ANÁLISIS DEL RETO

Rodrigo Paz Londoño, 202225425, r.pazl@uniandes.edu.co

Juan Diego Rodríguez Barragán, 202221822, jd.rodriguezb12@uniandes.edu.co

Samuel Escobar Pineda, 202310474, sa.escobarp1@uniandes.edu.co

Requerimiento 1

Descripción

```python
def req_1(data, numero_partidos, equipo, condicion_equipo):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    resultados = data["results"]
    total_de_partidos = 0
    rta = lt.newList("ARRAY_LIST")
    for r in lt.iterator(resultados):
        if condicion_equipo == "Local":
            if equipo == r["home_team"]:
                total_de_partidos += 1
                diccionario ={"date": r["date"], "home_team":r["home_team"], "away_team": r["away_team"],
                              "home_score": r["home_score"], "away_score": r["away_score"], "country": r["country"],
                              "city": r["city"], "tournament": r["tournament"]}
                lt.addLast(rta, diccionario)
        elif condicion_equipo == "Visitante":
            if equipo == r["away"]:
                total_de_partidos += 1
                diccionario ={"date": r["date"], "home_team":r["home_team"], "away_team": r["away_team"],
                              "home_score": r["home_score"], "away_score": r["away_score"], "country": r["country"],
                              "city": r["city"], "tournament": r["tournament"]}
                lt.addLast(rta, diccionario)
        elif condicion_equipo == "indiferent":
            if equipo == r["home_team"] or equipo == r["away_team"]:
                total_de_partidos += 1
                diccionario ={"date": r["date"], "home_team":r["home_team"], "away_team": r["away_team"],
                              "home_score": r["home_score"], "away_score": r["away_score"], "country": r["country"],
                              "city": r["city"], "tournament": r["tournament"]}
                lt.addLast(rta, diccionario)
    rta = lt.subList(rta, 1, int(numero_partidos))
    return total_de_partidos, rta
```

Descripción

La función principal del requerimiento uno es devolver una lista de partidos de un equipo en específico de longitud dada por parámetro con su información deseada la cual es: Fecha del partido, equipo local, equipo visitante, país del encuentro, ciudad donde se disputa el encuentro, marcador del equipo local (goles del local), marcador del equipo visitante (goles del visitante). Dada una condición del equipo ya sea: local, visitante o indiferente, si es indiferente toma todos los partidos de ese equipo sin darle relevancia a si en el partido el equipo jugó de visitante o local.

Entrada	Estructuras de datos del modelo, numero de equipos que quiere consultar (longitud de la lista), nombre del equipo que quiere consultar y la condición del equipo (Local, Visitante o indiferente).

Salidas	El número de partidos en el que el equipo participó según su condición y la lista con el número de partidos que se quisieron buscar.
Implementado (Sí/No)	Si. Implementado por: Todos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Declaración de variables y creación de listas	O(1)
Ciclo for	O(n **)**
Comparaciones y suma	***O(1)***
Creación de diccionario	***O(1)***
Agregar al final de una lista (ARRAYLIST)	***O(1)***
TOTAL	O(n)

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	0,736
5 pct	3,32
10 pct	4,82
20 pct	5
30 pct	6,07
50 pct	9,64
80 pct	14,1
large	28

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	**Tiempo (ms)**
small	0,959

5 pct	4,19
10 pct	6,35
20 pct	8,54
30 pct	11,6
50 pct	25,5
80 pct	34,1
large	41

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**
small	0.519
5 pct	1.83
10 pct	3.38
20 pct	5.4
30 pct	4.81
50 pct	8.49
80 pct	11.1
large	16.6

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	**Salida**	**Tiempo (ms)**
small	El número de partidos en el que el equipo participó según su condición y la lista con el número de partidos que se quisieron buscar.	0,736
5 pct		3,32
10 pct		4,82
20 pct		5
30 pct		6,07
50 pct		9,64
80 pct		14,1
large		28

Maquina 2:

Muestra	Salida	Tiempo (ms)
small	El número de partidos en el que el equipo participó según su condición y la lista con el número de partidos que se quisieron buscar.	0,959
5 pct		4,19
10 pct		6,35
20 pct		8,54
30 pct		11,6
50 pct		25,5
80 pct		34,1
large		41

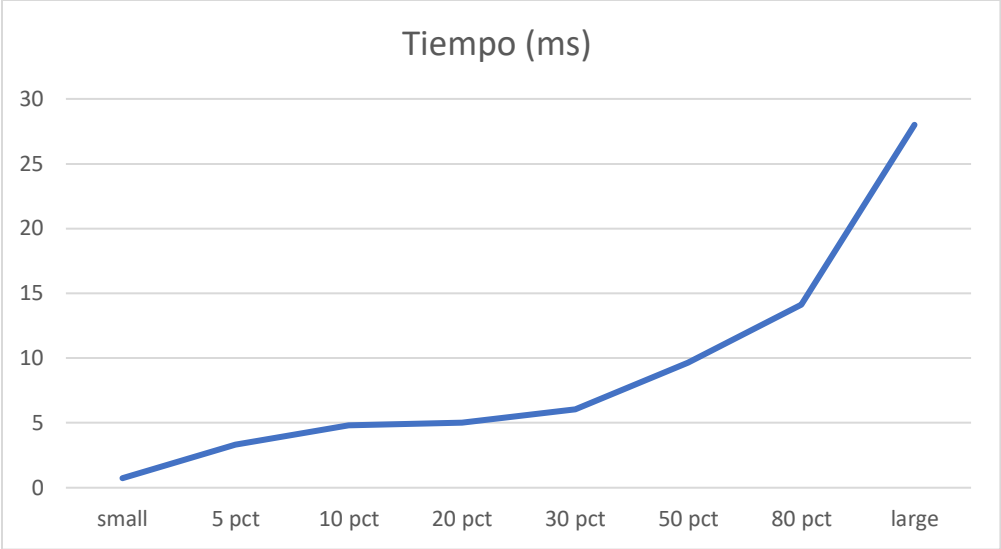
Maquina 3:

Muestra	Salida	Tiempo (ms)
small	El número de partidos en el que el equipo participó según su condición y la lista con el número de partidos que se quisieron buscar.	0.519
5 pct		1.83
10 pct		3.38
20 pct		5.4
30 pct		4.81
50 pct		8.49
80 pct		11.1
large		16.6

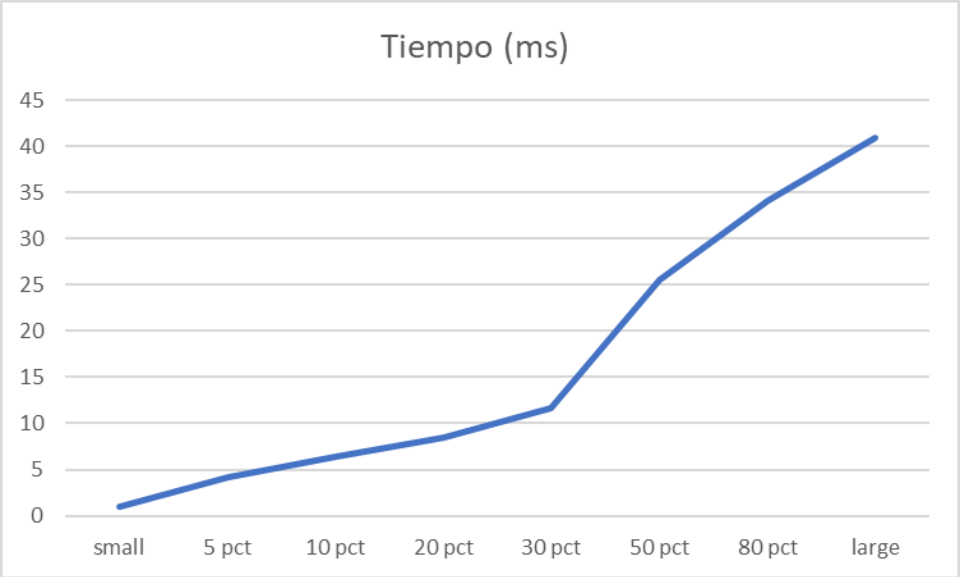
Graficas

Las gráficas con la representación de las pruebas realizadas.

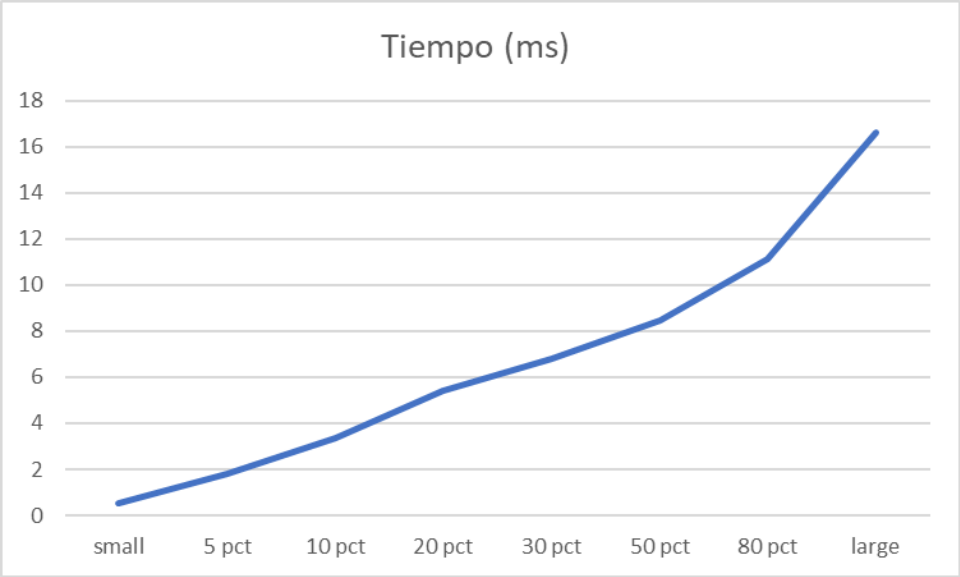
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

La complejidad es O(n), dado que por cada interación del For que cumpla con la condición de comparación que se ejecuta allí. En las gráficas es posible apreciar un comportamiento similar al lineal.

Requerimiento 2

Descripción

```
def req_2(data, number_goals, name_player):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    goals = data["goal_scorers"]
    scorer_goals = lt.newList("ARRAY_LIST")
    total_goals = 0
    final_scorer_goals = lt.newList("ARRAY_LIST")

    for goal in lt.iterator(goals):
        if goal["scorer"] == name_player:
            lt.addLast(scorer_goals, goal)
            total_goals += 1
    sorted_date_minute = sa.sort(scorer_goals, cmp_elementos_by_fecha_y_minuto2)

    i = 1
    while i <= lt.size(sorted_date_minute) and i <= int(number_goals):
        element = lt.getElement(sorted_date_minute, i)
        lt.addLast(final_scorer_goals, element)
        i += 1
    size_list = lt.size(final_scorer_goals)

    return final_scorer_goals, size_list, total_goals
```

El requerimiento 2 busca un numero de anotaciones específico para un jugador especifico, las anotaciones se retornan con su información requerida, la cual es: Fecha del partido, equipo local, equipo visitante, equipo del jugador, minuto en el que se marcó el gol, tipo de anotación, si fue por falta desde el penal, tipo de anotación, si fue autogol.

Entrada	Estructuras de datos del modelo, numero de goles que se quiere consultar y el jugador que se quiere consultar.
Salidas	El total de anotaciones del jugador y la lista con el número de anotaciones deseados del jugador deseado
Implementado (Sí/No)	Si. Implementado por: Todos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Peor caso

Pasos	**Complejidad**
Declaración de variables y creación de listas	**O(1)**
Ciclo for y ciclo while	**O(n) + O(n)**

Comparaciones y suma	***O(1)***
Función de ordenamiento (SHELL SORT)	***O(n^2)***
Agregar al final de una lista (ARRAYLIST)	***O(1)***
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	0,626
5 pct	1,04
10 pct	2,15
20 pct	2,79
30 pct	4,85
50 pct	7,02
80 pct	11,7
large	30,2

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	**Tiempo (ms)**
small	0,499
5 pct	2,54
10 pct	3,76
20 pct	4,98
30 pct	7,54
50 pct	13,4
80 pct	61,4
large	90,6

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**
small	0,524
5 pct	0,740
10 pct	2,07
20 pct	4,02
30 pct	4,9
50 pct	7,75
80 pct	9,27
large	14,3

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	**Salida**	**Tiempo (ms)**
small	El total de anotaciones del jugador y la lista con el número de anotaciones deseados del jugador deseado	0,626
5 pct		1,04
10 pct		2,15
20 pct		2,79
30 pct		4,85
50 pct		7,02
80 pct		11,7
large		30,2

Maquina 2:

Muestra	**Salida**	**Tiempo (ms)**
small	El total de anotaciones del jugador y la lista con el número de anotaciones deseados del jugador deseado	0,499
5 pct		2,54
10 pct		3,76
20 pct		4,98
30 pct		7,54
50 pct		13,4
80 pct		61,4
large		90,6

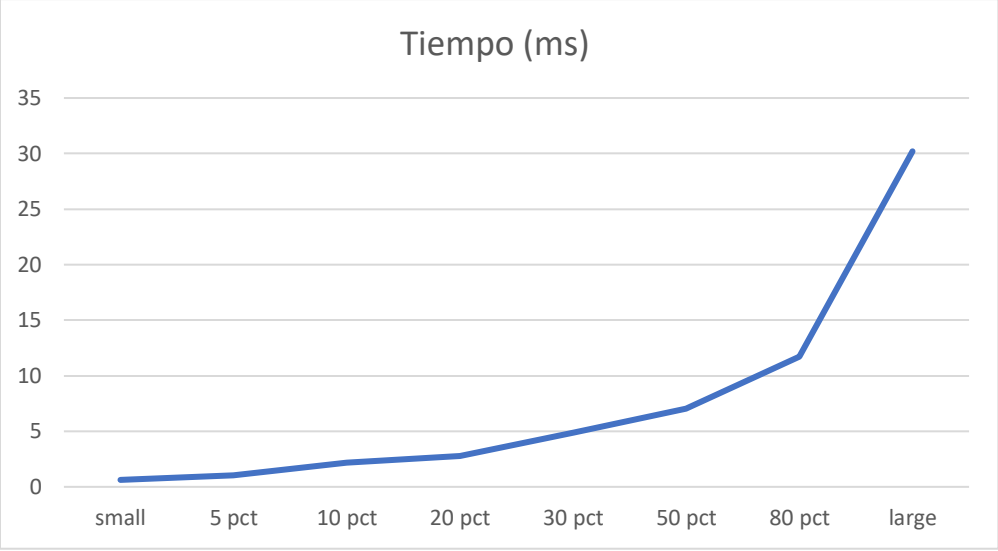
Maquina 3:

