respuesta, local, visitante
Implementado (Sí/No)	SI – Andres Felipe Chaparro

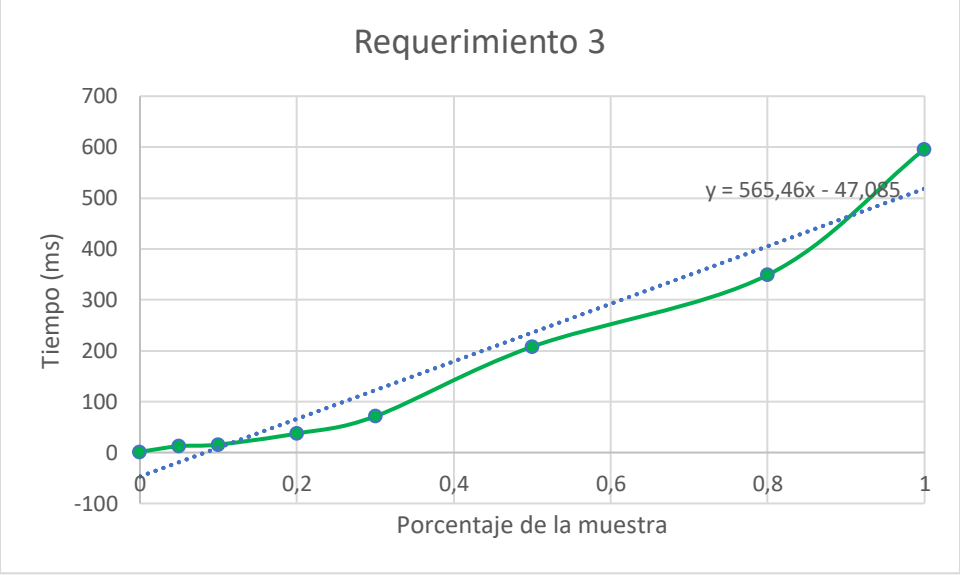
Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Iteración por Equipo	O(n)
Búsqueda binaria por cada partido del equipo	O(logn)
TOTAL	***O(nlogn)***

Pruebas Realizadas

Procesadores	**intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz, 1201**
Memoria RAM	12 GB
Entrada	**Tiempo (ms)**
small	1,5
5 pct	12,54
10 pct	15,65
20 pct	37,4
30 pct	71,23
50 pct	207,96
80 pct	348,84
large	596,2925



Análisis

Teniendo en cuenta las pruebas realizadas y el análisis de complejidad, podemos concluir que el uso de estructuras de datos como mapas mejoro significativamente la eficiencia de la función req_3. Por otro lado, el requerimiento tiene una complejidad n logarítmica ya que tiene que iterar primero el mapa de

resultados por equipos y por cada partido de acá se itera en el mapa de goles. Aunque la complejidad con notación O se mantiene igual, si uno revisa el n va a disminuir puesto que ahora solo recorrerá los partidos por equipo y no el de todos los equipos.

Requerimiento 4

Descripción

```
def req_4(control,torneo,inicio,final):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    torneos = control["map_tournament"]
    llave_valor_torneo = mp.get(torneos,torneo)
    valor_torneo = me.getValue(llave_valor_torneo)
    lista_considerar = ordenar_req_4(valor_torneo,inicio,final)
    pre_printeo = lt.newList("ARRAY_LIST")
    para_printear = lt.newList("ARRAY_LIST")

    total_torneos = mp.size(torneos)
    total_partidos_torneo = lt.size(lista_considerar)
    countries = lt.newList("ARRAY_LIST")
    cities = lt.newList("ARRAY_LIST")
    shoot = lt.newList("ARRAY_LIST")
    for i in range(1, lt.size(lista_considerar)+1):
        dato_primitivo = lt.getElement(lista_considerar,i)
        dato = {
                "date":dato_primitivo["date"],
                "tournament":dato_primitivo["tournament"],
                "country":dato_primitivo["country"],
                "city":dato_primitivo["city"],
                "home_team":dato_primitivo["home_team"],
                "away_team":dato_primitivo["away_team"],
                "home_score":dato_primitivo["home_score"],
                "away_score":dato_primitivo["away_score"],
                "winner":None,
            }
        if not lt.isPresent(cities,dato_primitivo["city"]):
            lt.addLast(cities,dato_primitivo["city"])

        if not lt.isPresent(countries,dato_primitivo["country"]):
            lt.addLast(countries,dato_primitivo["country"])

        aparecio = buscar_shoot(control,dato_primitivo["date"],dato_primitivo["home_team"])
        if aparecio[0] == -1:
            dato["winner"] = "Unavailable"
        else:
            dato["winner"] = aparecio[1]
            if not lt.isPresent(shoot, dato["winner"]+dato["date"]) and aparecio[0] !=-1:
                lt.addLast(shoot,dato["winner"]+dato["date"])
        lt.addLast(pre_printeo,dato)
    if lt.size(pre_printeo)>=6:
        num_tabla = "The tournament results hass more than 6 records..."
        for i in range(1, 4):
            lt.addLast(para_printear,lt.getElement(pre_printeo,i))
        for i in range(lt.size(lista_considerar)-2, lt.size(lista_considerar)+1):
            lt.addLast(para_printear,lt.getElement(pre_printeo,i))
    else:
        num_tabla = "The tournament results hass less than 6 records..."
        for i in range(1, lt.size(pre_printeo)+1):
            lt.addLast(para_printear,lt.getElement(pre_printeo,i))
    return  para_printear , total_torneos , total_partidos_torneo , lt.size(countries) , lt.size(cities) , lt.size(shoot),num_tabla
```

Para el desarrollo del requerimiento 3, se utilizaron un mapa y una lista enlazada del catálogo. Primero, el mapa ‘map_torunament’ que contiene llaves valor de tipo:

{nombre Tournament: [partido1] , [partido 2] … [partido n]}

Y la lista de shootouts que contiene los penaltis

[penalty] , [penalty2] … [penalty n]

Para el desarrollo del requerimiento se itero sobre el mapa que contiene los partidos de un torneo y se adquirió la información necesaria. Para oftener la información faltante de los partidos y se hizo una búsqueda binaria para

encontrar el penalty asociado. Luego de esto se agrega la información a un formato de presentación y lo agrega a la lista de solución que posteriormente se retorna.

