Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

Carlos Said Llain Rincon, 201920981, c.llain@uniandes.edu.co

Juan Esteban Triviño Nieves, 202315338, J.trivinon@uniandes.edu.co

Yohan Felipe Gaitan Carvajal 202312115 y.gaitan@uniandes.edu.co

Requerimiento 1

Descripción

```
def req_1(catalog,pais,condicion):

    partidos = catalog["Results"]
    home = catalog['Results_Team_H']
    away = catalog['Results_Team_A']
    entry = mp.get(partidos, pais)
    total_matches = me.getValue(entry)

    if condicion == "visitante" :
            entry = mp.get(away, pais)
            partidos_ = me.getValue(entry)

    elif condicion == "local" :
            entry = mp.get(home, pais)
            partidos_ = me.getValue(entry)

    elif condicion == "indiferente":
        partidos_ = {'partidos': None}
        partidos_['partidos'] = total_matches
    else:
        print("Error en la condicion")
    total_matches = mp.size(total_matches)
    total_teams = mp.size(partidos)
    return partidos_ , total_teams, total_matches
```

Entrada	Estructura de datos, país, condición.
Salidas	Un array que contiene los partidos que pertenezca a tanto al equipo que entra por parámetro, como la condición que también entra por

	parámetro.
Implementado (Sí/No)	Si. Implementado por Felipe Gaitán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conseguir el catalogo[“Results”]	O (1)
Conseguir la estructura de datos (“Results_Team_H”)	O (1)
Conseguir la estructura de datos (“Results_Team_A”)	O(1)
Conseguir la pareje llave-valor a ese equipo Mp.get()	O (1)
Conseguir los valores que perteneces a ese equipo me.getValue()	O(1)
Comprobar si la condición que entra por parametro	O (1)
Buscar en la estructura de datos (dependiendo si es local o visitante) los valores	O (1)
TOTAL	***O(1)***

Pruebas Realizadas

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

Procesadores	**Intel core i7-12700K**
Memoria RAM	32GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	0.080
5 pct	0.080
10 pct	0.081
20 pct	0.082
30 pct	0.083
50 pct	0.084
80 pct	0.085
large	0.085

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Al implementar el requerimiento 1 se obtiene un nuevo “Array_list” que contiene los partidos que cumpla tanto con la condición de equipo, como la condición si son visitante, locales o indiferentes, se entrega ordenado, ya que, desde la carga de datos, los datos están ordenados por fechas.

El comportamiento descrito en la complejidad concuerda con la de las gráficas, por que muestra un crecimiento lineal.

Requerimiento 2

Descripción

```python
def req_2(catalog,jugador):
    goleadores = catalog["Goalscorers"]
    entry = mp.get(goleadores, jugador)
    goles = me.getValue(entry)

    total_scorers = mp.size(goleadores)
    total_penalties = 0
    for gol in lt.iterator(goles):
        if gol['penalty'] == "True":
            total_penalties += 1

    return goles, total_scorers, total_penalties
```

Entrada	Estructura de datos, jugador
Salidas	Un mapa con los partidos en los que haya anotado el jugador también retorna los partidos posibles, y los penaltis que se haya realizado
Implementado (Sí/No)	Si. Implementado por Juan Triviño.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conseguir el catalogo[“Goalscorers”]	O (1)
Conseguir la pareje llave-valor al jugador Mp.get()	O (1)
Conseguir los valores que perteneces a ese jugador me.getValue()	O (1)
Sacar el tamaño mp.size()	O (1)
Crear una nueva variable	O (1)
Lt.iterator (goles) para sacar los goles del jugador	O(n)
Condiciones si penta es true	O(1)

TOTAL	***O(n)***

Pruebas Realizadas

Procesadores	**Intel® Core™ i7-1165G7 @2.80GHz**
Memoria RAM	15.6 GB
Sistema Operativo	Windows 11 Pro – 64 bits

Entrada	**Tiempo (ms)**
small	0.074
5 pct	0.076
10 pct	0.079
20 pct	0.081
30 pct	0.089
50 pct	0.090
80 pct	0.094
large	0.103

Graficas

Las gráficas con la representación de las pruebas realizadas.