Muestra	Salida	Tiempo (ms)
small	El total de anotaciones del jugador y la lista con el número de anotaciones deseados del jugador deseado	0,524
5 pct		0,740
10 pct		2,07
20 pct		4,02
30 pct		4,9
50 pct		7,75
80 pct		9,27
large		14,3

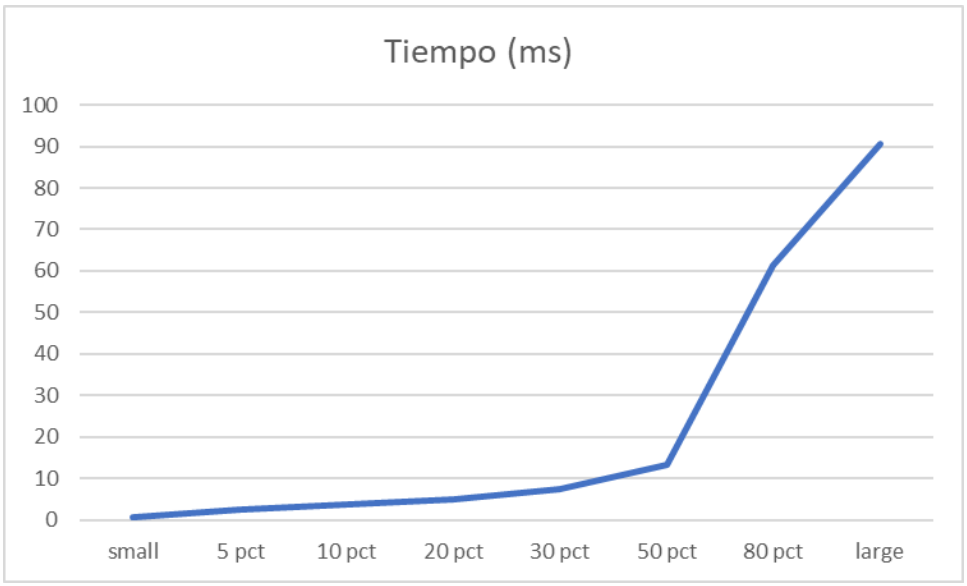
Graficas

Las gráficas con la representación de las pruebas realizadas.

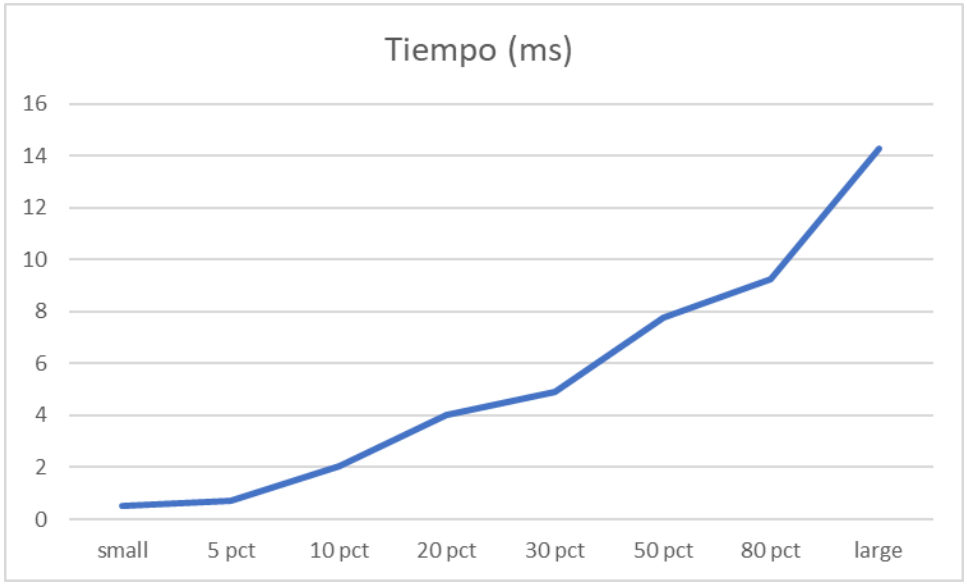
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

El peor caso de la complejidad algorítmica de la funcion seria O(n^2), debido a que, en el análisis de complejidad ordenar una lista mediante shell sort tiene el mayor peso. Como la lista que se recorre con el ciclo while, depende del primer ciclo con for. La complejdad de ambos ciclos se suman por lo que ambos ciclos suman o(n) por eso es mayor ordenar con shell sort.

En esta funcion todas las listas creadas son tipo ARRAY_LIST, debido a la complejidad espacial y al uso de funciones como lt.getElement o lt.addLast. ARRAY_LIST tiene menor complejidad espacial que SINGLE_LINKED y en el caso de getElement tiene menor complejidad temporal. Como las listas se editan mucho en distintas posiciones, la mejor opcion es ARRAY_LIST.

En esta función se hace el uso de shell sort y no se compara con el resto de las funciones de ordenamiento, debido al volumen de información que se maneja.

- Shell sort ordena mejor que insertion y selection para cualquier tamaño de datos
- Merge o quick sort ordenan mejor que shell, sin embargo, esto aplica únicamente cuando se maneja un volumen de informacion muy alto.

Como en este caso se maneja poca información (50.000 datos) la mejor opcion es shell sort

Requerimiento 3

```python
def req_3(data, team_name, inicial_date, final_date):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    matches = data["results"]
    range_team_matches = lt.newList("ARRAY_LIST")
    goals = data["goal_scorers"]
    range_goals_team = lt.newList("ARRAY_LIST")
    total_team_matches = 0
    total_home_team = 0
    total_away_team = 0

    for match in lt.iterator(matches):
        date = match["date"]
        if (match["home_team"] == team_name or match["away_team"] == team_name) and rango_by_fecha(date, inicial_date, final_date) == True:
            lt.addLast(range_team_matches, match)
            total_team_matches += 1
            if match["home_team"] == team_name:
                total_home_team += 1
            elif match["away_team"] == team_name:
                total_away_team += 1

    for goal in lt.iterator(goals):
        date2 = goal["date"]
        if (goal["home_team"] == team_name or goal["away_team"] == team_name) and rango_by_fecha(date2, inicial_date, final_date) == True:
            lt.addLast(range_goals_team, goal)

    complete_team_matches = find_type_goals(range_team_matches, range_goals_team)
    sorted_date = sa.sort(complete_team_matches, cmp_partidos_by_fecha)
    size_list = lt.size(sorted_date)

    return sorted_date, size_list, total_team_matches, total_home_team, total_away_team
```

```python
def find_type_goals(range_team_matches, range_goals_team):

    data_team_matches = lt.newList("ARRAY_LIST")
    data_match = {}

    for match in lt.iterator(range_team_matches):
        match_info = (match['date'], match['home_team'], match['away_team'])
        data_match[match_info] = {
                                  "date" : match["date"],
                                  "home_score" : match["home_score"],
                                  "away_score" : match["away_score"],
                                  "home_team" : match["home_team"],
                                  "away_team" : match["away_team"],
                                  "city" : match["city"],
                                  "country" : match["country"],
                                  "tournament" : match["tournament"],
                                  "penalty" : "Unknown",
                                  "own_goal" : "Unknown"
                                 }

    for goal in lt.iterator(range_goals_team):
        match_info2 = (goal['date'], goal['home_team'], goal['away_team'])

        if match_info2 in d  (variable) match_info2: tuple
            if goal["penalt
                data_match[match_info2]["penalty"] = "True"

            if data_match[match_info2]["penalty"] != "True" and goal["penalty"] == "False":
                data_match[match_info2]["penalty"] = "False"

            if goal["own_goal"] == "True":
                data_match[match_info2]["own_goal"] = "True"

            if data_match[match_info2]["own_goal"] != "True" and goal["own_goal"] == "False":
                data_match[match_info2]["own_goal"] = "False"

    for info in data_match.values():
        lt.addLast(data_team_matches, info)

    return data_team_matches
```

Descripción

El requerimiento 3 filtra los partidos de un equipo dado que se encuentran dentro de un rango de fechas dado y retorna una lista de los partidos con una información deseada, la cual es: Fecha del partido, marcador del equipo local (goles del local), marcador del equipo visitante (goles del visitante), equipo local, equipo visitante, país del encuentro, ciudad donde se disputa el encuentro, nombre del torneo asociado, anotación, si el partido presento goles por faltas desde el punto penal, anotación, si el partido presento autogoles.

Entrada	Estructuras de datos del modelo, nombre del equipo, fecha mínima y fecha máxima.
Salidas	Numero de los partidos disputado, número total de partidos disputados como local, número total de partidos disputados como visitante y la lista que contiene cada partido
Implementado (Sí/No)	Si. Implementado por: *Juan Diego Rodríguez Barragán*

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Peor caso

Pasos	Complejidad
Declaración de variables y creación de listas	**O(1)**
2 ciclos for	**O(n^2)**
Comparaciones y suma	***O(1)***
Función find_type_goals	***O(n)***
Función de ordenamiento (SHELL SORT)	***O(n^2)***
Agregar al final de una lista (ARRAYLIST)	***O(1)***
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	Tiempo (ms)
small	3,2
5 pct	13,1
10 pct	14,1
20 pct	37,8
30 pct	47,1
50 pct	77,8
80 pct	114,02
large	203,02

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	Tiempo (ms)
small	2,56
5 pct	19,3

10 pct	27,3
20 pct	62,7
30 pct	7,40E+01
50 pct	1,03E+02
80 pct	1,88E+02
large	2,02E+02

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**
small	27,3
5 pct	31,7
10 pct	22,4
20 pct	35,7
30 pct	61,4
50 pct	95,7
80 pct	1,4e+02
large	1,85e+02

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	**Salida**	**Tiempo (ms)**
small	Numero de los partidos disputado, número total de partidos disputados como local, número total de partidos disputados como visitante y la lista que contiene cada partido	3,2
5 pct		13,1
10 pct		14,1
20 pct		37,8
30 pct		47,1
50 pct		77,8
80 pct		114,02
large		203,02

Maquina 2:

Muestra	Salida	Tiempo (ms)
small	Numero de los partidos disputado, número total de partidos disputados como local, número total de partidos disputados como visitante y la lista que contiene cada partido	2,56
5 pct		19,3
10 pct		27,3
20 pct		62,7
30 pct		7,40E+01
50 pct		1,03E+02
80 pct		1,88E+02
large		2,02E+02

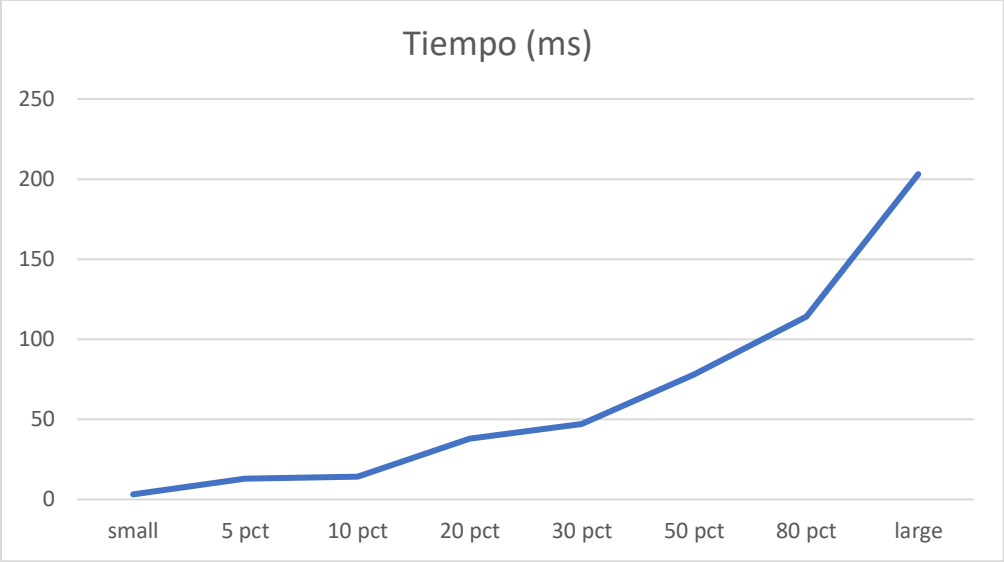
Maquina 3:

Muestra	Salida	Tiempo (ms)
small	Numero de los partidos disputado, número total de partidos disputados como local, número total de partidos disputados como visitante y la lista que contiene cada partido	27,3
5 pct		31,7
10 pct		22,4
20 pct		35,7
30 pct		61,4
50 pct		95,7
80 pct		1,4e+02
large		1,85e+02

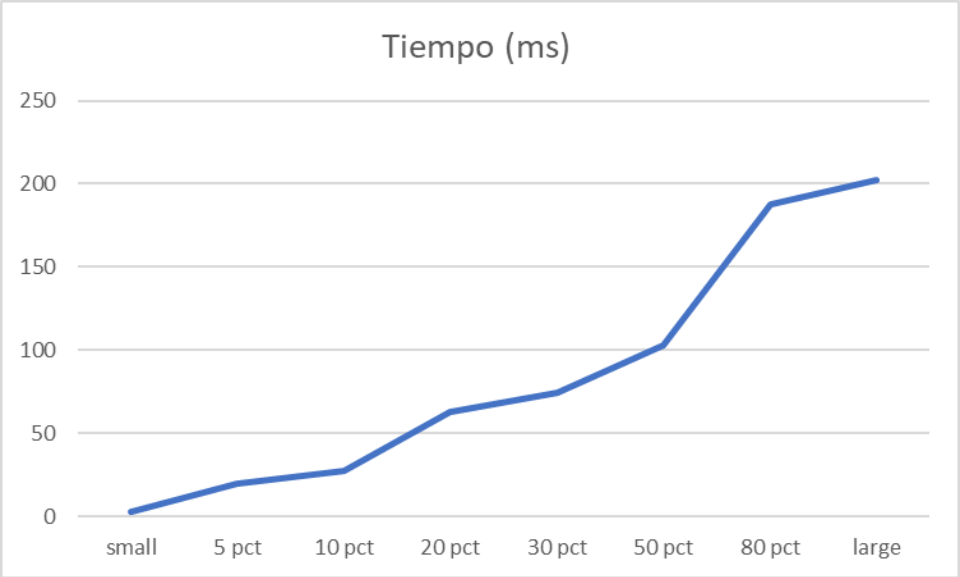
Graficas

Las gráficas con la representación de las pruebas realizadas.