Entrada	control, Torneo, Inicio, Final
Salidas	para_printear , total_torneos , total_partidos_torneo , lt.size(countries) , lt.size(cities) , lt.size(shoot),num_tabla
Implementado (Sí/No)	SI – Esteban

Análisis de complejidad

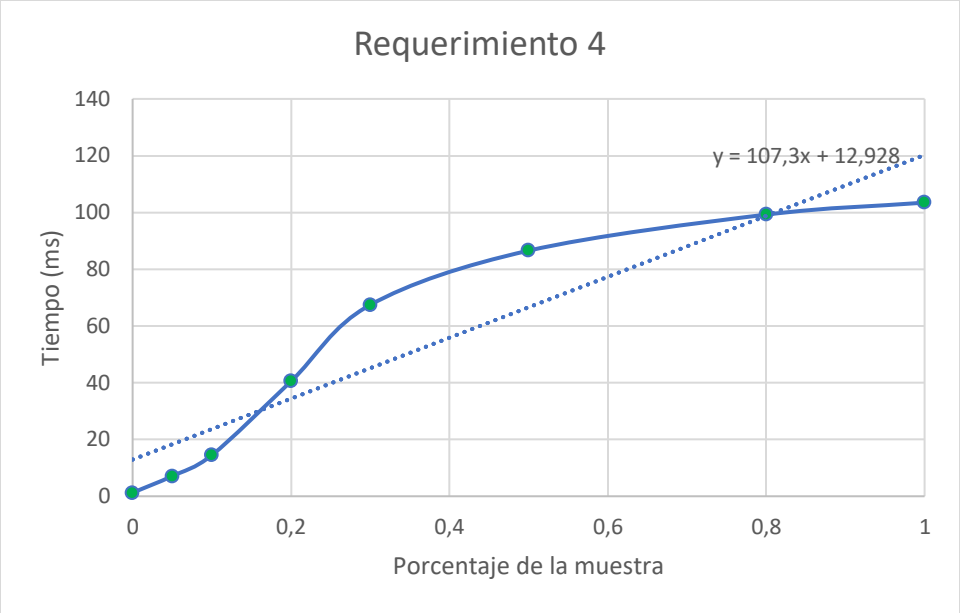
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Itera por torneo	O(n)
Búsqueda binaria en la lista penalties	O(logn)
TOTAL	***O(nlogn)***

Pruebas Realizadas

Procesadores	**M2**
Memoria RAM	8 GB

Entrada	**Tiempo (ms)**
small	1.25
5 pct	7.11
10 pct	14.47
20 pct	40.5
30 pct	67.42
50 pct	86.54
80 pct	99.22
large	103.45



Análisis

Teniendo en cuenta las pruebas realizadas y el análisis de complejidad, podemos concluir que el uso de estructuras de datos como mapas mejoro significativamente la eficiencia de la función req_4. Por otro lado, el requerimiento tiene una complejidad n log n ya que solo tiene que hacer una gran iteración en los partidos de cada mapa y después se hace una búsqueda binaria para los penaltis.

Requerimiento 5

Descripción

```python
def req_5(control, player, Inicio, Final):
    """
    Función que soluciona el requerimiento 5
    """
    mapa_results = control['map_TeamResults']
    mapa_scorers = control['scorer']

    entry = mp.get(mapa_scorers, player)
    player_list = me.getValue(entry)['datos']

    total_scores = lt.size(player_list)
    total_players = mp.size(mapa_scorers)

    solution = lt.newList('ARRAY_LIST')
    tournaments = lt.newList('ARRAY_LIST')
    shootouts = 0
    own_goals = 0

    formato = "%Y-%m-%d"
    for score in lt.iterator(player_list):
        if datetime.strptime(score["date"], formato) > datetime.strptime(Inicio, formato) and datetime.strptime(score["date"], formato) < datetime.strptime(Final, formato):
            team = score['team']
            date = score['date']
            entry = mp.get(mapa_results, team)
            team_list = me.getValue(entry)['datos']
            sort_team_list = sa.sort(team_list, sort_criteria_eng)
            dict_match = busqueda_binaria_req5(sort_team_list, date )
            score_format = {}
            score_format['date'] = date
            score_format['minute'] = score['minute']
            score_format['home_team'] = score['home_team']
            score_format['away_team'] = score['away_team']
            score_format['team'] = score['team']
            score_format['home_score'] = dict_match['home_score']
            score_format['away_score'] = dict_match['away_score']
            score_format['tournament'] = dict_match['tournament']
            score_format['penalty'] = score['penalty']
            score_format['own_goal'] = score['own_goal']

            lt.addLast(solution, score_format)
            if not(lt.isPresent(tournaments, score_format['tournament'])):
                lt.addLast(tournaments,score_format['tournament'] )

            if score['penalty'] == 'True':
                shootouts += 1
            if score['own_goal'] == 'True':
                own_goals += 1

    total_tournaments = lt.size(tournaments)
    sorted_solution = sa.sort(solution, sort_criteria_req5)
    if lt.size(solution) <= 6:
        return sorted_solution, total_players, total_scores, total_tournaments, shootouts, own_goals
    else:
        final_solution = FirstandALst(sorted_solution)
        return final_solution, total_players, total_scores, total_tournaments, shootouts, own_goals
```

Para el desarrollo del requerimiento 5, se utilizaron 2 mapas del catálogo. Primero, el mapa ‘map_TeamResults’ que contiene llaves valor de tipo:

{nombre Equipo: [partido1] , [partido 2] … [partido n]}

Y el mapa “scorer” que contiene llaves valor de tipo:

{nombre Jugador: [gol1] , [gol2] … [gol n]}

Para el desarrollo del requerimiento se itero sobre la lista que contiene los goles del jugador y se adquirió la información necesaria. Para obtener la información de los partidos y se hizo una búsqueda binaria para encontrar el partido que coincide con el gol anotado. Luego de esto se agrega la información a un formato de presentación y lo agrega a la lista de solución que posteriormente se retorna.