- Máquina 1:

Análisis

Al implementar el requerimiento 2 se obtiene un Hash Map con todos los datos, de los goles obtenido por un jugador, ordenado por la fecha ya que viene ordenados desde la estructura de datos.
La complejidad concuerda con las gráficas obtenidas ya que la complejidad termina siendo lineal.

Requerimiento 3

Descripción

```python
def req_3(catalog,pais,inicial,final):
    """
    Función que soluciona el requerimiento 3
    """
    contadorTotal=0
    contadorAway=0
    contadorHome=0
    filtro=lt.newList("ARRAY_LIST")
    mapa=catalog["paises"]
    entry = mp.get(mapa, pais)
    partidos = me.getValue(entry)

    tamañoC=mp.size(catalog["Results"])



    for partido in lt.iterator(partidos):
     if comparardelta(partido["date"][0:4],inicial, final):
        if partido["home_team"] == pais:
            contadorHome+=1
            contadorTotal+=1
            lt.addLast(filtro,partido)
        if partido["away_team"]== pais:
            contadorAway+=1
            contadorTotal+=1
            lt.addLast(filtro,partido)



    return filtro,tamañoC,contadorTotal,contadorAway, contadorHome
```

Entrada	Estructura de datos, nombre del país, fecha inicial, fecha final
Salidas	Una lista con los partidos que disputo un equipo en el periodo de tiempo entrado por parámetro.
Implementado (Sí/No)	Si. Implementado por Juan Triviño

Pasos	**Complejidad**
Conseguir de la estructura de datos, las diferentes listas que vamos a iterar	O (1)
Crear una nueva Array List vacia con la función lt.newList	O (1)
Iterar *elemento* sobre la lista partidos	O (n)
Comprobar si ese partido está dentro del periodo de tiempo especifico .	O (1)
Comprobar si el equipo está en el equipo local o visitante	O (1)
Si cumple con las especificaciones, se agrega la lista, con la función lt.addLast, también se agrega los últimos parámetros.	O (1)
Se toma el size del catalog general.	O (1)
TOTAL	**O(n)**

Pruebas Realizadas

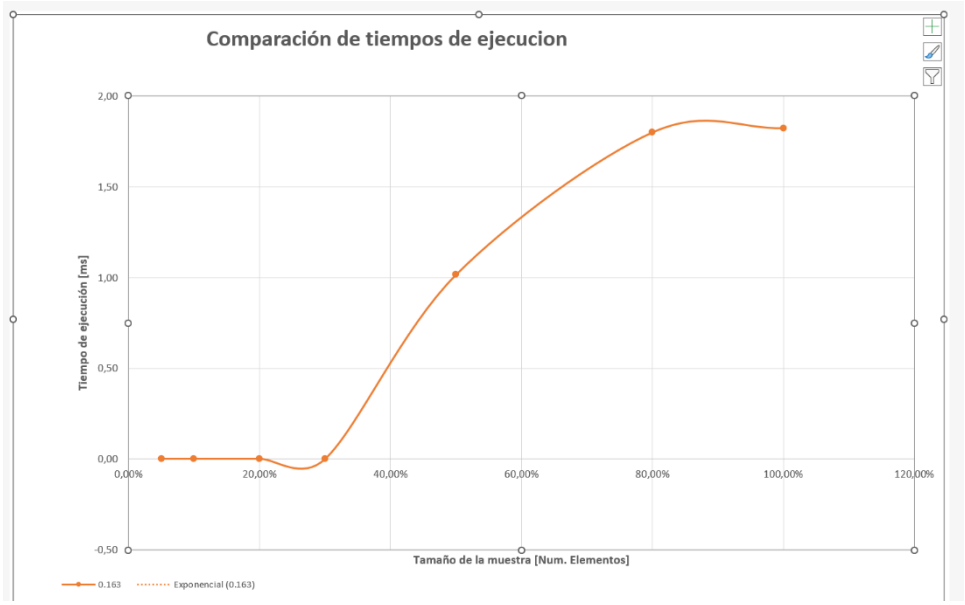
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz**
Memoria RAM	8.0 GB
Sistema Operativo	Windows 11 Pro

Entrada	**Tiempo (ms)**
small	0.163
5 pct	0.245
10 pct	0.424
20 pct	0.801
30 pct	0.912
50 pct	1,02
80 pct	1,80
large	2

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al implementar el requerimiento 3 obtenemos como resultado una lista de tipo Array_list con los datos que queremos obtener, la complejidad tiende a ser lineal, con algunos ajustes debido al equipo donde se realizan las pruebas.