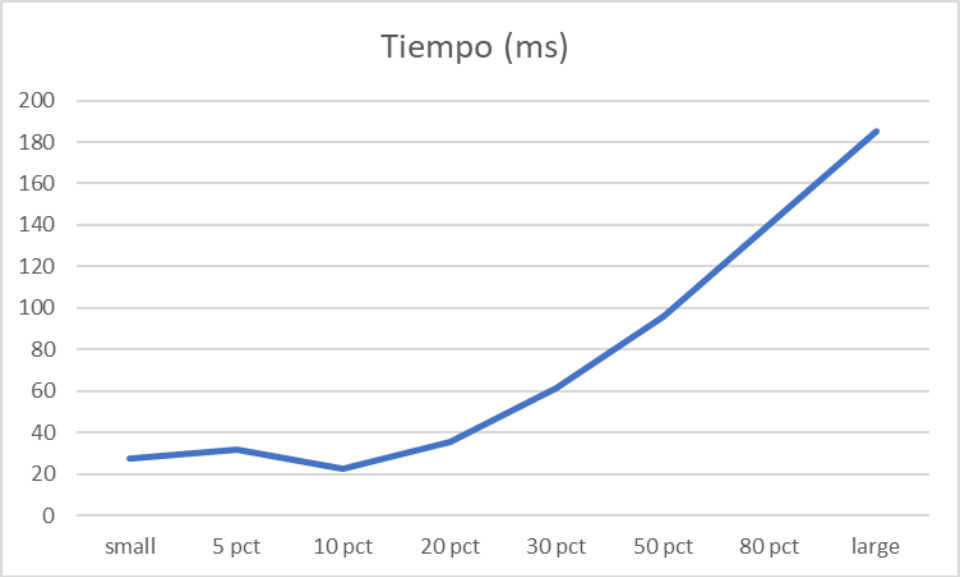
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

El peor caso de la complejidad algoritmica de la funcion seria O(n^2), debido a que, en el análisis de complejidad ordenar una lista mediante shell sort y los primeros ciclos tienen el mayor peso.

- En la funcion find_type_goals el primer for solo se usa para agregar info a un dict, ya el segundo ciclo se usa para recorrer las llaves del dict y cambiar algunas llaves por ende como ambos ciclos dependen en si es O(n) + O(n). El tercer ciclo del diccionario es solo para acceder a los valores y agregar a la lista por lo O(n). En general esta funcion es O(n).

Ya la funcion del requerimiento, ambos for recorren una lista distinta y sacan informacion de está creando variables o listas nuevas, como no dependen de si mismas estas O(n^2).

En esta funcion todas las listas creadas son tipo ARRAY_LIST, debido a la complejidad espacial y al uso de funciones como lt.getElement o lt.addLast. ARRAY_LIST tiene menor complejidad espacial que SINGLE_LINKED y en el caso de getElement tiene menor complejidad temporal. Como las listas se editan mucho en distintas posiciones, la mejor opcion es ARRAY_LIST.

En esta función se hace el uso de shell sort y no se compara con el resto de las funciones de ordenamiento, debido al volumen de información que se maneja.

- Shell sort ordena mejor que insertion y selection para cualquier tamaño de datos
- Merge o quick sort ordenan mejor que shell, sin embargo, esto aplica únicamente cuando se maneja un volumen de informacion muy alto.

Como en este caso se maneja poca informacion (50.000 datos) la mejor opcion es shell sort

Requerimiento 4

Descripción

```python
def req_4(data, torneo, lim_inf, lim_sup):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    partidos_torneo = 0
    paises = lt.newList("ARRAY_LIST")
    ciudades = lt.newList()
    resultados = data["results"]
    partidos_penales = data["shootouts"]
    penales = 0
    rta = lt.newList("ARRAY_LIST")
    for r in lt.iterator(resultados):
        winner = "Unknown"
        if r["tournament"] == torneo:
            for s in lt.iterator(partidos_penales):
                if r["home_team"] == s["home_team"] and r["away_team"] == s["away_team"]:
                    penales += 1
                    winner = s["winner"]
            fecha = r["date"]
            if rango_by_fecha(fecha, lim_inf, lim_sup) == True:
                diccionario = {"date": r["date"], "tournament": r["tournament"],
                               "country": r["country"], "city": r["city"],
                               "home_team":r["home_team"], "away_team": r["away_team"],
                               "home_score": r["home_score"], "away_score": r["away_score"],
                               "winner" : winner}
                partidos_torneo +=1
                lt.addLast(rta, diccionario)
                if lt.isPresent(paises, r["country"]) == 0:
                    lt.addLast(paises, r["country"])
                if lt.isPresent(ciudades, r["city"]) == 0:
                    lt.addLast(ciudades, r["city"])

    return partidos_torneo, lt.size(paises), lt.size(ciudades), penales, rta
```

El requerimiento 4 filtra los partidos de un torneo especifico dentro de un rango de fechas especifico, y retorna una lista de los partidos con una información específica, la cual es: Fecha del partido, país del encuentro, ciudad donde se disputa el encuentro, equipo local, equipo visitante, marcador del equipo local (goles del local), marcador del equipo visitante (goles del visitante), anotación, si el partido se definió por penales, nombre del equipo ganador por definiciones desde el punto penal (si existe), si no existe se toma como “Unknown”.

Entrada	Estructuras de datos del modelo, nombre del torneo, fecha mínima y fecha máxima.
Salidas	El total de partidos relevantes al torneo, el total de países involucrados en el torneo, el total de ciudades donde se disputan los partidos del torneo, el total de partidos definidos por cobros de punto penal y la lista de cada partido con su información.
Implementado (Sí/No)	Si. Implementado por: Samuel Escobar Pineda.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Declaración de variables y creacióon de listas	O(1)
Primer ciclo for	O(n)
Comparaciones, suma y declaraciones de variable	***O(1)***
Is Present en Array List	***O(n)***
Comparaciones y agregar al final de un Array_list	***O(1)***
Segundo ciclo for	O(n)
Declaración de varables, suma y comparaciones	***O(1)***
Is Present en Arraylist	***O(n)***
Comparaciones y agregar al final de un Arraylist	***O(1)***
Total	$O(n^2)$

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	4,02
5 pct	12,3
10 pct	11,2
20 pct	23,3
30 pct	26,1
50 pct	60,7
80 pct	91,7
large	208,2

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	**Tiempo (ms)**

small	5,07
5 pct	15,6
10 pct	19,9
20 pct	55
30 pct	76,4
50 pct	1,54E+02
80 pct	1,89E+02
large	2,10E+02

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**
small	3,95
5 pct	12,3
10 pct	17,8
20 pct	45
30 pct	68,2
50 pct	132,7
80 pct	176,4
large	223

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	**Salida**	**Tiempo (ms)**
small	El total de partidos relevantes al torneo, el total de países involucrados en el torneo, el total de ciudades donde se disputan los partidos del torneo, el total de partidos definidos por cobros de punto penal y la lista de cada partido con su información.	4,02
5 pct		12,3
10 pct		11,2
20 pct		23,3
30 pct		26,1
50 pct		60,7
80 pct		91,7
large		208,2

Maquina 2:

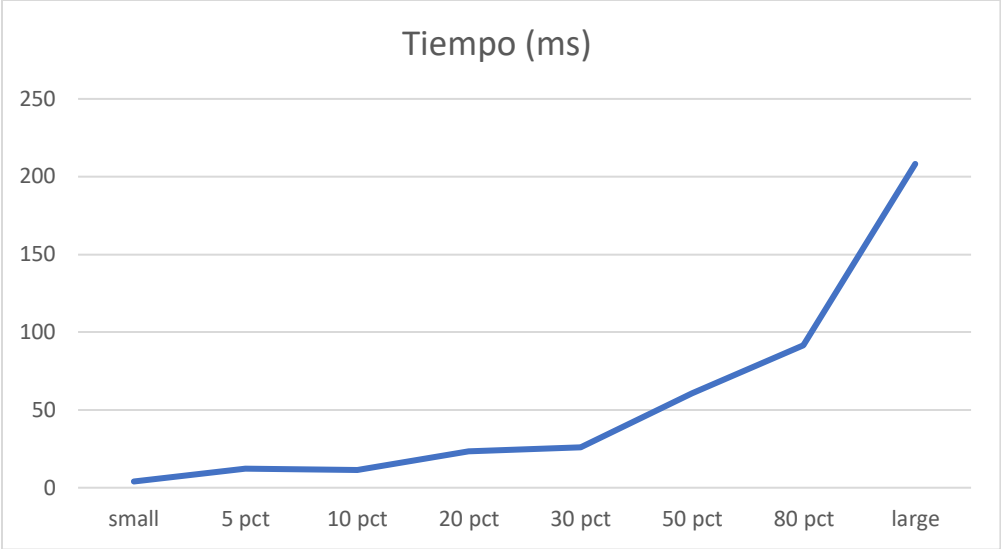
Muestra	Salida	Tiempo (ms)
small	El total de partidos relevantes al torneo, el total de países involucrados en el torneo, el total de ciudades donde se disputan los partidos del torneo, el total de partidos definidos por cobros de punto penal y la lista de cada partido con su información.	5,07
5 pct		15,6
10 pct		19,9
20 pct		55
30 pct		76,4
50 pct		1,54E+02
80 pct		1,89E+02
large		2,10E+02

Maquina 3:

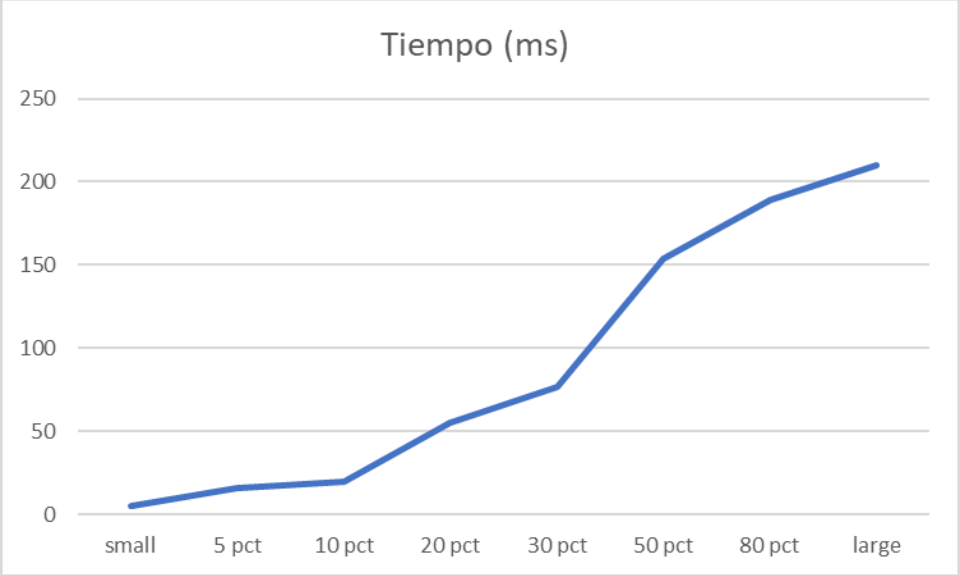
Muestra	Salida	Tiempo (ms)
small	El total de partidos relevantes al torneo, el total de países involucrados en el torneo, el total de ciudades donde se disputan los partidos del torneo, el total de partidos definidos por cobros de punto penal y la lista de cada partido con su información.	3,95
5 pct		12,3
10 pct		17,8
20 pct		45
30 pct		68,2
50 pct		132,7
80 pct		176,4
large		223

Graficas

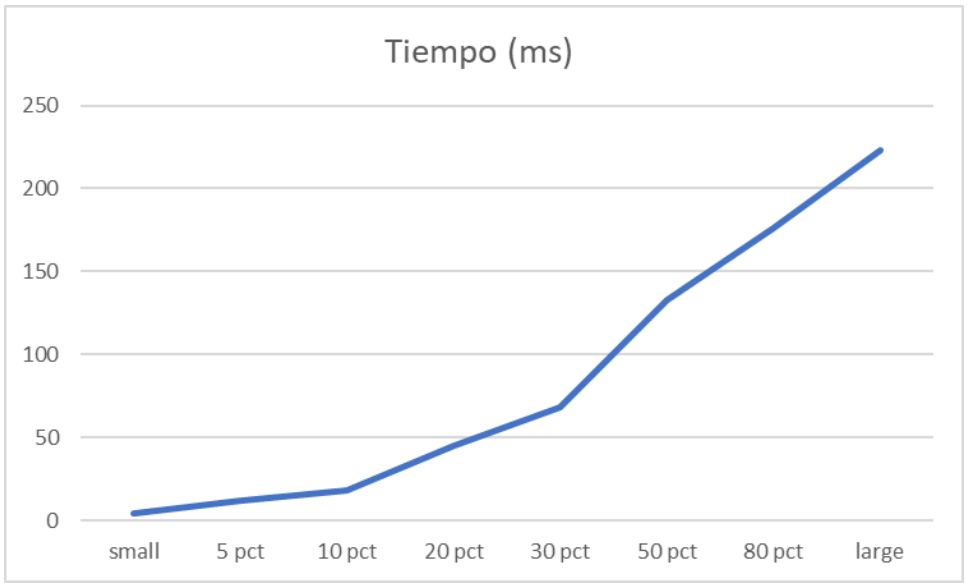
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

La complejidad es O(n^2), dado que por cada interacion del for que cumpla con la condición de rango se ejecutara las comparaciones necesarias para crear las variables necesarias y agregar al final del Arraylist y lo mismo pasa con el segundo for, además estos dos ciclos son independientes el uno del otro por lo que sus complejidades se multiplican. En las gráficas es posible apreciar un comportamiento similar al cuadrático.