Entrada	control, player, Inicio, Final

Salidas	final_solution, total_players, total_scores, total_tournaments, shootouts, own_goals
Implementado (Sí/No)	SI – Camilo Sánchez Novoa

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

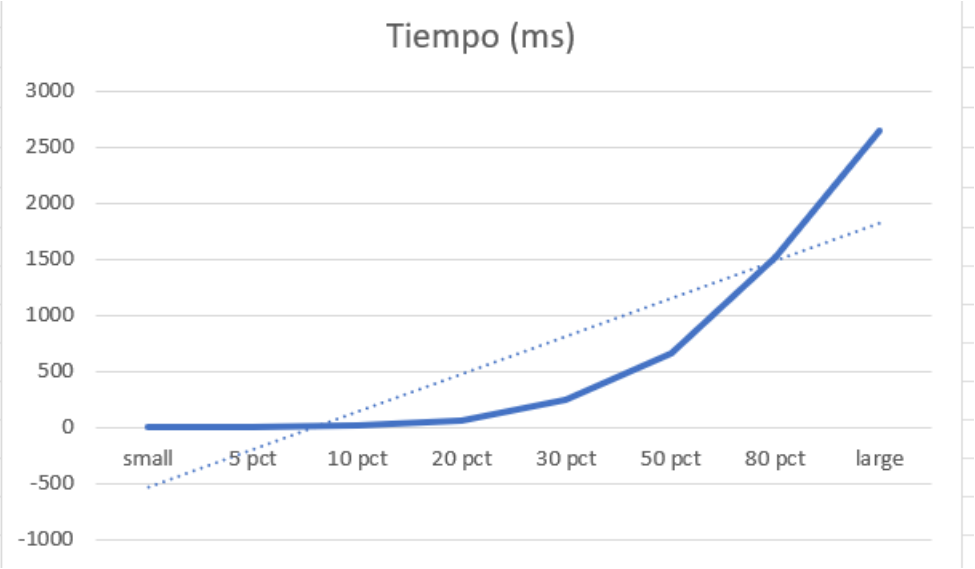
Pasos	**Complejidad**
Iteración por Player_list	O(n)
lt.addLast	O(1)
(isPresent)	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

Procesadores	**Intel(R) Core(TM) i7-10750H CPU**
Memoria RAM	32 GB

Entrada	**Tiempo (ms)**
small	4.419500000003609
5 pct	5.092900000003283
10 pct	10.725000000034925
20 pct	62.64199999999255
30 pct	241.8712999999988
50 pct	656.1456000000035
80 pct	1524.0718999999808
large	2648.34009999997

Graficas



Análisis

Teniendo en cuenta las pruebas realizadas y el análisis de complejidad, podemos concluir que el uso de estructuras de datos como mapas mejoro significativamente la eficiencia de la función req_5. Por otro lado, el requerimiento tiene una complejidad lineal ya que solo tiene que hacer una gran iteración en los goles de un jugador, que es menos compleja que la misma función del req_5 del reto 1 ya que en el anterior reto hacia una iteración a todos los goles del archivo goalscorers.

Requerimiento 6

Descripción

- Para empezar, en el requerimiento 6 se guardan los mapas que se utilizaran para resolver el requerimiento, y se crean algunas listas y mapas como estructuras internas para resolver el requerimiento. También se inicia una iteración sobre los partidos de un torneo y se recolecta alguna información de ellos.

```
def req_6(control, torneo, año, n):
    """
    Función que soluciona el requerimiento 6
    """
    map_torneos = control['map_tournament']
    map_dates = control['map_ScoresDates']
    map_years = control['map_ResultsDates']
    map_teams_goals= mp.newMap(100,
                               maptype="CHAINING",
                               loadfactor=4)

    entry = mp.get(map_torneos, torneo)
    matches_list = me.getValue(entry)['datos']
    inicio = str(año) + '-01-01'
    final = str(año) + '-12-31'
    formato = "%Y-%m-%d"
    teams = lt.newList('ARRAY_LIST')
    ranking = lt.newList('ARRAY_LIST')
    torneos = lt.newList('ARRAY_LIST')
    countries = lt.newList('ARRAY_LIST')
    citiesUnique = lt.newList('ARRAY_LIST')
    citiesMatches = lt.newList('ARRAY_LIST')
    totalmatches = 0

    entryyear = mp.get(map_years, año)
    yearlst = me.getValue(entryyear)['datos']
    # Iteracion para encontrar el total de torneos
    for partido in lt.iterator(yearlst):
        if not (lt.isPresent( torneos, partido['tournament'])):
            lt.addLast(torneos, partido['tournament'])

    #iteracion sobre los partidos de un torneo
    for match in lt.iterator(matches_list):
        home_added = False
        if datetime.strptime(match["date"], formato) > datetime.strptime(inicio, formato) and datetime.strptime(ma
            totalmatches += 1
            lt.addLast(citiesMatches, match['city'])
            #Cuantas ciudades
            if not (lt.isPresent(citiesUnique, match['city'])):
                lt.addLast(citiesUnique, match['city'])
            #Cuantos paises
            if not(lt.isPresent(countries, match['country'])):
                lt.addLast(countries, match['country'])

            entry1 = mp.get(map_dates, match['date'])
            if entry1 is not None:
                date_list = me.getValue(entry1)
                date_list1 = date_list['datos']
                #Encontrar los datos de los goels del partido
                golinmatch = encontrar_diccionario(date_list1, match['home_team'], match['away_team'] )
                penalty_goals_home, penalty_goals_away  = penalty_points(golinmatch)
                own_goals_home, own_goals_away = own_points(golinmatch)
                #agrego goles a mapa auxiliar de goles de un equipo en el torneo
                addgoalsHome(golinmatch, map_teams_goals)
```

- Luego, Se verifica si los países que juegan el partido ya están en le ranking, si ya están se suma la información del partido a la información total

```
#Suma informacion al ranking
if lt.isPresent(teams, match['home_team']) and lt.isPresent(teams, match['away_team']):
    home_team_dict = encontrar_diccionario_home(ranking, match['home_team'])
    away_team_dict = encontrar_diccionario_home(ranking, match['away_team'])
    #agrega los puntos y vicorias derrotas y empates
    if int(match['home_score']) > int(match['away_score']):
        home_team_dict['total_points'] += 3
        home_team_dict['wins'] += 1
        away_team_dict['losses'] += 1
    elif int(match['home_score']) == int(match['away_score']):
        home_team_dict['total_points'] += 1
        away_team_dict['total_points'] += 1
        home_team_dict['draws'] += 1
        away_team_dict['draws'] += 1
    else:
        away_team_dict['total_points'] += 3
        away_team_dict['wins'] +=1
        home_team_dict['losses'] += 1

    #agrega la diferencia de gol
    home_team_dict['goal_difference'] += int(match['home_score']) - int(match['away_score'])
    away_team_dict['goal_difference'] += int(match['away_score']) - int(match['home_score'])

    #agrega los penalty points
    if entry1 is not None:
        home_team_dict['penalty_points'] += penalty_goals_home
        away_team_dict['penalty_points'] += penalty_goals_away

    #agraga llave matches
    home_team_dict['matches'] += 1
    away_team_dict['matches'] += 1

    # agrega los own goal points
    if entry1 is not None:
        home_team_dict['own_goal_points'] += own_goals_home
        away_team_dict['own_goal_points'] += own_goals_away

    #total de goles
    home_team_dict['goals_for'] += int(match['home_score'])
    away_team_dict['goals_for'] += int(match['away_score'])

    # total goles en contra
    home_team_dict['goals_against'] += int(match['away_score'])
    away_team_dict['goals_against'] += int(match['home_score'])
```