Requerimiento 4

Descripción

```
535     #Se saca una sublista de acuerdo al rango proporcionado por el usuario
536     partidos_filtrados = lt.subList(partidos,fecha2, numelem)
537
538     #Se recorren los partidos para las estadisticas
539     for partido in lt.iterator(partidos_filtrados):
540         if lt.isPresent(paises, partido['country']) == 0:
541             lt.addLast(paises, partido['country'])
542
543         if lt.isPresent(ciudades, partido['city']) == 0:
544             lt.addLast(ciudades, partido['city'])
545
546         if lt.isPresent(penaltis, partido['penalty']) == 0 and partido['winner'] != 'penalty':
547             lt.addLast(penaltis, partido['penalty'])
548
549         if lt.isPresent(penaltis, partido['winner']) == 0 and partido['winner'] != 'unknown':
550             lt.addLast(penaltis, partido['winner'])
551
552     #Se transforman los datos para poder imprmir en pantalla
553     total_torneos = mp.size(torneos)
554     total_partidos = lt.size(partidos_filtrados)
555     paises = lt.size(paises)
556     ciudades = lt.size(ciudades)
557     penaltis = lt.size(penaltis)
558     return partidos_filtrados, total_torneos, total_partidos, paises, ciudades, penaltis
```

Entrada	Estructura de datos, fecha inicial, fecha final y el torneo
Salidas	Una lista con los partidos que estén en la delta de tiempo, y que pertenezca al torneo que entra por parámetro, el número total de torneos, el número total de partidos, el número total de paises, ciudades y penaltis del torneo en cuestión.
Implementado (Sí/No)	Si. Implementado por Felipe Gaitán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Se crean tres array_list vacias que van a funcionar de contadores para países, ciudades y penaltis respectivamente.	O (1)
Se obtiene el mapa de torneos de la estructura de datos.	O (1)
Se consiguen los partidos del torneo especificado por parámetro por medio de las llaves del mapa y las funciones mp.get y mp.getValue.	O (1)
Se realizan dos búsquedas binarias dentro de la lista de los partidos para encontrar la fecha que coincida con las proporcionadas por parámetro.	O (logN)
Comprobar si ese partido está dentro del periodo de tiempo especifico	O (1)
Condicional y asignación de variables	O (1)
Sublista de acuerdo al rango de fechas.	O (m)
Se itera esta última lista para rellenar los contadores.	O (m)
Se obtienen los size de los mapas o listas para los datos requeridos.	O (1)
TOTAL	**O(2m(logN))**

Pruebas Realizadas

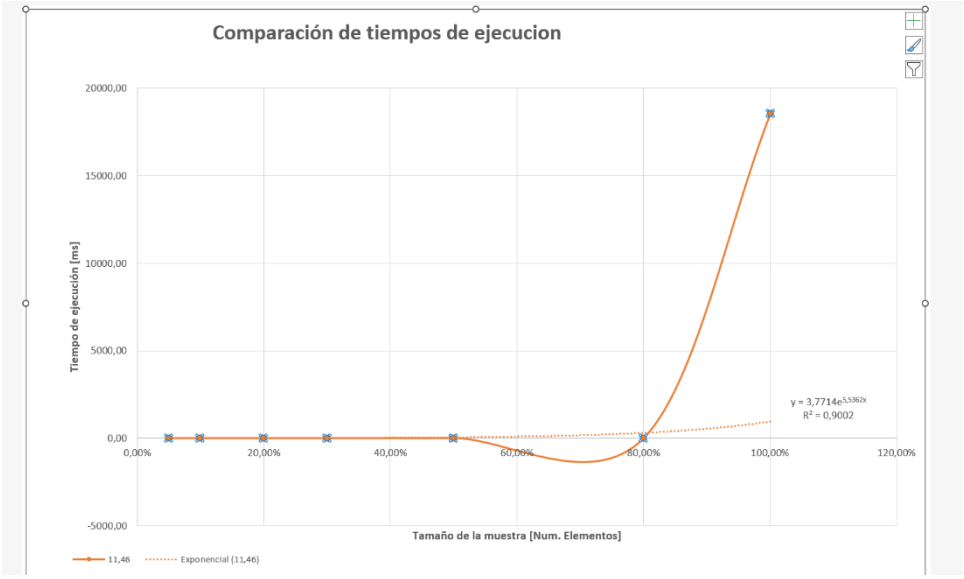
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Intel core i7-12700K**
Memoria RAM	32GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (ms)**
small	11,46
5 pct	11,90
10 pct	12,53
20 pct	13,11
30 pct	13,21
50 pct	16,14
80 pct	17,43
large	18,500