Requerimiento 5

Descripción

```python
def req_5(data, name, lim_inf, lim_sup):
    """
    Función que soluciona el requerimiento 5
    """
    anotaciones = 0
    torneos = lt.newList("ARRAY_LIST")
    penales = 0
    autogoles = 0
    rta = lt.newList("ARRAY_LIST")
    current_date = ((dt.strptime(lim_sup,"%Y-%m-%d")).date()).toordinal()
    top_date = ((dt.strptime(lim_inf,"%Y-%m-%d")).date()).toordinal()
    i = 1
    while current_date >= top_date:
        elemento = lt.getElement(data["goal_scorers"], i)
        if elemento["scorer"] == name:
            rango= rango_by_fecha( elemento["date"], lim_inf, lim_sup)
            if rango:
                tournament, home_score, away_score = find_results(data["results"],elemento["date"], elemento["home_team"], elemento["away_team"])
                dict_scorer= {
                    "date":  elemento["date"],
                    "minute":  elemento["minute"],
                    "home_team":  elemento["home_team"],
                    "away_team":  elemento["away_team"],
                    "team":  elemento["team"],
                    "home_score": home_score,
                    "away_score": away_score,
                    "tournament": tournament,
                    "penalty":  elemento["penalty"],
                    "own_goal":  elemento["own_goal"]
                   }
                lt.addLast(rta, dict_scorer)

                if dict_scorer["penalty"] == True:
                    penales += 1
                if dict_scorer["own_goal"] == True:
                    autogoles += 1
                if lt.isPresent(torneos, dict_scorer["tournament"]) == 0:
                    lt.addLast(torneos, dict_scorer["tournament"])
                anotaciones += 1
        i += 1
        current_date = ((dt.strptime( elemento["date"],"%Y-%m-%d")).date()).toordinal()
        rta_final = sa.sort(rta, cmp_elementos_by_fecha_y_minuto)
    return anotaciones, lt.size(torneos), penales, autogoles, rta_final
```

```python
def rango_by_fecha(fecha, limite_inferior, limite_superior):
    """
    Devuelve verdadero (True) si la fecha esta dentro del rango estipulado.
    Args:
        Fecha: fecha a comparar
        limite_inferior: fecha en la que incia el rango.
        imite_superior: fecha en la que termina el rango.
    """
    rta = False
    fecha = ((dt.strptime(fecha,"%Y-%m-%d")).date()).toordinal()
    limite_inferior = ((dt.strptime(limite_inferior,"%Y-%m-%d")).date()).toordinal()
    limite_superior = ((dt.strptime(limite_superior,"%Y-%m-%d")).date()).toordinal()
    if (fecha >= limite_inferior) and (fecha <= limite_superior):
        rta = True
    return rta
```

```python
def find_results (results, date, home_team, away_team):
    """
    Busca los datos de results requeridos para completar el requerimiento
    Args:
        results: lista de resultados
        date, home_team, away_team: son los datos requeridos para comparar y determinar si
        los resultados corresponden a esa fecha
    """
    tournament = "Unknown"
    home_score = "Unknown"
    away_score = "Unknown"

    for r in lt.iterator(results):

        if (r["date"] == date) and (r["home_team"] == home_team) and (r["away_team"] == away_team):
            tournament = r["tournament"]
            home_score = r["home_score"]
            away_score = r["away_score"]

    return tournament, home_score, away_score
```

EL requerimiento 5 filtra las anotaciones de un jugador específico dentro de un rango de fechas especifico y retorna una lista de las anotaciones con una información específica la cual es: Fecha del partido, minuto en el que se marcó el gol, equipo local, equipo visitante, equipo del jugador, marcador del equipo local (goles del local), marcador del equipo visitante (goles del visitante), nombre del torneo donde se marcó el gol, tipo de anotación, si fue por falta desde el penal, Tipo de anotación, si fue autogol.

Entrada	Estructuras de datos del modelo, el nombre del jugador, fehca mínima y fecha máxima.
Salidas	Número de anotaciones obtenidas por el jugador, numero de torneos en que anoto el jugador, número de anotaciones obtenidas desde el punto penal, número total de autogoles cometidos y el listado de las anotaciones con su información
Implementado (Sí/No)	Si. Implementado por *Rodrigo Paz Londoño*

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear nuevos arreglos (newList)	O(1)
Obtener el elemento (getElement)	O(1)
Verificar si un elemento ya está (isPresent)	O(n)
Sumas (+=)	O(1)
Agregar un elemento al final de arreglo (addLast)	O(1)
Obtener el tamaño de un arreglo (Size)	O(1)
Declaración de variables	O(1)
Recorrer el primer ciclo (while)	O(n)
Verificar rango (rango_by_fecha)	O(1)

Recorrer el segundo ciclo (for)	O(n)
Comparaciones (==), (>=)	O(1)
Acceso a valores de un diccionario	O(1)
Ordenamiento Shell (sa.sort)	O(n log(n))
TOTAL	**$O(n^2)$**

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	10,7
5 pct	11
10 pct	11,7
20 pct	20,8
30 pct	37,3
50 pct	79,1
80 pct	204,1
large	426,04

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	**Tiempo (ms)**
small	4,42
5 pct	48,9
10 pct	68,2
20 pct	3,45E+02
30 pct	7,41E+02
50 pct	2,06E+03
80 pct	3,77E+03
large	5,51E+03

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**
small	10,12
5 pct	11,1
10 pct	13,6
20 pct	20,9
30 pct	46,7
50 pct	100,3
80 pct	204,8
large	486,3

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	**Salida**	**Tiempo (ms)**
small	Número de anotaciones obtenidas por el jugador, numero de torneos en que anoto el jugador, número de anotaciones obtenidas desde el punto penal, número total de autogoles cometidos y el listado de las anotaciones con su información	10,7
5 pct		11
10 pct		11,7
20 pct		20,8
30 pct		37,3
50 pct		79,1
80 pct		204,1
large		426,04

Maquina 2:

Muestra	**Salida**	**Tiempo (ms)**
small	Número de anotaciones obtenidas por el jugador, numero de torneos en que	4,42
5 pct		48,9
10 pct		68,2

20 pct	anoto el jugador, número de anotaciones obtenidas desde el punto penal, número total de autogoles cometidos y el listado de las anotaciones con su información	3,45E+02
30 pct		7,41E+02
50 pct		2,06E+03
80 pct		3,77E+03
large		5,51E+03

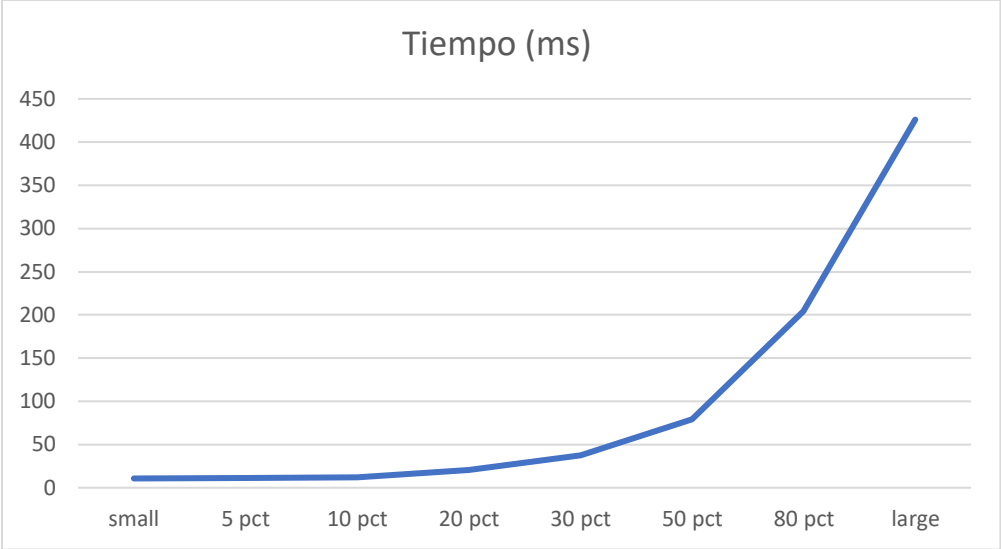
Maquina 3:

Muestra	**Salida**	**Tiempo (ms)**
small	Número de anotaciones obtenidas por el jugador, numero de torneos en que anoto el jugador, número de anotaciones obtenidas desde el punto penal, número total de autogoles cometidos y el listado de las anotaciones con su información	10,12
5 pct		11,1
10 pct		13,6
20 pct		20,9
30 pct		46,7
50 pct		100,3
80 pct		204,8
large		486,3

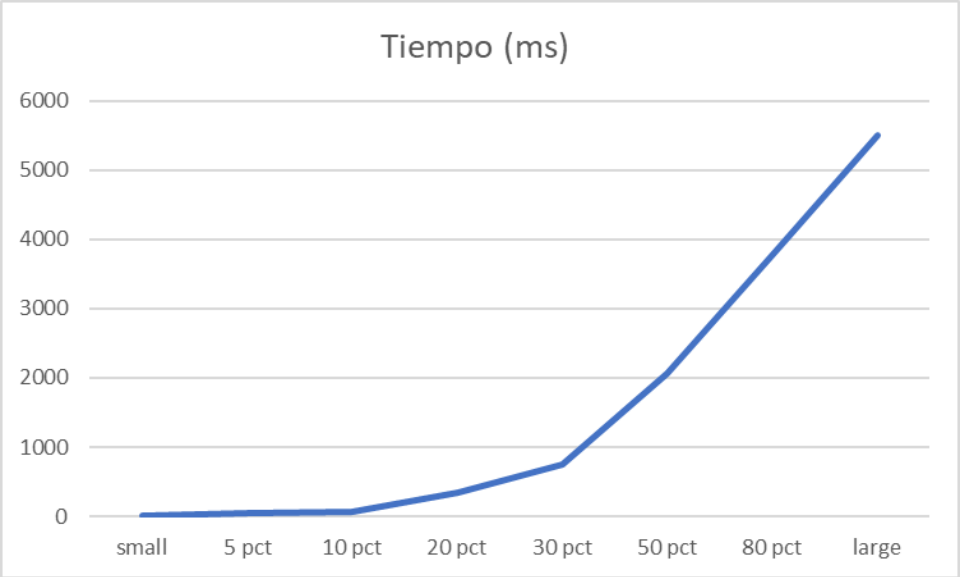
Graficas

Las gráficas con la representación de las pruebas realizadas.

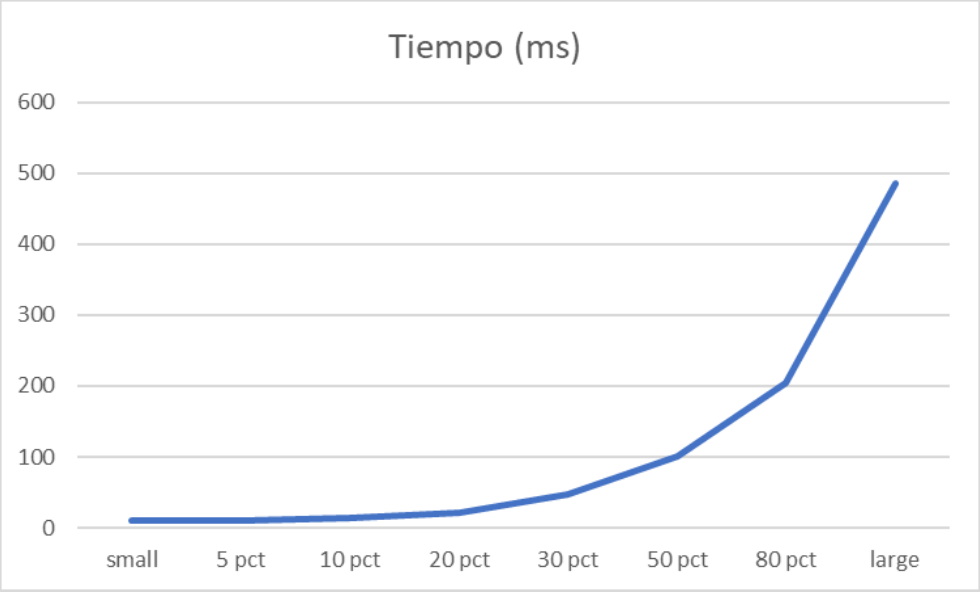
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