- Mas adelante, si el país local no está en el ranking aún se crea el espacio para el país en el ranking y se añade la información del partido.

```
#Creación de paises en el ranking en caso de no existir
if not lt.isPresent(teams, match['home_team']):
    lt.addLast(teams, match['home_team'])
    away_team_dict = encontrar_diccionario_home(ranking, match['away_team'])
    team_format ={}
    #Agrega el nombre del equpo
    team_format['team'] = match['home_team']

    #agrega los puntos y vicorias derrotas y empates
    if int(match['home_score']) > int(match['away_score']):
        team_format['total_points'] = 3
        team_format['wins'] = 1
        team_format['draws'] = 0
        team_format['losses'] = 0
        if lt.isPresent(teams, match['away_team']):
            away_team_dict['losses'] += 1
    elif int(match['home_score']) == int(match['away_score']):
        team_format['total_points'] = 1
        team_format['wins'] = 0
        team_format['draws'] = 1
        team_format['losses'] = 0
        if lt.isPresent(teams, match['away_team']):
            away_team_dict['draws'] += 1
            away_team_dict['total_points'] += 1
    else:
        team_format['total_points'] = 0
        team_format['wins'] = 0
        team_format['draws'] = 0
        team_format['losses'] = 1
        if lt.isPresent(teams, match['away_team']):
            away_team_dict['wins'] += 1
            away_team_dict['total_points'] += 3

    #agrega la diferencia de gol
    team_format['goal_difference'] = int(match['home_score']) - int(match['away_score'])
    if lt.isPresent(teams, match['away_team']):
        away_team_dict['goal_difference'] += int(match['away_score']) - int(match['home_score'])

    #agrega los penalty points
    if entry1 is None:
        team_format['penalty_points'] = 0
    else:
        team_format['penalty_points'] = penalty_goals_home
        if lt.isPresent(teams, match['away_team']):
            away_team_dict['penalty_points'] += penalty_goals_away

    #agraga llave matches
    team_format['matches'] = 1
    if lt.isPresent(teams, match['away_team']):
        away_team_dict['matches'] += 1

    # agrega los own goal points
    if entry1 is None:
        team_format['own_goal_points'] = 0
    else:
        team_format['own_goal_points'] = own_goals_home
        if lt.isPresent(teams, match['away_team']):
            away_team_dict['own_goal_points'] += own_goals_away

    #total de goles
    team_format['goals_for'] = int(match['home_score'])
    if lt.isPresent(teams, match['away_team']):
        away_team_dict['goals_for'] += int(match['away_score'])

    # total goles en contra
    team_format['goals_against'] = int(match['away_score'])
    if lt.isPresent(teams, match['away_team']):
        away_team_dict['goals_against'] += int(match['home_score'])

    lt.addLast(ranking, team_format)
    home_added = True
```

- Por último, se verifica si el equipo local está en el ranking, y de no estarlo se agregar al ranking y se agregara la información del partido.

```
        if not lt.isPresent(teams, match['away_team']):
            lt.addLast(teams, match['away_team'])
            home_team_dict = encontrar_diccionario_home(ranking, match['home_team'])
            team_format ={}
            #Agrega el nombre del equpo
            team_format['team'] = match['away_team']

            #agrega los puntos y vicorias derrotas y empates
            if int(match['home_score']) > int(match['away_score']):
                team_format['total_points'] = 0
                team_format['wins'] = 0
                team_format['draws'] = 0
                team_format['losses'] = 1

                if lt.isPresent(teams, match['home_team']) and home_added == False:
                    home_team_dict['wins'] += 1
                    home_team_dict['total_points'] += 3

            elif int(match['home_score']) == int(match['away_score']):
                team_format['total_points'] = 1
                team_format['wins'] = 0
                team_format['draws'] = 1
                team_format['losses'] = 0
                if lt.isPresent(teams, match['home_team']) and home_added == False:
                    home_team_dict['draws'] += 1
                    home_team_dict['total_points'] += 1
            else:
                team_format['total_points'] = 3
                team_format['wins'] = 1
                team_format['draws'] = 0
                team_format['losses'] = 0
                if lt.isPresent(teams, match['home_team']) and home_added == False:
                    home_team_dict['losses'] += 1

            #agrega la diferencia de gol
            team_format['goal_difference'] = int(match['away_score']) - int(match['home_score'])
            if lt.isPresent(teams, match['home_team']) and home_added == False:
                home_team_dict['goal_difference'] += int(match['home_score']) - int(match['away_score'])

            #agrega los penalty points
            if entry1 is None:
                team_format['penalty_points'] = 0
            else:
                team_format['penalty_points'] = penalty_goals_away
                if lt.isPresent(teams, match['home_team']) and home_added == False:
                    home_team_dict['penalty_points'] += penalty_goals_home

            #agraga llave matches
            team_format['matches'] = 1
            if lt.isPresent(teams, match['home_team']) and home_added == False:
                home_team_dict['matches'] += 1

            # agrega los own goal points
            if entry1 is None:
                team_format['own_goal_points'] = 0
            else:
                team_format['own_goal_points'] = own_goals_away
                if lt.isPresent(teams, match['home_team'])  and home_added == False :
                    home_team_dict['own_goal_points'] += own_goals_home


            #total de goles
            team_format['goals_for'] = int(match['away_score'])
            if lt.isPresent(teams, match['home_team']) and home_added == False:
                    home_team_dict['goals_for'] += int(match['home_score'])
            # total goles en contra
            team_format['goals_against'] = int(match['home_score'])

            if lt.isPresent(teams, match['home_team']) and home_added == False :
                    home_team_dict['goals_against'] += int(match['away_score'])

            lt.addLast(ranking, team_format)

    rankingF = addScorer_Req6(ranking, map_teams_goals)
    orderRanking = sa.sort(rankingF, sort_crit_ranking_7)
    añosTotales = mp.size(map_years)
    torneosTotales = lt.size(torneos)
    totalequipos = lt.size(ranking)
    totalcountries = lt.size(countries)
    citiesUniqueN = lt.size(citiesUnique)
    mostPopCity = str_mas_repetido(citiesMatches)

    TopnList = lt.subList(orderRanking, 1, n)


    if lt.size(TopnList) <= 6:
        return TopnList, añosTotales, torneosTotales, totalequipos, totalmatches, totalcountries, citiesUniqueN, mostPopCity
    else:
        newTop = FirstandALst(TopnList)
        return newTop, añosTotales, torneosTotales, totalequipos, totalmatches, totalcountries, citiesUniqueN, mostPopCity
```