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al implementar el requerimiento 4 obtenemos como resultado un Array_list con el rango de fecha, la gráfica no coindice mucho con lo esperado, se observa una complejidad mucho más constante lo cual es beneficioso.

Requerimiento 5

```
def req_5(catalog, jugador, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    turnaments = lt.newList("ARRAY_LIST")
    penalties = lt.newList("ARRAY_LIST")
    autogoals = lt.newList("ARRAY_LIST")

    size = mp.size(catalog["Goalscorers"])

    jugadores = catalog["Goalscorers"]
    entry = mp.get(jugadores, jugador)
    jug = me.getValue(entry)

    fecha1 = binary_search_start_date(jug, fecha_inicial)
    fecha2 = binary_search_end_date(jug, fecha_final)

    num = fecha1 - fecha2

    if fecha2 == 0:
        fecha2 == 1

    matches = lt.subList(jug,fecha2, num)

    for partido in lt.iterator(matches):
        if partido["penalty"] == "True":
            lt.addLast(penalties, partido['penalty'])

        if partido["own_goal"] == "True":
            lt.addLast(autogoals, partido['own_goal'])

    s_penalties = lt.size(penalties)
    s_autogoals = lt.size(autogoals)

    return jug,size,s_penalties,s_autogoals
```

Descripción

Entrada	Estructura de datos, jugador, fecha de inicio y fecha final.
Salidas	Una array list con los partidos goles que realizo el jugador que entro por parámetro y en la delta de tiempo que entra por parámetro. También se retorna los diferentes datos que nos pide en el req
Implementado (Sí/No)	Si. Implementado por Carlos Said.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conseguir el catalogo[“Goalscorers”]	O (1)

Crear múltiples array_list vacías con lt.newList	O (1)
Conseguir la pareja llave valor de los datos que conicidad con el jugador	O (1)
Conseguir sola los datos de esa pareja llave valor	O (1)
Realizar la una búsqueda binaria por la primera fecha	O (log n)
Realizar la una búsqueda binaria por la segunda fecha	O (log n)
Creamos una sublista que abarque las posiciones según la fecha	O (n)
Itearmos matches con el lt.iterator	O (m)
Comprobamos diferentes variables, para agregarlos o no a los array_list previamentes creados	O (1)
Sacamos el size de los array_LIST	O (1)
Total	**O (n+log(n)+m)**