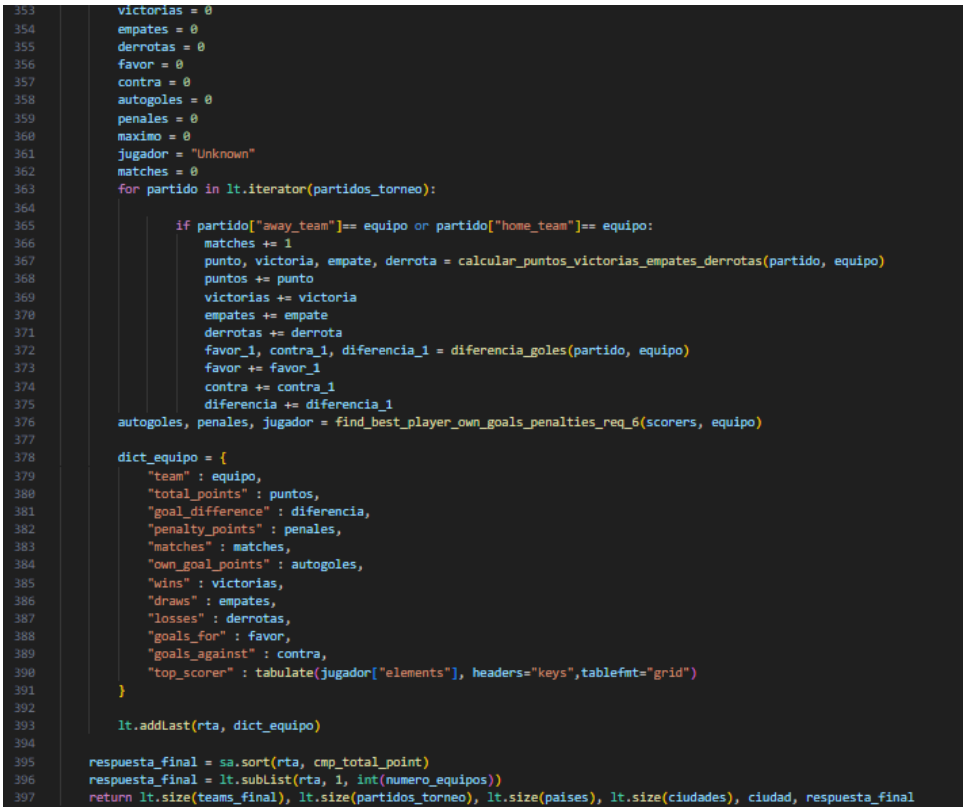
Todas las listas de retorno y auxiliares son de tipo ARRAY ya que son más eficientes para recorrerlas y acceder a sus elementos, lo cual facilita le implementación de ciclos.

La complejidad es $O(n^2)$, dado que por cada interacción de while que cumpla con la condición de rango se ejecutara el for de la función de find_results. Estos son los dos grandes ciclos que determinan el comportamiento de este requerimiento. En las gráficas es posible apreciar un comportamiento similar al cuadrático lo cual es congruente con lo planteado.

Requerimiento 6

Descripción

```
def req_6(data, numero_equipos, torneo, lim_inf, lim_sup):

    current_date = ((dt.strptime(lim_sup,"%Y-%m-%d")).date()).toordinal()
    top_date = ((dt.strptime(lim_inf,"%Y-%m-%d")).date()).toordinal()
    resultados = data["results"]
    partidos_torneo = lt.newList("ARRAY_LIST")
    paises = lt.newList("ARRAY_LIST")
    ciudades = lt.newList("ARRAY_LIST")
    ciudades_mas_partidos = {}
    maximo = 0
    scorers = data["goal_scorers"]
    scorers_2 = lt.newList("ARRAY_LIST")
    teams = lt.newList("ARRAY_LIST")
    rta = lt.newList("ARRAY_LIST")
    i = 1

    while current_date >= top_date:
        r = lt.getElement(data["results"], i)
        if r["tournament"] == torneo:
            fecha = r["date"]
            if rango_by_fecha(fecha, lim_inf, lim_sup) == True:
                lt.addLast(partidos_torneo, r)
                if lt.isPresent(paises, r["country"]) == 0:
                    lt.addLast(paises, r["country"])
                if lt.isPresent(ciudades, r["city"]) == 0:
                    lt.addLast(ciudades, r["city"])
                if r["city"]  not in ciudades_mas_partidos:
                    ciudades_mas_partidos[r["city"]] = 1
                else:
                    ciudades_mas_partidos[r["city"]] +=1

                if ciudades_mas_partidos[r["city"]] > maximo:
                    maximo = ciudades_mas_partidos[r["city"]]
                if ciudades_mas_partidos[r["city"]] == maximo:
                    ciudad = r["city"]

                lt.addLast(teams, r["home_team"])


        current_date = ((dt.strptime( r["date"],"%Y-%m-%d")).date()).toordinal()
        i += 1

    scorers = find_goal_scorers_2(data["goal_scorers"], partidos_torneo)
    teams_final = find_teams(scorers, teams)

    for equipo in lt.iterator(teams_final):
        puntos = 0
        diferencia = 0
```



```
        victorias = 0
        empates = 0
        derrotas = 0
        favor = 0
        contra = 0
        autogoles = 0
        penales = 0
        maximo = 0
        jugador = "Unknown"
        matches = 0
        for partido in lt.iterator(partidos_torneo):

                if partido["away_team"]== equipo or partido["home_team"]== equipo:
                    matches += 1
                    punto, victoria, empate, derrota = calcular_puntos_victorias_empates_derrotas(partido, equipo)
                    puntos += punto
                    victorias += victoria
                    empates += empate
                    derrotas += derrota
                    favor_1, contra_1, diferencia_1 = diferencia_goles(partido, equipo)
                    favor += favor_1
                    contra += contra_1
                    diferencia += diferencia_1
        autogoles, penales, jugador = find_best_player_own_goals_penalties_req_6(scorers, equipo)

        dict_equipo = {
            "team" : equipo,
            "total_points" : puntos,
            "goal_difference" : diferencia,
            "penalty_points" : penales,
            "matches" : matches,
            "own_goal_points" : autogoles,
            "wins" : victorias,
            "draws" : empates,
            "losses" : derrotas,
            "goals_for" : favor,
            "goals_against" : contra,
            "top_scorer" : tabulate(jugador["elements"], headers="keys",tablefmt="grid")
        }

        lt.addLast(rta, dict_equipo)

    respuesta_final = sa.sort(rta, cmp_total_point)
    respuesta_final = lt.subList(rta, 1, int(numero_equipos))
    return lt.size(teams_final), lt.size(partidos_torneo), lt.size(paises), lt.size(ciudades), ciudad, respuesta_final
```

El requerimiento 6 filtra los equipos de un torneo específico dentro de un rango de fechas especifico y retorna una lista de longitud específica (top) con una información específica la cual es: El nombre del equipo, el total de puntos obtenidos la diferencia de goles, el total de partidos disputados, el total de puntos obtenidos desde la línea penal, el total de puntos recibidos por autogol, el total de victorias, el total de empates, el total de derrotas, el total de goles obtenidos por sus jugadores, el total de goles recibidos por el equipo y el jugador con más anotaciones con su total de goles, su total de partidos y su promedio de tiempo (minutos) para anotar los goles, si algún dato no se encuentra se toma como “Unknown”.

Entrada	Estructuras de datos del modelo, el top de equipo, el nombre del torneo, fecha mínima y fecha máxima.
Salidas	El total de equipos involucrados en el torneo, el total de encuentros disputados en el periodo de tiempo, el total de países involucrados en el torneo, el total de ciudades involucradas en el torneo, el nombre de la ciudad donde más partidos se han disputado y el listado de equipos con su respectiva información
Implementado (Sí/No)	Si. Implementado por: Todos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear arreglos (newList) y declarar variables	O(1)

Operaciones básicas (suma y resta)	O(1)
Primer ciclo while	O(n)
Get element (ARRAY_LIST)	O(1)
Comparación y agregar al final en Array_list	O(1)
Comparaciones is present	O(n)
Llama a una función auxiliar	O(n)
LLama función auxiliar	$O(n^2$
Segundo ciclo For	O(n)
Declaración de variables y declaración de variables	O(1)
Tercer ciclo for	O(n)
Llama a función auxiliar	O(1)
Llama función auxiliar	O(1)
Llama función auxiliar	O(n)
Compara y declara variables	O(1)
Ordenamiento Shell (sa.sort)	O(n log(n))
TOTAL	**$O(n^2)$**

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	16,3
5 pct	22,2
10 pct	94,8
20 pct	305,03
30 pct	572,03
50 pct	1470,04
80 pct	2960,04
large	4120,04

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	Tiempo (ms)
small	3,64E+01
5 pct	3,86E+02
10 pct	1,59E+03
20 pct	6,04E+03
30 pct	1,19E+04
50 pct	2,65E+04
80 pct	5,13E+04
large	9,03E+05

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	Tiempo (ms)
small	18,75
5 pct	24,62
10 pct	100,78
20 pct	251
30 pct	802,47
50 pct	1601,7
80 pct	4568,2
large	7000,2

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	Salida	Tiempo (ms)
small	El total de equipos involucrados en el torneo, el total de encuentros disputados en el periodo de tiempo, el total de países involucrados en el torneo, el total de ciudades involucradas en el torneo, el nombre de la ciudad donde más partidos se han	16,3
5 pct		22,2
10 pct		94,8
20 pct		305,03
30 pct		572,03
50 pct		1470,04
80 pct		2960,04
large		4120,04

	disputado y el listado de equipos con su respectiva información	

Maquina 2:

Muestra	Salida	Tiempo (ms)
small	El total de equipos involucrados en el torneo, el total de encuentros disputados en el periodo de tiempo, el total de países involucrados en el torneo, el total de ciudades involucradas en el torneo, el nombre de la ciudad donde más partidos se han disputado y el listado de equipos con su respectiva información	3,64E+01
5 pct		3,86E+02
10 pct		1,59E+03
20 pct		6,04E+03
30 pct		1,19E+04
50 pct		2,65E+04
80 pct		5,13E+04
large		6,86E+04

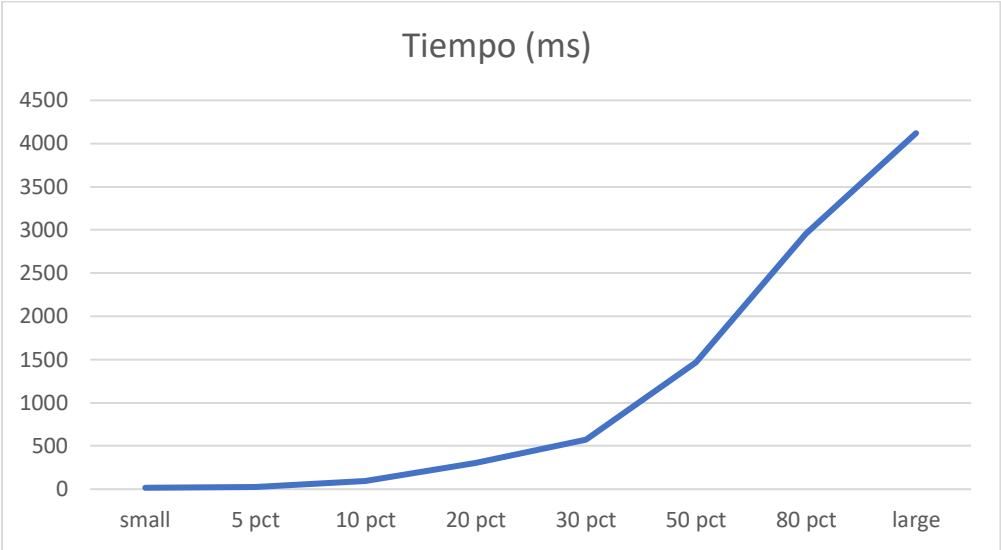
Maquina 3:

Muestra	Salida	Tiempo (ms)
small	El total de equipos involucrados en el torneo, el total de encuentros disputados en el periodo de tiempo, el total de países involucrados en el torneo, el total de ciudades involucradas en el torneo, el nombre de la ciudad donde más partidos se han disputado y el listado de equipos con su respectiva información	18,75
5 pct		24,62
10 pct		100,78
20 pct		251
30 pct		802,47
50 pct		1601,7
80 pct		4568,2
large		7000,2

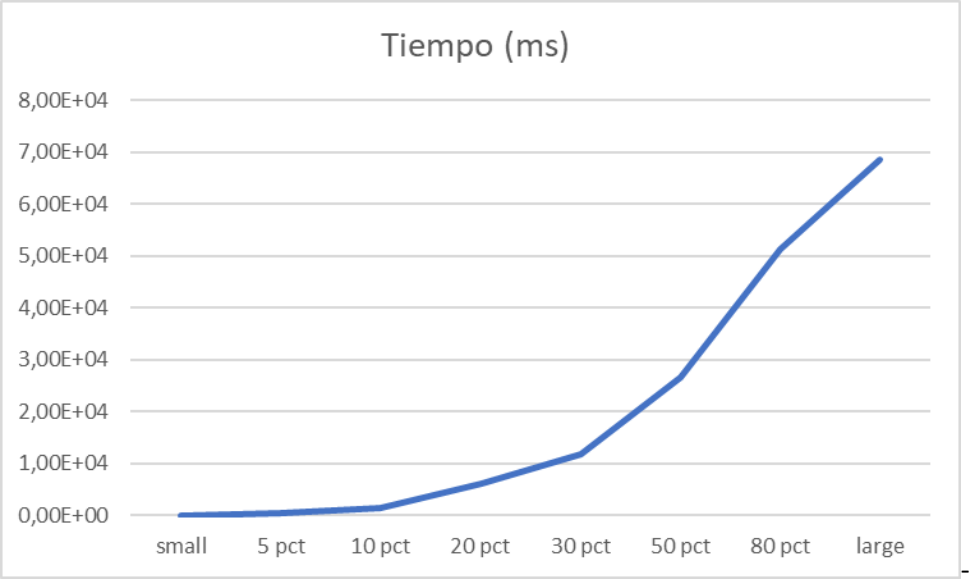
Graficas

Las gráficas con la representación de las pruebas realizadas.