Entrada	control, torneo, año, n

Salidas	TopnList, añosTotales, torneosTotales, totalequipos, totalmatches, totalcountries, citiesUniqueN, mostPopCity
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

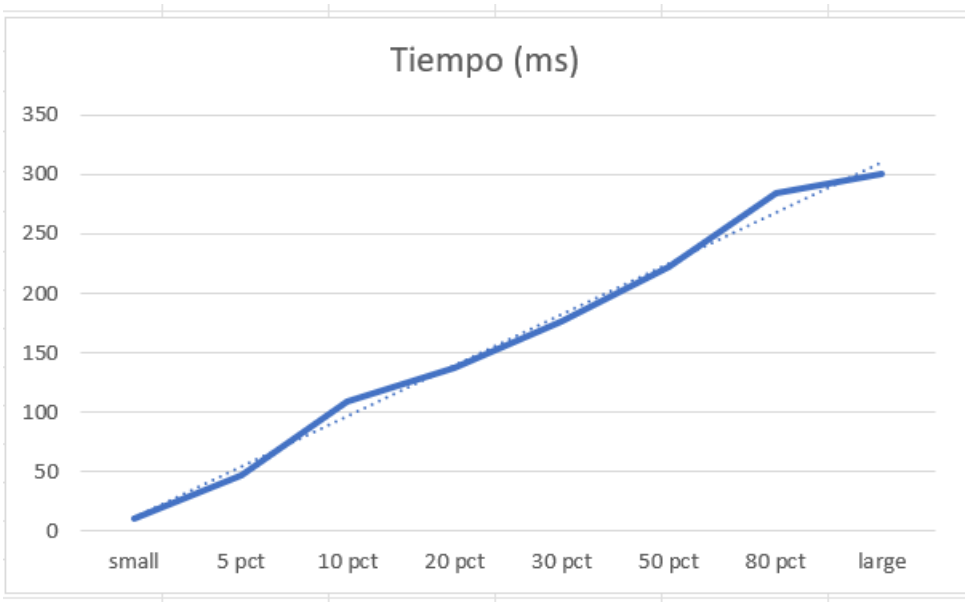
Pasos	**Complejidad**
Iteración por years_list	O (n)
Iteración por matches_list	O (n)
encontrar_diccionario	O (n)
AddLast	O (1)
isPresent	O (n)
Shell Sort	O n log n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesadores	**Intel(R) Core(TM) i7-10750H CPU**
Memoria RAM	32 GB

Entrada	**Tiempo (ms)**
small	10.665800000002491
5 pct	46.36209999999846
10 pct	109.26800000001094
20 pct	137.692199999990
30 pct	176.00699999992503
50 pct	222.4996000000392
80 pct	284.1129999998957
large	300.35479999985546

Graficas



Análisis

El requerimiento 6 tiene una complejidad de O (n log n) ya que después de hacer la iteración principal se hace un shell sort el cual cuenta con esa complejidad, aunque hay algunas funciones o(n) en el interior del ciclo estas reciben como parámetros listas muy pequeñas lo que hace que no afecte su complejidad. Esto se puede evidenciar con la gráfica, ya que podemos ver como los datos se encuentra casi sobre la línea de tendencia.

Por otro lado, cabe resaltar que las nuevas estructuras de datos implementadas, hace más eficiente el requerimiento 6. Mientras el requerimiento 6 del reto1 tiene una complejidad de O(n*3) este solo tiene una complejidad de O(n)

Requerimiento 7

Descripción

La función recorre los datos de una lista con los partidos del torneo, esta fue definida en la carga de datos, en base a este recorrido crea o actualiza la información solicitada por goleador, a su vez recopila información solicitada para saber el número de goles hechos en el partido y demás.

```
def req_7(data_structs,torneo,puntos):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    goles = 0
    penales = lt.newList("ARRAY_LIST")
    autogoles = lt.newList("ARRAY_LIST")
    partidos_torneos = data_structs["map_tournament"]
    torneo = me.getValue(mp.get(partidos_torneos,torneo))["datos"]
    total_torneos = mp.size(partidos_torneos)
    total_partidos_torneo = lt.size(torneo)
    puntajes_jugadores = mp.newMap(152,
                                   maptype="CHAINING",
                                   loadfactor=4)
    for dato in lt.iterator(torneo):
        goles+=int(dato["home_score"])
        goles+=int(dato["away_score"])
        añadir_jugador(dato,data_structs,puntajes_jugadores,goles,penales,autogoles)

    jugadores = mp.valueSet(puntajes_jugadores)
    total_jugadores = mp.size(puntajes_jugadores)
    jugadores_match = lt.newList("ARRAY_LIST")
    for dato in lt.iterator(jugadores):
        if dato["total_points"]==puntos:
            dato["avg_time [min]"] = dato["avg_time [min]"]//dato["total_goals"]
            lt.addLast(jugadores_match,dato)
    quk.sort(jugadores_match,criterio_req_7)
    jugadores_match_1_6 = lt.newList("ARRAY_LIST")

    if lt.size(jugadores_match)>=6:
        num_tabla = "The tournament results hass more than 6 records..."
        for i in range(1, 4):
            lt.getElement(jugadores_match,i)["last_goal"] = ultimogol(lt.getElement(jugadores_match,i)["scorer"],data_structs)
            lt.addLast(jugadores_match_1_6,lt.getElement(jugadores_match,i))
```