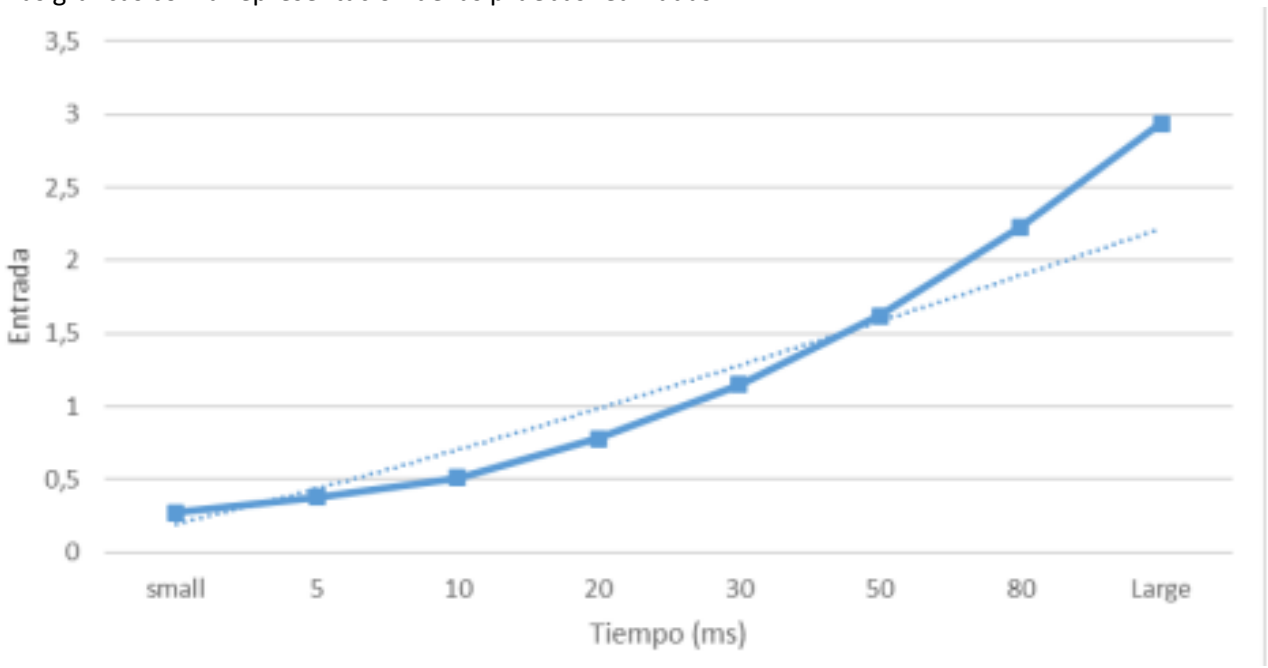
Pruebas Realizadas

Procesadores	**AMD Ryzen 5 5700x 3.4GHz GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Pro – 64 bits

Entrada	**Tiempo (ms)**
small	0.22
5 pct	0,38
10 pct	0,51
20 pct	0,78
30 pct	1,15
50 pct	1,62
80 pct	2,23
Large pct	2,94

Graficas

Las gráficas con la representación de las pruebas realizadas.



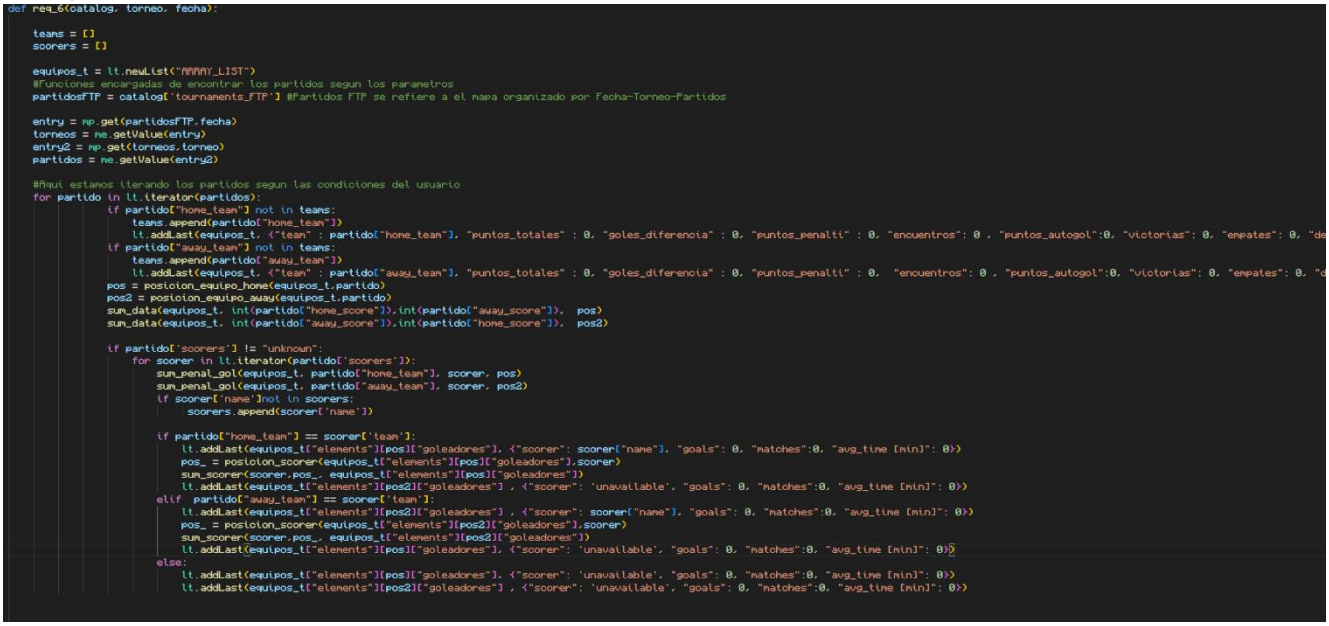
Análisis

Al implementar el requerimiento 5 obtenemos como resultado un Array_list de con los datos que se nos pide para el requerimientos, y gracias a la busqueda binaria que se implementos, llega a tener una complejidad baja.