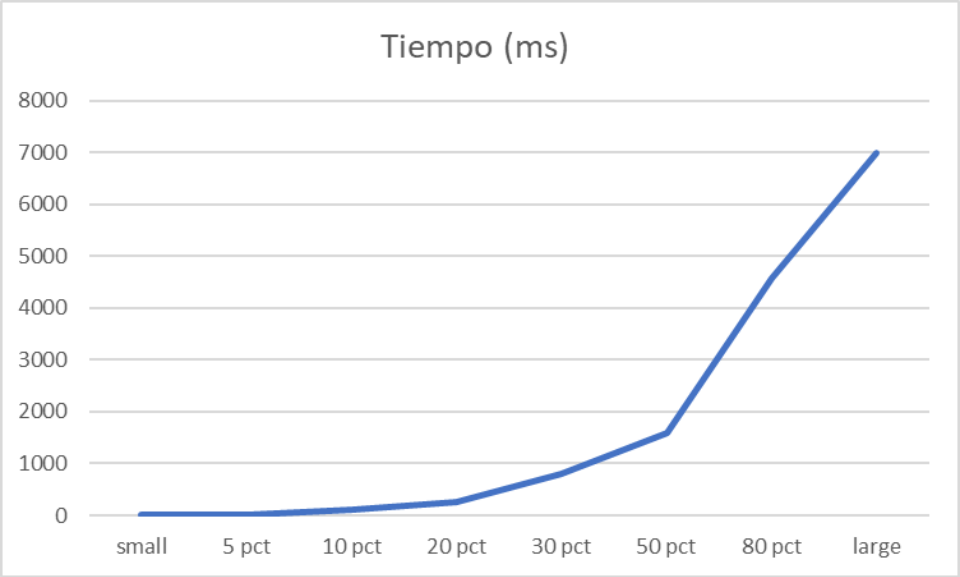
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

La graficas muestra un comportamiento cuadrático, dado que se tienen distintos ciclos dobles independientes. Además, a medida que la función avanza la cantidad de elementos se reduce, lo cual resulta en ciclos menos largos.

Requerimiento 7

```python
def req_7(data, top_scorers, inicial_date, final_date):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    tournaments = data["results"]
    goals_players = data["goal_scorers"]
    range_ofical_matches = lt.newList("ARRAY_LIST")
    range_date_goals = lt.newList("ARRAY_LIST")

    for oficial in lt.iterator(tournaments):
        date = oficial["date"]
        if oficial["tournament"] != "Friendly" and rango_by_fecha(date, inicial_date, final_date) == True:
            lt.addLast(range_ofical_matches, oficial)

    for player in lt.iterator(goals_players):
        date2 = player["date"]
        if rango_by_fecha(date2, inicial_date, final_date) == True:
            lt.addLast(range_date_goals, player)

    info_completa_players = lt.newList("ARRAY_LIST")

    for match in lt.iterator(range_ofical_matches):
        for goles in lt.iterator(range_date_goals):
            resultado = "Unknown"
            if (match["date"] == goles["date"]) and (match["home_team"] == goles["home_team"]) and  (match["away_team"] =
                if goles["team"] == match["home_team"] and int(match["home_score"]) > int(match["away_score"]):
                    resultado = "Victoria"
                elif goles["team"] == match["home_team"] and int(match["home_score"]) == int(match["away_score"]):
                    resultado = "Empate"
                elif goles["team"] == match["home_team"] and int(match["home_score"]) < int(match["away_score"]):
                    resultado = "Derrota"
                elif goles["team"] == match["away_team"] and int(match["away_score"]) > int(match["home_score"]):
                    resultado = "Victoria"
                elif goles["team"] == match["away_team"] and int(match["home_score"]) == int(match["away_score"]):
                    resultado = "Empate"
                elif goles["team"] == match["away_team"] and int(match["home_score"]) < int(match["away_score"]):
                    resultado = "Derrota"
```

```
            info_completa_goles = {
                                    "scorer": goles["scorer"],
                                    "minute": goles["minute"],
                                    "penalty": goles["penalty"],
                                    "own_goal": goles["own_goal"],
                                    "tournament": match["tournament"],
                                    "date": match["date"],
                                    "home_team": match["home_team"],
                                    "away_team": match["away_team"],
                                    "home_score": match["home_score"],
                                    "away_score": match["away_score"],
                                    "resultado": resultado,
                                   }

            lt.addLast(info_completa_players, info_completa_goles)

    partidos = set()
    torneos_tot = set()
    total_anota = set()
    total_goles_anotadores = lt.size(info_completa_players)
    total_penaltys = 0
    total_autogoles = 0

    for infor in lt.iterator(info_completa_players):
        total_anota.add(infor["scorer"])
        partidos.add((infor["date"], infor["home_team"], infor["away_team"]))
        torneos_tot.add(infor["tournament"])
        if infor["penalty"] == "True":
            total_penaltys += 1
        if infor["own_goal"] == "True":
            total_autogoles += 1

    total_partidos = len(partidos)
    total_torneos = len(torneos_tot)
    total_anotadores = len(total_anota)

    sorted_info_completa_players = sa.sort(info_completa_players, cmp_elementos_by_fecha_y_minuto2)
    lista_jugadores = lt.newList("ARRAY_LIST")
    dict_jugadores = {}
```

```
i = 1
while i <= lt.size(sorted_info_completa_players):
    element = lt.getElement(sorted_info_completa_players, i)
    jug = element["scorer"]

    if jug in dict_jugadores:
        lt.addLast(dict_jugadores[jug], element)
    if jug not in dict_jugadores:
        list_jug = lt.newList("ARRAY_LIST")
        lt.addLast(list_jug, element)
        dict_jugadores[jug] = list_jug
    i += 1

for jug, goles_jug in dict_jugadores.items():
    lt.addLast(lista_jugadores, goles_jug)

lista_info_goleadores = lt.newList("ARRAY_LIST")

for sub_list in lt.iterator(lista_jugadores):

    last_goals = {
                    "date": None,
                    "tournament": None,
                    "home_team": None,
                    "away_team": None,
                    "home_score": None,
                    "away_score": None,
                    "minute": None,
                    "penalty": None,
                    "own_goal": None
                  }

    info_player ={
                    "scorer": None,
                    "puntaje": 0,
                    "total_goals": 0,
                    "penalty_goals": 0,
                    "autogoles": 0,
                    "avg_time [min]": 0,
```

```
    torneos = set()
    total_minutes = 0

    for golesitos in lt.iterator(sub_list):

        info_player["total_goals"] += 1
        info_player["puntaje"] +=1

        if golesitos["scorer"] != "":
            info_player["scorer"] = golesitos["scorer"]

        if golesitos["tournament"] != "":
            torneos.add(golesitos["tournament"])

        if golesitos["minute"] != "":
            total_minutes += float(golesitos["minute"])

        if golesitos["penalty"] == "True":
            info_player["penalty_goals"] += 1
            info_player["puntaje"] +=1

        if golesitos["own_goal"] == "True":
            info_player["autogoles"] += 1
            info_player["puntaje"] -=1

        if golesitos["resultado"] == "Victoria":
            info_player["goals_wins"] += 1

        if golesitos["resultado"] == "Empate":
            info_player["goals_deal"] += 1

        if golesitos["resultado"] == "Derrota":
            info_player["goals_losses"] += 1
```

```
            last_goals["date"] = golesitos["date"]
            last_goals["tournament"] = golesitos["tournament"]
            last_goals["home_team"] = golesitos["home_team"]
            last_goals["away_team"] = golesitos["away_team"]
            last_goals["home_score"] = golesitos["home_score"]
            last_goals["away_score"] = golesitos["away_score"]
            last_goals["minute"] = golesitos["minute"]
            last_goals["penalty"] = golesitos["penalty"]
            last_goals["own_goal"] = golesitos["own_goal"]

        info_player["avg_time [min]"] = total_minutes / info_player["total_goals"]
        info_player["torneos_totales"] = len(torneos)
        info_player["last_goal"] = last_goals
        lt.addLast(lista_info_goleadores, info_player)

    sorted_info_goleadores = sa.sort(lista_info_goleadores, cmp_jugadores_by_puntaje)
    data_top_scorers = lt.newList("ARRAY_LIST")

    for scorer in lt.iterator(sorted_info_goleadores):
        lista = [scorer["last_goal"]]
        scorer["last_goal"] = tabulate(lista, headers = "keys", tablefmt="grid")

    i = 1
    while i <= lt.size(sorted_info_goleadores) and i <= int(top_scorers):
        element = lt.getElement(sorted_info_goleadores, i)
        lt.addLast(data_top_scorers, element)
        i += 1

    size_list = lt.size(data_top_scorers)

    return data_top_scorers, size_list, total_anotadores, total_partidos, total_torneos, total_goles_anotadores, total
```

Descripción

El requerimiento 7 filtra los mejores jugadores que participaron en partidos oficiales (no amistosos) dentro de un rango de fechas específico y retorna una lista de jugadores de longitud especifica con una información necesaria la cual es: El nombre del anotador, el puntaje que obtiene el jugador como anotador, el total de goles anotados, el total de goles anotados por penales, el total de autogoles

anotados, el tiempo promedio para anotar en minutos, el total de torneos en que anotó el jugador, el total de anotaciones obtenidos en una victoria, el total de anotaciones obtenidos en un empate, el total de anotaciones obtenidos en una derrota, ultimo gol anotado por el jugador con la siguiente información: ▪ Fecha del encuentro. ▪ Nombres de los equipos local y visitante. ▪ Puntaje de los equipos local y visitante. ▪ Minuto en que anotó el gol. ▪ Detalles técnicos del gol (si fue por falta desde el punto penal o autogol).

Entrada	Estructuras de datos del modelo, el número de jugadores a consultar, fecha mínima, fecha máxima.
Salidas	El total de anotadores que se encontraron en la consulta, el total de partidos o encuentros en que participaron los anotadores, el total de torneos donde participaron los anotadores en ese periodo, el total de anotaciones o goles obtenidos durante los partidos de ese periodo, el total de goles por penal obtenidos en ese periodo, el total de autogoles en que incurrieron los anotadores en ese periodo y la lista de jugadores con su respectiva información.
Implementado (Sí/No)	Si. Implementado por: Todos

Análisis de complejidad

Peor_caso

Pasos	**Complejidad**
Crear arreglos y declarar variables	O(1)
Primer ciclo for (recorre "tournaments")	O(n)
Segundo ciclo for (recorre "goals_players")	O(n)
Tercer ciclo for (recorre "range_ofical_matches")	O(n)
Dentro del tercer ciclo for: Cuarto ciclo for (recorre "range_date_goals")	O(n)* O(n)
Dentro del cuarto ciclo for: Crear y operar sobre diccionarios y listas	O(n)
Ordenar lista "info_completa_players" con Shell sort	O(n^2)
Quinto ciclo while (recorre "sorted_info_completa_players")	O(n)
Sexto ciclo for (recorre "lista_jugadores")	O(n)
Dentro del sexto ciclo for: Séptimo ciclo for (recorre sub-listas) O(m)	O(n)* O(n)
Octavo ciclo for (recorre "sorted_info_goleadores")	O(n)
Noveno ciclo while (recorre "sorted_info_goleadores")	O(n)
Comparaciones y asignaciones	O(1)
Agregar elementos a listas	O(1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	172.6
5 pct	986.04
10 pct	1053,5
20 pct	42700,03
30 pct	62500,03
50 pct	105000,04
80 pct	306400,04
large	546000,04

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	**Tiempo (ms)**
small	2,34E+02
5 pct	7,39E+02
10 pct	1,70E+03
20 pct	5,86E+03
30 pct	2,03E+04
50 pct	3,99E+04
80 pct	8,27E+04
large	1,44E+05

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**

small	128,7
5 pct	470,02
10 pct	1023,03
20 pct	6356,5
30 pct	12356,5
50 pct	31638.08
80 pct	786002.3
large	146050,06

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	Salida	Tiempo (ms)
small	El total de anotadores que se encontraron en la consulta, el total de partidos o encuentros en que participaron los anotadores, el total de torneos donde participaron los anotadores en ese periodo, el total de anotaciones o goles obtenidos durante los partidos de ese periodo, el total de goles por penal obtenidos en ese periodo, el total de autogoles en que incurrieron los anotadores en ese periodo y la lista de jugadores con su respectiva información.	68,2
5 pct		4470,02
10 pct		1023,03
20 pct		34300,03
30 pct		62500,03
50 pct		105000,04
80 pct		306400,04
large		546000,04

Maquina 2:

Muestra	Salida	Tiempo (ms)
small	El total de anotadores que se encontraron en la	2,34E+02
5 pct		7,39E+02

10 pct	consulta, el total de partidos o encuentros en que participaron los anotadores, el total de torneos donde participaron los anotadores en ese periodo, el total de anotaciones o goles obtenidos durante los partidos de ese periodo, el total de goles por penal obtenidos en ese periodo, el total de autogoles en que incurrieron los anotadores en ese periodo y la lista de jugadores con su respectiva información.	1,70E+03
20 pct		5,86E+03
30 pct		2,03E+04
50 pct		3,99E+04
80 pct		8,27E+04
large		1,44E+05

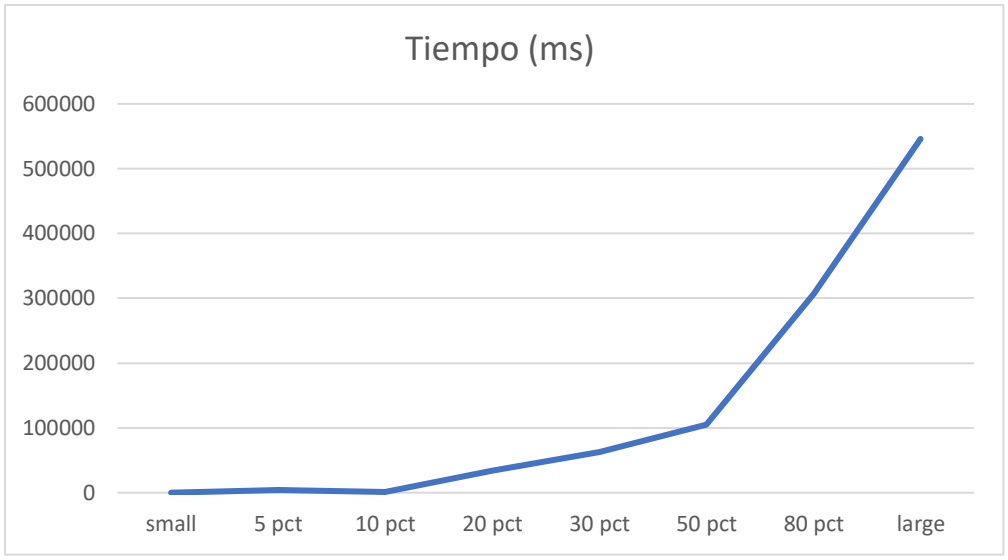
Maquina 3:

Muestra	**Salida**	**Tiempo (ms)**
small	El total de anotadores que se encontraron en la consulta, el total de partidos o encuentros en que participaron los anotadores, el total de torneos donde participaron los anotadores en ese periodo, el total de anotaciones o goles obtenidos durante los partidos de ese periodo, el total de goles por penal obtenidos en ese periodo, el total de autogoles en que incurrieron los anotadores en ese periodo y la lista de jugadores con su respectiva información.	128,7
5 pct		470,02
10 pct		1023,03
20 pct		6356,5
30 pct		12356,5
50 pct		31638.08
80 pct		786002.3
large		146050,06

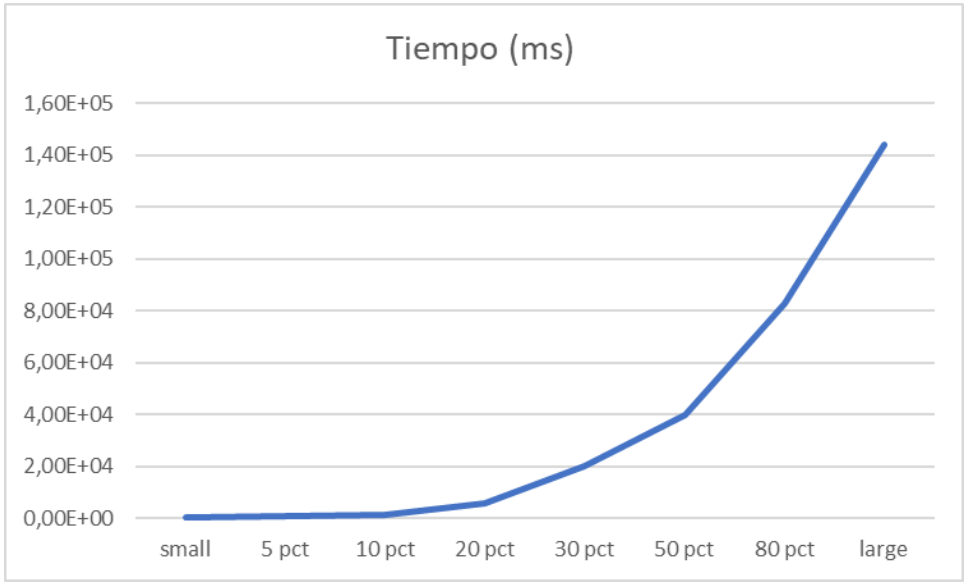
Graficas

Las gráficas con la representación de las pruebas realizadas.

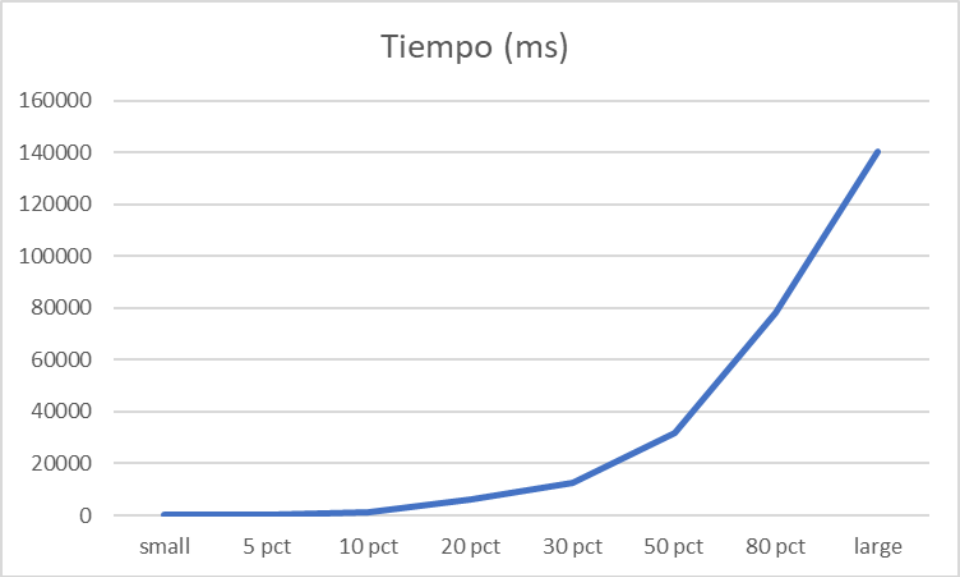
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

El peor caso de la complejidad algoritmica de la funcion seria O(n^2) acompañado de constantes muy grandes, debido a que, en el análisis de complejidad ordenar una lista mediante shell sort y los ciclos for anidados en los que el código itera sobre los datos tienen el mayor peso.

La complejidad dominante se encuentra en el ciclo "Tercer ciclo for", que tiene una complejidad de O(n), y dentro de él, el "Cuarto ciclo for", que tiene una complejidad de O(n). Y dentro de esta se hace un gran cambio de variables y llaves que hacen que su complejidad sea O(n^2) acompañado de una constante muy grande. (puede llegar a un valor mayor a O(n^2) pero sin llegar a ser O(n^3))

En esta funcion todas las listas creadas son tipo ARRAY_LIST, debido a la complejidad espacial y al uso de funciones como lt.getElement o lt.addLast. ARRAY_LIST tiene menor complejidad espacial que SINGLE_LINKED y en el caso de getElement tiene menor complejidad temporal. Como las listas se editan mucho en distintas posiciones, la mejor opcion es ARRAY_LIST.

En esta función se hace el uso de shell sort y no se compara con el resto de las funciones de ordenamiento, debido al volumen de información que se maneja.

- Shell sort ordena mejor que insertion y selection para cualquier tamaño de datos
- Merge o quick sort ordenan mejor que shell, sin embargo, esto aplica únicamente cuando se maneja un volumen de informacion muy alto.

Como en este caso se maneja poca informacion (50.000 datos) la mejor opcion es shell sort

Requerimiento 8

Descripción

```
def req_8(data, team1, team2, lim_inf, lim_sup):
    """
    Función que soluciona el requerimiento 8
    """
    current_date = ((dt.strptime(lim_sup,"%Y-%m-%d")).date()).toordinal()
    top_date = ((dt.strptime(lim_inf,"%Y-%m-%d")).date()).toordinal()
    partidos_1 = lt.newList("ARRAY_LIST")
    partidos_2 = lt.newList("ARRAY_LIST")
    ultimo_1 = lt.newList("ARRAY_LIST")
    ultimo_2 = lt.newList("ARRAY_LIST")
    años_1 = lt.newList("ARRAY_LIST")
    años_2 = lt.newList("ARRAY_LIST")
    local_1 = 0
    local_2 = 0
    visitante_1 = 0
    visitante_2 = 0
    victorias_1 = 0
    derrotas_1 = 0
    empates = 0
    victorias_2 = 0
    derrotas_2= 0
    encuentros = lt.newList("ARRAY_LIST")
    ultimo_partido_1_2 = lt.newList("ARRAY_LIST")


    i = 1
    while current_date >= top_date:

        elemento = lt.getElement(data["results"], i)
        if elemento["tournament"] != "Friendly":
            home_team1 = elemento["home_team"] == team1
            home_team2 = elemento["home_team"] == team2
            away_team1 = elemento["away_team"] == team1
            away_team2 = elemento["away_team"] == team2

            rango= rango_by_fecha(elemento["date"], lim_inf, lim_sup)

            if (home_team1 or away_team1) and rango:
                if home_team1:
                    local_1 += 1
                else:
                    visitante_1 += 1

                lt.addLast(partidos_1, elemento)

                año = (elemento["date"].split("-"))[0]
                if lt.isPresent(años_1, año) == 0:
                    lt.addLast(años_1, año)


            if (home_team2 or away_team2) and rango:
                if home_team2:
                    local_2 += 1
                else:
                    visitante_2 += 1

                lt.addLast(partidos_2, elemento)

                año = (elemento["date"].split("-"))[0]
                if lt.isPresent(años_2, año) == 0:
                    lt.addLast(años_2, año)

            if home_team1 and away_team2:

                if int(elemento["home_score"]) > int(elemento["away_score"]):
                    victorias_1 += 1
                    derrotas_2 += 1

                elif int(elemento["home_score"]) < int(elemento["away_score"]):
                    victorias_2 += 1
                    derrotas_1 += 1

                else:
                    empates += 1

                lt.addLast(encuentros, elemento)

            elif home_team2 and away_team1:

                if int(elemento["home_score"]) > int(elemento["away_score"]):
```

```
            if int(elemento["home_score"]) > int(elemento["away_score"]):
                victorias_2 += 1
                derrotas_1 += 1

            elif int(elemento["home_score"]) < int(elemento["away_score"]):
                victorias_1 += 1
                derrotas_2 += 1

            else:
                empates += 1

            lt.addLast(encuentros, elemento)

        current_date = ((dt.strptime(elemento["date"],"%Y-%m-%d")).date()).toordinal()
        i += 1

    ultimo_partido1 = lt.firstElement(partidos_1)
    del ultimo_partido1["neutral"]
    lt.addFirst(ultimo_1, ultimo_partido1)
    years_1 = int(lt.firstElement(años_1)) - int(lt.lastElement(años_1))
    oldest_match_1 = lt.lastElement(partidos_1)["date"]

    ultimo_partido2 = lt.firstElement(partidos_2)
    del ultimo_partido2["neutral"]
    lt.addFirst(ultimo_2, ultimo_partido2)
    years_2 = int(lt.firstElement(años_2)) - int(lt.lastElement(años_2))
    oldest_match_2 = lt.lastElement(partidos_1)["date"]

    ultimo_1_2 = lt.firstElement(encuentros)
    if len(ultimo_1_2)>0:
        del ultimo_1_2["neutral"]
    lt.addFirst(ultimo_partido_1_2, ultimo_1_2)
    anotaciones_ultimo = find_goal_scorers(data["goal_scorers"], ultimo_1_2)


    estadisticas_1 = estadisticas_anuales(partidos_1, años_1, team1, data["goal_scorers"])
    estadisticas_2 = estadisticas_anuales(partidos_2, años_2, team2, data["goal_scorers"])

    return years_1, lt.size(partidos_1), local_1, visitante_1, oldest_match_1, ultimo_1, estadisticas_1, years_2, lt.size(partidos_2), local_2, visitante_2, oldest_match_2, ultimo_2, estadisticas_2, lt.size(encuentros), victorias_1, derrotas_1, victorias_2, derrotas_2, empates, ultimo_partido_1_2, anotaciones_ultimo
```

```
def estadisticas_anuales(partidos, años, team, goal_scorers):

    estadisticas = lt.newList("ARRAY_LIST")

    for año in lt.iterator(años):
        partidos_año = 0
        puntos = 0
        diferencia = 0
        victorias = 0
        empates = 0
        derrotas = 0
        favor = 0
        contra = 0
        for partido in lt.iterator(partidos):
            if año in partido["date"]:
                partidos_año += 1
                punto, victoria, empate, derrota = calcular_puntos_victorias_empates_derrotas(partido, team)
                puntos += punto
                victorias += victoria
                empates += empate
                derrotas += derrota
                favor_1, contra_1, diferencia_1 = diferencia_goles(partido, team)
                favor += favor_1
                contra += contra_1
                diferencia += diferencia_1
        autogoles, penales, jugador = find_best_player_own_goals_penalties(goal_scorers, team, año)

        dict_año = {
            "year" : año,
            "matches" : partidos_año,
            "total_points" : puntos,
            "goal_difference" : diferencia,
            "penalties" : penales,
            "own_goal" : autogoles,
            "wins" : victorias,
            "draws" : empates,
            "losses" : derrotas,
            "goals_for" : favor,
            "goals_against" : contra,
            "top_scorer" : tabulate(jugador["elements"], headers="keys",tablefmt="grid")
        }
        lt.addLast(estadisticas, dict_año)

    return estadisticas

def calcular_puntos_victorias_empates_derrotas(partido, team):
    """Funcion que calcula las victorias, derrotas o empates de
    un equipo, a la vez que calcula sus puntos por cada una de estas"""
    puntos = 0
    victoria = 0
    derrota = 0
    empate = 0

    if ((int(partido["away_score"]) > int(partido["home_score"])) and partido["away_team"] == team) or ((int(partido["home_score"]) > int(partido["away_score"])) and partido["home_team"] == team):
        puntos = 3
        victoria = 1
    elif int(partido["away_score"]) == int(partido["home_score"]):
        puntos = 1
        empate = 1
    else:
        derrota = 1
    return puntos, victoria, empate, derrota

def diferencia_goles (partido, team):
    """Funcion que calcula los goles a favor y en contra de un equipo,
    con los cuales cacula la diferencia"""

    if partido["home_team"]== team:
        favor = int(partido["home_score"])
        contra = int(partido["away_score"])
    else:
        favor = int(partido["away_score"])
        contra = int(partido["home_score"])
    diferencia = favor - contra
    return favor, contra, diferencia
```

```python
def find_best_player_own_goals_penalties(goal_scorers, team, año):

    autogoles = 0
    penales = 0
    goleadores = {}
    rta = lt.newList("ARRAY_LIST")

    for jugador in lt.iterator(goal_scorers):
        if(año in jugador["date"]) and (jugador["away_team"] == team or jugador["home_team"] == team):

            if jugador["own_goal"] == "True" and jugador["team"] == team:
                autogoles += 1
            if jugador["penalty"] == "True"  and jugador["team"] == team:
                penales += 1

            if jugador["own_goal"] != "True":

                if jugador["scorer"] not in goleadores:
                    goleadores[jugador["scorer"]] = {}
                    goleadores[jugador["scorer"]]["goles"] = 1
                    goleadores[jugador["scorer"]]["partidos"] = [jugador["date"]]
                    goleadores[jugador["scorer"]]["minutos"] = float(jugador["minute"])

                else:
                    goleadores[jugador["scorer"]]["goles"] += 1
                    if jugador["date"] not in goleadores[jugador["scorer"]]["partidos"]:
                        goleadores[jugador["scorer"]]["partidos"].append(jugador["date"])
                    goleadores[jugador["scorer"]]["minutos"] += float(jugador["minute"])

    if len(goleadores) > 1:
        mayor = -1
        for scorer, data in goleadores.items():
            if data["goles"] > mayor:
                mayor = data["goles"]
                top = scorer


        for scorer, data in goleadores.items():
            if (data["goles"] == mayor) and (scorer == top):
                promedio = round( data["minutos"]/ data["goles"])

                top_scorer = {"scorer": scorer,
                              "goals" : data["goles"],
                              "matches" : len(data["partidos"]),
                              "avg_time" : promedio
                              }
                lt.addFirst(rta, top_scorer)

    else:
        for scorer, data in goleadores.items():
                promedio = round( data["minutos"]/ data["goles"])

                top_scorer = {"scorer": scorer,
                              "goals" : data["goles"],
                              "matches" : len(data["partidos"]),
                              "avg_time" : promedio
                              }
                lt.addFirst(rta, top_scorer)

    return autogoles, penales, rta
```