Una vez se tiene la lista con la informacion solicitada se busca el ultimo gol para cada jugador, esto se hace llamando al mapa con todos los goles de cada jugador, se organiza de lo mas reciente a lo mas antiguo y se retorna la primera posicion, posterior a esto para buscar la informacion solicitada de llama otro mapa en base a la id del ultimo gol (la id es igual a fecha_equipoLocal_equipovisitante), esta busqueda es rapida pues se hace en un mapa con la informacion del archivo Results en el cual las llaves son estas id y los valores son los datos del partido.

```
        for i in range(lt.size(jugadores_match)-2, lt.size(jugadores_match)+1):
            lt.getElement(jugadores_match,i)["last_goal"] = ultimogol(lt.getElement(jugadores_match,i)["scorer"],data_structs)
            lt.addLast(jugadores_match_1_6,lt.getElement(jugadores_match,i))
    else:
        num_tabla = "The tournament results hass less than 6 records..."
        for i in range(1, lt.size(jugadores_match)+1):
            lt.getElement(jugadores_match,i)["last_goal"] = ultimogol(lt.getElement(jugadores_match,i)["scorer"],data_structs)
            lt.addLast(jugadores_match_1_6,lt.getElement(jugadores_match,i))
    return jugadores_match_1_6 , total_torneos , total_jugadores , total_partidos_torneo , goles , lt.size(penales) , lt.size(autogoles), lt.size(jugadores_match) , num_tabla

def ultimogol(jugador,mapa):
    resultadosgoal = me.getValue(mp.get(mapa["scorer"],jugador))["datos"]
    quk.sort(resultadosgoal,criterio_ultimo_gol)
    datos = lt.getElement(resultadosgoal,1)
    id = datos["date"]+"_"+datos["home_team"]+"_"+datos["away_team"]
    resultadosresults = lt.getElement(me.getValue(mp.get(mapa["id_results"],id))["datos"],1)
    dato = lt.newList("ARRAY_LIST")
    data = {
        "date":str(datos["date"]),
        "tournament": resultadosresults["tournament"],
        "home_team":datos["home_team"],
        "away_team":datos["away_team"],
        "home_score": resultadosresults["home_score"],
        "away_score": resultadosresults["away_score"] ,
        "minute":datos["minute"],
        "penalty":datos["penalty"],
        "own_goal":datos["own_goal"],
    }
    lt.addLast(dato,data)
    return tabulate(dato["elements"] , headers= 'keys', tablefmt = 'grid')

def criterio_ultimo_gol(dato1,dato2):
    if dato1["date"] > dato2["date"]:
        return True
    else:
        return False
```

```python
def añadir_jugador(dato,data_structs,puntajes_jugadores,goles,penales,autogoles):
    ids = data_structs["id_goalscorers"]
    id = dato["date"]+"_"+dato["home_team"]+"_"+dato["away_team"]
    if mp.contains(ids,id):
        partido = me.getValue(mp.get(ids,id))["datos"]
        for gol in lt.iterator(partido):
            if not mp.contains(puntajes_jugadores,gol["scorer"]):
                jugador = {"scorer":gol["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":None
                           }
                mp.put(puntajes_jugadores,gol["scorer"],jugador)
            if mp.contains(puntajes_jugadores,gol["scorer"]):
                goleador = me.getValue(mp.get(puntajes_jugadores,gol["scorer"]))
                goleador["total_goals"]+=1
                goleador["total_points"]+=1
                goleador["avg_time [min]"]+=float(gol["minute"])
                if gol["team"] == dato["home_team"]:
                    if int(dato["home_score"])>int(dato["away_score"]):
                        goleador["scored in wins"]+=1
                    elif int(dato["home_score"])<int(dato["away_score"]):
                        goleador["scored in losses"]+=1
                    else:
                        goleador["scored in draws"]+=1

                if gol["team"] == dato["away_team"]:
                    if int(dato["home_score"])<int(dato["away_score"]):
                        goleador["scored in wins"]+=1
                    elif int(dato["home_score"])>int(dato["away_score"]):
                        goleador["scored in losses"]+=1
                    else:
                        goleador["scored in draws"]+=1

                if gol["penalty"] == "True" or gol["penalty"] == True:
                    lt.addLast(penales,id)
                    goleador["penalty_goals"]+=1
                    goleador["total_points"]+=1
                elif gol["own_goal"] == "True" or gol["own_goal"] == True:
                    lt.addLast(autogoles,id)
                    goleador["own_goals"]+=1
                    goleador["total_points"]-=1
```

Por ultimo se retornan unicamente los primeros 3 y los ultimos 3 datos en una lista del tipo ARRAY_LIST para ser tabulados a la hora de imprimir los resultados.

Entrada	data_structs , torneo , puntos
Salidas	jugadores_match_1_6 , total_torneos , total_jugadores , total_partidos_torneo , goles , lt.size(penales) , lt.size(autogoles), lt.size(jugadores_match) , num_tabla
Implementado (Sí/No)	SI

Análisis de complejidad

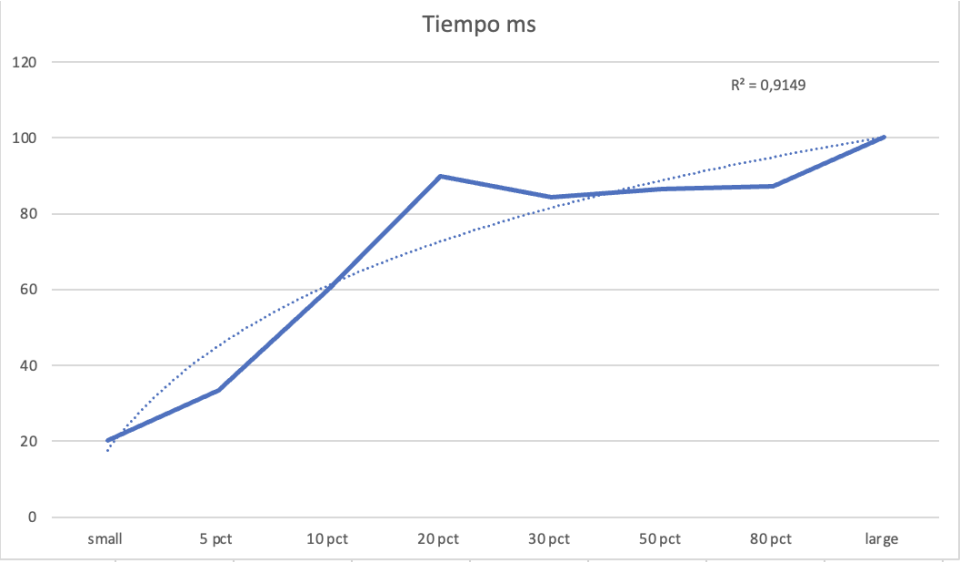
Pasos	**Complejidad**
Iteracion lista inicial	O (n)
Busqueda de los ultimos goles	O (n log n)
Armar lista para tabular	O (6)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesadores	M2
Memoria RAM	8 GB

Entrada	Tiempo (ms)
small	20.18
5 pct	33.43
10 pct	60.34
20 pct	89.84
30 pct	84.25
50 pct	86.43
80 pct	87.32
large	100.2

Análisis



El ciclo tiene una complejidad logaritmica, en el algoritmo podemos observar que lo mas tardado es encontrar los ultimos goles correspondientes a cada jugador, sin embargo, para este proceso lo mas tardado es organizar cada lista de los goles de cada jugador de mas reciente a mas antiguo mediante un qck.sort, este tiene una complejidad de n log n, por este la grafica se comporta de esa manera.