Se retorna un Array_list que contiene los partidos jugados por el jugador que entra por parámetro y en la delta de tiempo que también entra por parámetro. Y retorna una lista ordena de esos partidos por las fechas

Lo descrito anteriormente es posible visualizarlo con las gráficas, puesto que esta muestra cómo se tiene un crecimiento de orden lineal, ya que al duplicar los datos de entrada los tiempos de ejecución no se duplican.

Requerimiento 6



```
def req_6(catalog, torneo, fecha):

    teams = []
    scorers = []

    equipos_t = lt.newList("ARRAY_LIST")
    #Funciones encargadas de encontrar los partidos segun los parametros
    partidosFTP = catalog['tournaments_FTP'] #Partidos FTP se refiere a el mapa organizado por Fecha-Torneo-Partidos

    entry = mp.get(partidosFTP,fecha)
    torneos = me.getValue(entry)
    entry2 = mp.get(torneos,torneo)
    partidos = me.getValue(entry2)

    #Aqui estamos iterando los partidos segun las condiciones del usuario
    for partido in lt.iterator(partidos):
            if partido["home_team"] not in teams:
                teams.append(partido["home_team"])
                lt.addLast(equipos_t, {"team" : partido["home_team"], "puntos_totales" : 0, "goles_diferencia" : 0, "puntos_penalti" : 0, "encuentros": 0 , "puntos_autogol":0, "victorias": 0, "empates": 0, "de
            if partido["away_team"] not in teams:
                teams.append(partido["away_team"])
                lt.addLast(equipos_t, {"team" : partido["away_team"], "puntos_totales" : 0, "goles_diferencia" : 0, "puntos_penalti" : 0,  "encuentros": 0 , "puntos_autogol":0, "victorias": 0, "empates": 0, "d
            pos = posicion_equipo_home(equipos_t,partido)
            pos2 = posicion_equipo_away(equipos_t,partido)
            sum_data(equipos_t, int(partido["home_score"]),int(partido["away_score"]),  pos)
            sum_data(equipos_t, int(partido["away_score"]),int(partido["home_score"]),  pos2)

            if partido['scorers'] != "unknown":
                for scorer in lt.iterator(partido['scorers']):
                    sum_penal_gol(equipos_t, partido["home_team"], scorer, pos)
                    sum_penal_gol(equipos_t, partido["away_team"], scorer, pos2)
                    if scorer['name']not in scorers:
                        scorers.append(scorer['name'])

                    if partido["home_team"] == scorer['team']:
                        lt.addLast(equipos_t["elements"][pos]["goleadores"], {"scorer": scorer["name"], "goals": 0, "matches":0, "avg_time [min]": 0})
                        pos_ = posicion_scorer(equipos_t["elements"][pos]["goleadores"],scorer)
                        sum_scorer(scorer,pos_, equipos_t["elements"][pos]["goleadores"])
                        lt.addLast(equipos_t["elements"][pos2]["goleadores"] , {"scorer": 'unavailable', "goals": 0, "matches":0, "avg_time [min]": 0})
                    elif  partido["away_team"] == scorer['team']:
                        lt.addLast(equipos_t["elements"][pos2]["goleadores"] , {"scorer": scorer["name"], "goals": 0, "matches":0, "avg_time [min]": 0})
                        pos_ = posicion_scorer(equipos_t["elements"][pos2]["goleadores"],scorer)
                        sum_scorer(scorer,pos_, equipos_t["elements"][pos2]["goleadores"])
                        lt.addLast(equipos_t["elements"][pos]["goleadores"], {"scorer": 'unavailable', "goals": 0, "matches":0, "avg_time [min]": 0})
                    else:
                        lt.addLast(equipos_t["elements"][pos]["goleadores"], {"scorer": 'unavailable', "goals": 0, "matches":0, "avg_time [min]": 0})
                        lt.addLast(equipos_t["elements"][pos2]["goleadores"] , {"scorer": 'unavailable', "goals": 0, "matches":0, "avg_time [min]": 0})
```

```
    for equipo in lt.iterator(equipos_t):
        for goleador in lt.iterator(equipo['goleadores']):
            if goleador['scorer'] != 'unavailable':
                if goleador["goals"] >=2:
                    goleador["avg_time [min]"] = (goleador["avg_time [min]"]/goleador["matches"])

        if lt.size(equipo['goleadores']) >= 2:
            scorers = sort_List(equipo["goleadores"], ordenar_goles)

        top_scorer = equipo["goleadores"]["elements"][0:1]
        equipo["goleadores"] = tabulate(top_scorer, headers="keys", tablefmt="grid")


    ordenados = sort_List(equipos_t, ordenar_puntos)

    return ordenados
```

```python
def sum_data(equipos_t, main, second, pos):
    # suma al total de encuentros del equipo
    equipos_t["elements"][pos]["encuentros"] +=1
    equipos_t["elements"][pos]["goles_anotados"] += main
    equipos_t["elements"][pos]["goles_en_contra"] += second

    if main > second:
        # suma al total de puntos y a la diferencia de goles
       equipos_t["elements"][pos]["puntos_totales"] += 3
       equipos_t["elements"][pos]["goles_diferencia"] += (main-second)
       equipos_t["elements"][pos]["victorias"] += 1
    elif main == second:
        #Si fue empate no modifica goles diferencia porque los datos a restar son los mismos a sumar
        equipos_t["elements"][pos]["puntos_totales"] += 1
        equipos_t["elements"][pos]["empates"] += 1
    elif main < second:
        equipos_t["elements"][pos]["goles_diferencia"] += (main-second)
        equipos_t["elements"][pos]["derrotas"] += 1


    return equipos_t

def sum_penal_gol(equipos_t, main_name, scorer, pos):
    # Suma puntos penalti si hubo anotacion
    if main_name == scorer["team"] and scorer["penalty"] == "True":
        equipos_t["elements"][pos]["puntos_penalti"] += 1
    if main_name == scorer["team"] and scorer["own_goal"] == "True":
        equipos_t["elements"][pos]["puntos_autogol"] += 1

def sum_scorer(partido,pos, goleadores):

    goleadores["elements"][pos]["goals"] +=1
    goleadores["elements"][pos]["matches"] +=1
    goleadores["elements"][pos]["avg_time [min]"] += float(partido["minute"])

    return goleadores
```