El requerimiento 8 compara el desempeño de dos equipos anualmente dentro de un rango de fechas especifico y retorna una lista del partido más reciente de los dos equipos con su información específica, una lista para cada equipo con sus estadísticas anuales, una lista del último partido entre los dos equipos y una lista de anotaciones del último partido entre los dos equipos.

Entrada	Estructuras de datos del modelo, nombre del primer equipo, nombre del segundo equipo, fecha mínima, fecha máxima
Salidas	Los años que comprende el historial entre las fechas especificadas, el total de partidos disputados por el equipo, el total de partidos disputados por el equipo como local, el total de partidos disputados

	por el equipo como visitante, La fecha del último partido más antiguo, el total de partidos disputados entre los dos equipos, el total de victorias del primer equipo, el total de derrotas del primer equipo, el total de empates entre los dos equipos, el total de victorias del segundo equipo, el total de derrotas del segundo equipo y las listas anteriormente mencionadas
Implementado (Sí/No)	Si. Implementado por: Todos.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Obtener el elemento (getElement)	O(1)
Crear arreglos (newList)	O(1)
Declarar variables	O(1)
Operaciones básicas (suma y resta)	O(1)
Comparaciones (==), (>), (i=)	O(1)
Acceso a valores de diccionario	O(1)
Calcular diferencias de goles	O(1)
Calcular victorias, derrotas, empates y puntos	O(1)
Encontrar mejor jugador	O(n)
Primer ciclo (while)	O(n)
Calcular estadísticas	$O(n^2)$
Ordenamiento Shell (sa.sort)	O(n log(n))
TOTAL	**$O(n^2)$**

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel(R) Core(TM) i7- 10750H CPU @ 2.60GHz 2.59 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home Single Language – 64 bits

Entrada	**Tiempo (ms)**
small	36,2
5 pct	36,1
10 pct	19,9
20 pct	39,2
30 pct	56,1
50 pct	93,7

80 pct	182,03
large	353

Procesadores	**Intel(R) Core(TM) i5- 7200U CPU @2.50GHz 2.70 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10 Home – 64 bits

Entrada	**Tiempo (ms)**
small	37,7
5 pct	1,76E+02
10 pct	3,30E+02
20 pct	7,02E+02
30 pct	1,07E+03
50 pct	2,06E+03
80 pct	2,70E+03
large	3,37E+03

Procesadores	**Intel(R) Core(TM) i5- 1035G7 CPU @ 1.20GHz 1.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home - 64 bits

Entrada	**Tiempo (ms)**
small	37.75
5 pct	175.95
10 pct	330.50
20 pct	703.30
30 pct	1070.55
50 pct	2061.20
80 pct	2699.80
large	3371.10

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Maquina 1:

Muestra	**Salida**	**Tiempo (ms)**
small		36,2

5 pct	Los años que comprende el historial entre las fechas especificadas, el total de partidos disputados por el equipo, el total de partidos disputados por el equipo como local, el total de partidos disputados por el equipo como visitante, La fecha del último partido más antiguo, el total de partidos disputados entre los dos equipos, el total de victorias del primer equipo, el total de derrotas del primer equipo, el total de empates entre los dos equipos, el total de victorias del segundo equipo, el total de derrotas del segundo equipo y las listas anteriormente mencionadas	36,1
10 pct		19,9
20 pct		39,2
30 pct		56,1
50 pct		93,7
80 pct		182,03
large		353

Maquina 2:

Muestra	**Salida**	**Tiempo (ms)**
small	Los años que comprende el historial entre las fechas especificadas, el total de partidos disputados por el equipo, el total de partidos disputados por el equipo como local, el total de partidos disputados por el equipo como visitante, La fecha del último partido más antiguo, el total de partidos disputados entre los dos equipos, el total de	37,7
5 pct		1,76E+02
10 pct		3,30E+02
20 pct		7,02E+02
30 pct		1,07E+03
50 pct		2,06E+03
80 pct		2,70E+03
large		3,37E+03

	victorias del primer equipo, el total de derrotas del primer equipo, el total de empates entre los dos equipos, el total de victorias del segundo equipo, el total de derrotas del segundo equipo y las listas anteriormente mencionadas	

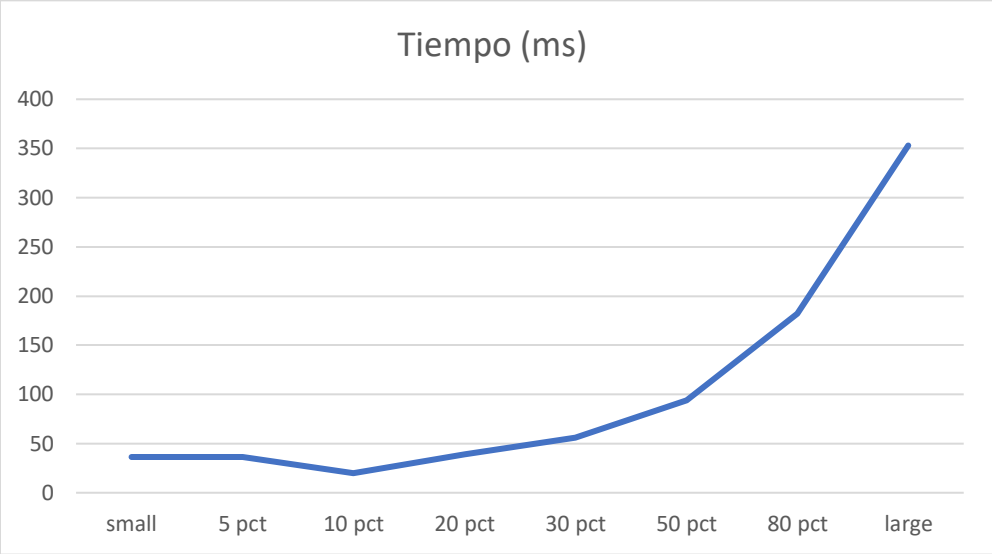
Maquina 3:

Muestra	Salida	Tiempo (ms)
small	Los años que comprende el historial entre las fechas especificadas, el total de partidos disputados por el equipo, el total de partidos disputados por el equipo como local, el total de partidos disputados por el equipo como visitante, La fecha del último partido más antiguo, el total de partidos disputados entre los dos equipos, el total de victorias del primer equipo, el total de derrotas del primer equipo, el total de empates entre los dos equipos, el total de victorias del segundo equipo, el total de derrotas del segundo equipo y las listas anteriormente mencionadas	37.75
5 pct		175.95
10 pct		330.50
20 pct		703.30
30 pct		1070.55
50 pct		2061.20
80 pct		2699.80
large		3371.10

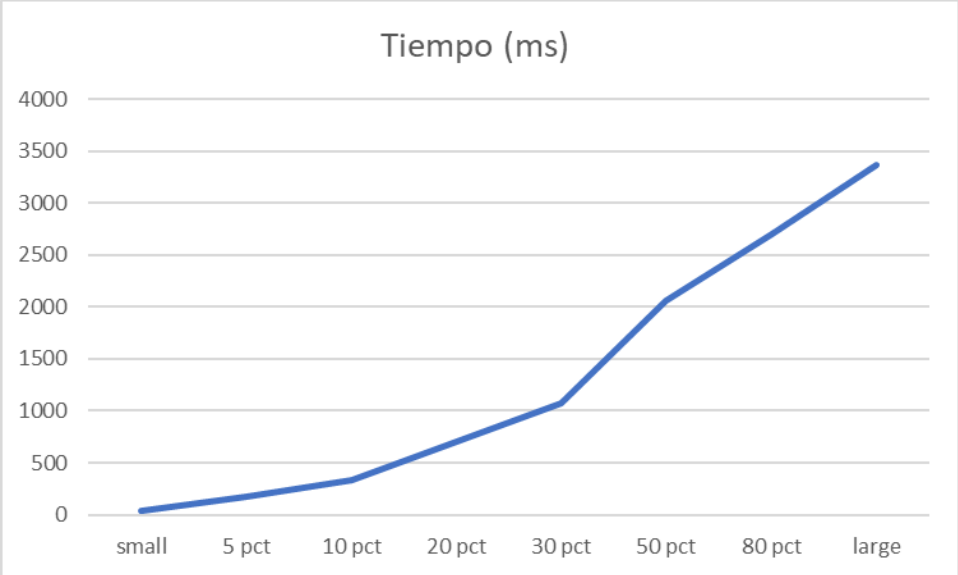
Graficas

Las gráficas con la representación de las pruebas realizadas.

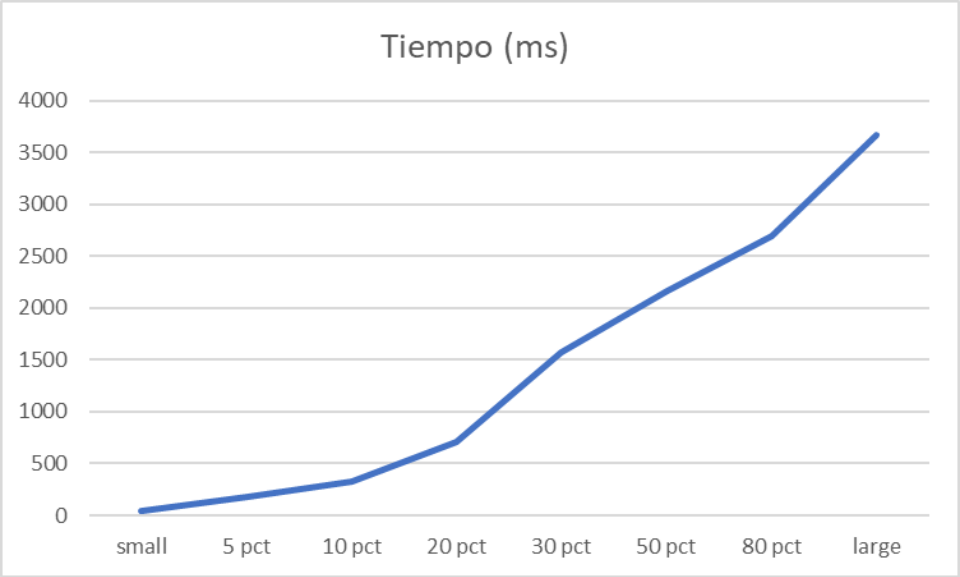
Maquina 1:



Maquina 2:



Maquina 3:



Análisis

- Inicialización de variables y listas: Operaciones de tiempo constante (O(1)).
- Todas las listas de retorno y auxiliares son de tipo ARRAY ya que son más eficientes para recorrerlas y acceder a sus elementos, lo cual facilita le implementación de ciclos.
- Bucle while que itera sobre los resultados (results), este bucle se ejecuta mientras current_date sea mayor o igual a top_date. La cantidad de iteraciones depende de las fechas en los resultados.
- Varias conversiones de fechas y asignaciones de variables, todas de tiempo constante (O(1)).
- Llamadas a lt.isPresent para buscar elementos en listas. La complejidad de búsqueda en una lista puede ser O(n), donde "n" es la cantidad de elementos en la lista a la que se hace la búsqueda.
- Llamadas a funciones como find_best_player_own_goals_penalties, estadisticas_anuales, calcular_puntos_victorias_empates_derrotas, y diferencia_goles. Estas funciones pueden tener una complejidad de tiempo de O(n) en el peor caso, donde "n" es la cantidad de elementos en la lista de resultados. Por otro lado, calcular estadísticas tiene un crecimiento cuadratico dado que se ejcutan dos ciclos, cada uno independiente del otro y esta es la función que determina la complejidad del req 8.

El requerimiento tiene un crecimiento cuadrático dado que el ciclo de estadísticas se ejecuta 2 for, además de que lo hace para dos países. Las gráficas muestran experimentalmente lo planteado ya que se evidencia un comportamiento cuadrático.

NOTA PARA TODOS: Todos los requerimientos se corrieron con shellsort dado que es un ordenamiento que no implica el uso de memoria adicional y si complejidad es O(nlog(n)), lo cual es sumamente efectivo para pequeñas cantidades de datos, como las estructurasa retornar de cada requerimirnto.