Requerimiento 8 BONO

Descripción

```python
def req_8(control, Equipo, Inicio, Final):
    """
    Función que soluciona el requerimiento 8
    """
    mapa = control['map_Req8']
    año_inicio = int(Inicio[0:4])
    año_final = int(Final[0:4])
    a = 0
    matches = 0
    winner = 0
    losses = 0
    draws = 0
    total_points = 0
    diferencia_goles = 0
    penalties = 0
    own_goals = 0
    goals_for = 0
    goals_against = 0
    jugadores = {}
    ListaRespuesta = lt.newList("ARRAY_LIST")
    total_home = 0
    total_away = 0
    for a in range(año_final-año_inicio+1):
        año = año_inicio + a
        a += 1
        llave = (Equipo, str(año))
        entry = mp.get(mapa, llave)
        if entry != None:
            ListaGoleador = lt.newList("ARRAY_LIST")
            last_game =  lt.newList("ARRAY_LIST")
            SubLista = me.getValue(entry)
            SubLista = SubLista['datos']
            ListaOrd = merg.sort(SubLista, sort_criteria_eng)
            size = lt.size(ListaOrd)
            i = 0
            for game in lt.iterator(ListaOrd):
                torneo = game['tournament']
                fecha = game['date']
                if torneo != "Friendly" and fecha >= Inicio and fecha <= Final:
                    matches +=1
                    winner = winner_losses_team(control, game, Equipo, fecha)[0] + winner
                    losses = winner_losses_team(control, game, Equipo, fecha)[1] + losses
                    draws = winner_losses_team(control, game, Equipo, fecha)[2] + draws
                    diferencia_goles = diferencia_goles_def(game, Equipo) + diferencia_goles
                    penalties = penalties_def(control, game, Equipo, fecha)[0] + penalties
                    own_goals = penalties_def(control, game, Equipo, fecha)[1] + own_goals
                    goals_for = goals_def(game, Equipo)[0] + goals_for
                    goals_against = goals_def(game, Equipo)[1] + goals_against
                    if game["home_team"] == Equipo:
                        total_home += 1
                    elif game["away_team"] == Equipo:
```

```
                total_home += 1
            elif game["away_team"] == Equipo:
                total_away += 1
        i +=1
        if i == size:
            last_game = game
    if matches > 0:
        total_points = (winner*3) + draws
        jugadores = jugadores_def(control, game, Equipo, fecha, jugadores)
        if jugadores:
            maximo_jugador, maximo_goles = max(jugadores.items(), key=lambda x: x[1])
        else:
            maximo_jugador, maximo_goles = ("Nadie", 0)
        avg_time = avg_time_def(control, game, año, maximo_jugador, maximo_goles, Equipo)
        lt.addLast(ListaGoleador, {'player': maximo_jugador, 'goals': maximo_goles, 'matches': matches, 'avg time': avg_time})
        lt.addLast(ListaRespuesta, {'year': año, 'matches': matches,'total points': total_points,'goal_diference': diferencia_goles,
                                    'penalties': penalties, 'own_goals': own_goals,
                                    'winner':winner, 'draws': draws,
                                    'losses': losses, 'goals_for': goals_for,
                                    'goals_against': goals_against, 'top_scorer': ListaGoleador})
        matches = 0
        winner = 0
        losses = 0
        draws = 0
        diferencia_goles = 0
        total_points = 0
        penalties = 0
        own_goals = 0
        goals_for = 0
        goals_against = 0
        jugadores = {}
return ListaRespuesta, last_game, total_home, total_away
```

En esta función se recorre por los años que piden como parámetro haciendo uso de los mapas. Para el desarrollo del requerimiento 8, se utilizaron 2 mapas del catálogo. Primero, el mapa ‘map_Req8’ que contiene llaves valor de tipo:

{(año, nombre Equipo): [partido1] , [partido 2] … [partido n]}

Y el mapa “map_Req8_GoalScorers” que contiene llaves valor de tipo:

{(año, nombre Equipo): [gol1] , [gol2] … [gol n]}

```
def avg_time_def(control, game, año, maximo_jugador, maximo_goles, Equipo):
    if maximo_jugador == "Nadie":
        return -1
    else:
        mapa = control['map_Req8_GoalScorers']
        año = str(año)
        llave = (Equipo, año[0:4])
        entry = mp.get(mapa, llave)
        if entry != None:
            SubLista = me.getValue(entry)
            SubLista = SubLista['datos']
            listaOrd = quk.sort(SubLista, sort_criteria_eng)
            minutos_totales = 0
            for game in lt.iterator(listaOrd):
                Own_goal = game['own_goal']
                scorer = game['scorer']
                minuto = int(float(game['minute']))
                team = game['team']
                if  scorer == maximo_jugador and Own_goal == 'False' and team == Equipo:
                    minutos_totales = minutos_totales + minuto
            return minutos_totales/maximo_goles
        else:
            return -1
```

Esta función se utiliza para calcular el average time del máximo jugador del año que se está iterando.

```python
def jugadores_def(control, game, Equipo, fecha, jugadores):
    mapa = control['map_Req8_GoalScorers']
    llave = (Equipo, str(fecha[0:4]))
    entry = mp.get(mapa, llave)
    if entry != None:
        SubLista = me.getValue(entry)
        SubLista = SubLista['datos']
        listaOrd = quk.sort(SubLista, sort_criteria_eng)
        for game in lt.iterator(listaOrd):
            team = game['team']
            own_goal = game['own_goal']
            if team == Equipo and own_goal == 'False':
                nombre_jugador = game['scorer']
                if nombre_jugador in jugadores:
                    jugadores[nombre_jugador] += 1
                else:
                    jugadores[nombre_jugador] = 1
        return jugadores
```

Esta función se utiliza para calcular los jugadores que anotaron goles en el año iterado, se utiliza para después sacar el máximo y hallar los datos del máximo goleador de cada año.

```python
def goals_def(game, Equipo):
    away_score = int(game["away_score"])
    home_score = int(game["home_score"])
    away_team = game["away_team"]
    home_team = game["home_team"]
    if home_team == Equipo:
        return home_score, away_score
    elif away_team == Equipo:
        return away_score, home_score
```

Esta función se utiliza para calcular los goles que anoto y que le anotaron al equipo que se mete como parámetro por cada año.

```python
def penalties_def(control, game, Equipo, fecha):
    if int(game["home_score"]) > 0 or int(game["away_score"]) > 0:
        mapa = control['map_Req8_GoalScorers']
        llave = (Equipo, str(fecha[0:4]))
        entry = mp.get(mapa, llave)
        if entry != None:
            SubLista = me.getValue(entry)
            SubLista = SubLista['datos']
            listaOrd = quk.sort(SubLista, sort_criteria_eng)
            total_penalties = 0
            total_autogoles = 0
            for game in lt.iterator(listaOrd):
                team = game['team']
                penalty = game['penalty']
                own_goal = game['own_goal']
                if team == Equipo and penalty == 'True':
                    total_penalties += 1
                if team == Equipo and own_goal == 'True':
                    total_autogoles += 1
            return total_penalties, total_autogoles
        else:
            return 0, 0
    else:
        return 0, 0
```

Esta función se utiliza para calcular los penaltis de cada partido iterado por cada año y se hace utilizando el mapa creado para este requerimiento que tiene como llave una tupla que tiene el año y tupla y los valores los partidos de este año para este equipo.

```
def diferencia_goles_def(game, Equipo):
    away_score = int(game["away_score"])
    home_score = int(game["home_score"])
    away_team = game["away_team"]
    home_team = game["home_team"]
    if away_team == Equipo:
        return away_score - home_score
    elif home_team == Equipo:
        return home_score - away_score
```

Esta función se utiliza para calcular la diferencia de goles de cada partido por cada año.

```
def winner_losses_team(control, game, Equipo, fecha):
    if game["home_team"] == Equipo:
        if game["home_score"] > game["away_score"]:
            return 1, 0, 0
        elif game["home_score"] < game["away_score"]:
            return 0, 1, 0
        else:
            mapa = control['map_TeamShootouts']
            entry = mp.get(mapa, Equipo)
            if entry != None:
                SubLista = me.getValue(entry)
                SubLista = SubLista['datos']
                listaOrd = quk.sort(SubLista, sort_criteria_eng)
                Pos = busqueda_binaria_req3(listaOrd, fecha)
                datos = lt.getElement(listaOrd, Pos)
                winner = datos["winner"]
                if winner == Equipo:
                    return 1, 0, 0
                else:
                    return 0, 1, 0
            else:
                return 0, 0, 1
    if game["away_team"] == Equipo:
        if game["home_score"] < game["away_score"]:
            return 1, 0, 0
        elif game["home_score"] > game["away_score"]:
            return 0, 1, 0
        else:
            mapa = control['map_TeamShootouts']
            entry = mp.get(mapa, Equipo)
            if entry != None:
                SubLista = me.getValue(entry)
                SubLista = SubLista['datos']
                listaOrd = quk.sort(SubLista, sort_criteria_eng)
                Pos = busqueda_binaria_req3(listaOrd, fecha)
                datos = lt.getElement(listaOrd, Pos)
                winner = datos["winner"]
                if winner == Equipo:
                    return 1, 0, 0
                else:
                    return 0, 1, 0
            else:
                return 0, 0, 1
```

Esta función se utiliza para calcular en cada partido si el equipo que se mete como parámetro gano, perdió o empato. Esto se hace primero revisando si los goles anotados son mayores a los que les anotaron y en el caso en que empatan se utiliza el mapa de los goleadores y se revisa si en ese partido se metió goles por penalty.

Para el desarrollo del requerimiento se itero sobre el mapa que contiene la lista que contiene los partidos de un equipo por cada año y se adquirió la información necesaria. Para obtener la información faltante de los partidos se itero sobre el mapa que contiene los goles por cada año y equipo.

Entrada	control, Equipo, Inicio, Final
Salidas	ListaRespuesta, Last_game, total_home, total_away
Implementado (Sí/No)	SI

Análisis de complejidad

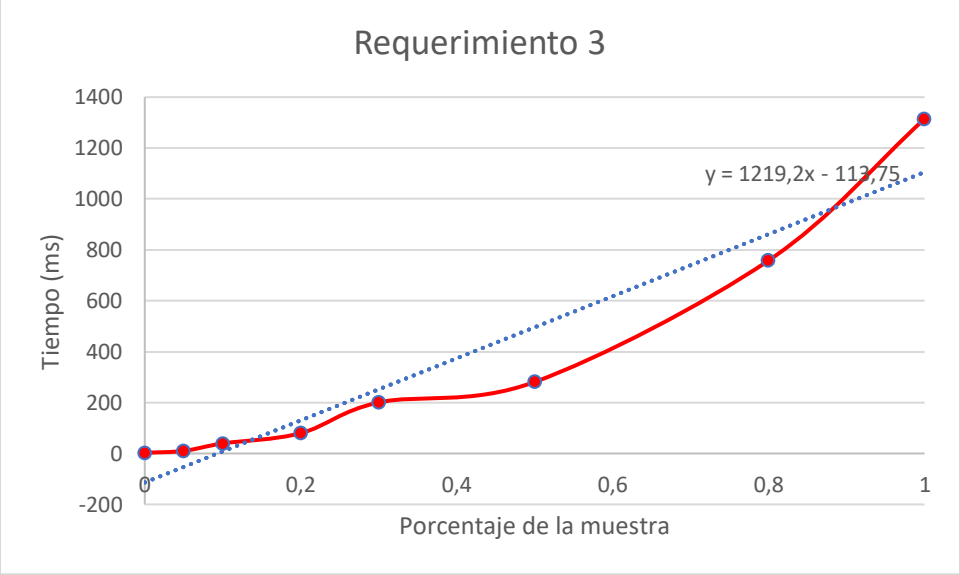
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Iteración por Equipo por año	O(n)
Búsqueda por cada gol del equipo y del año	O(n)
TOTAL	***O(n^2)***

Pruebas Realizadas

Procesadores	**intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz, 1201**
Memoria RAM	12 GB

Entrada	**Tiempo (ms)**
small	2,66
5 pct	10,15
10 pct	39,45
20 pct	79,71
30 pct	200,64
50 pct	281
80 pct	758,37
large	1314,715



Análisis

Teniendo en cuenta las pruebas realizadas y el análisis de complejidad, podemos concluir que el uso de estructuras de datos como mapas mejoro significativamente la eficiencia de la función req_8. Por otro lado, el requerimiento tiene una complejidad cuadrática ya que tiene que iterar primero el mapa de resultados por equipos por cada año y por cada partido de acá se itera en el mapa de goles de este equipo por este año. Aunque la complejidad con notación O se mantiene igual, si uno revisa el n va a disminuir puesto que ahora solo recorrerá los partidos por equipo por año y no el de todos los equipos.