Entrada	Lista de datos, torneo, fecha
Salidas	Una lista compleja que contiene divido por torneo, los datos que pide el requerimiento y otra más pequeña que contiene los datos del mejor jugador.
Implementado (Sí/No)	Si. Implementado por Felipe Gaitán

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Cear las lista que vamos a utilizar para el requerimiento	O(1)
Sacar los datos que necesitamos para el requerimiento	O(1)
For partidos in lt.iterator(partidos)	O(n)
Posicion_equipo_home()	O(m)
Posicion_equipo_away()	O(m)
Los 2 sum_data	O(1)
Comprobar que sea los datos que necesitamos	O(1)
Iteramos la lista de los jugadores	O(k)
Comprobarmos las cosas que necesitamos	O(1)
Posicion_scorer	O(l)
For equipo in lt.iterator	O(k)
For goleador in lt.iterator(equipo[“goleadores”]	O(k)
Comprobamos las bariables que necesitamos	O(1)
Tabulamos	O(n)
Sort_list(filtrados, ordenar_puntos2)	O(l log (l))
Return todos los datos que necesitamos	O(1)
TOTAL	O(n+m+k+l *log(l))

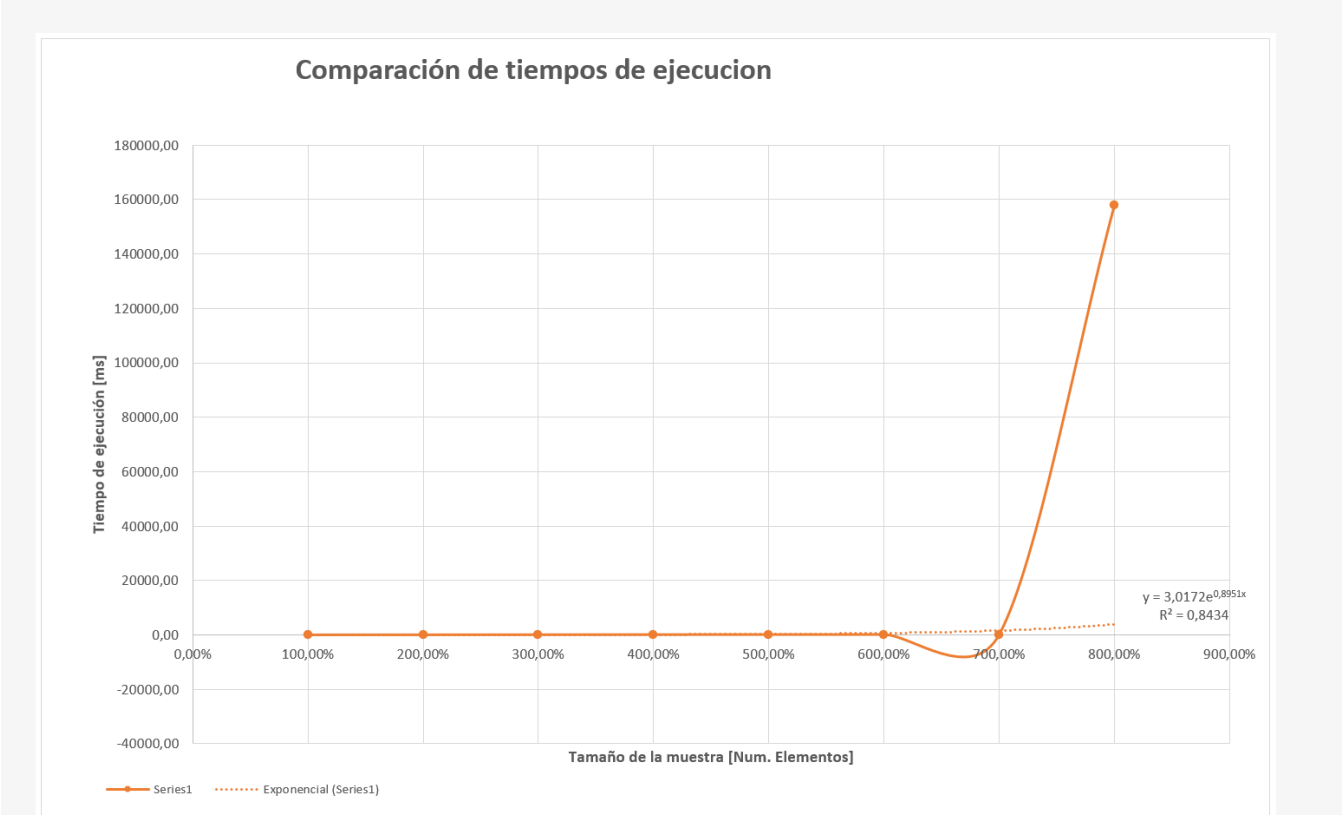
Pruebas Realizadas

Procesadores	**Intel core i7-12700K**
Memoria RAM	32GB
Sistema Operativo	Windows 11 Pro – 64 bits

Entrada	**Tiempo (ms)**
small	14,34
5 pct	28,23
10 pct	60,32
20 pct	99,60
30 pct	109,23
50 pct	115,88
80 pct	139,23
large	158.067

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Las gráficas y el tiempo de ejecución demuestran que el código tiene una complejidad que lo permite ser eficiente al tratar con una carga de datos de gran tamaño. Así mismo, evidencia que el código tiene una complejidad de O(n+m+k+l *log(l)).

Requerimiento 7

```
def req_7(catalog,torneo, puntos):
    #Variables para info arriba de la tabla
    total_goals = 0
    total_penalties = 0
    total_own_goals = 0
    total_matches = 0

    #Variables para los calculos del requerimiento
    torneos = catalog['torneos']
    entry = mp.get(torneos,torneo)
    partidos = me.getValue(entry)
    scorers = lt.newList("ARRAY_LIST")
    nombres = []
    filtrados = lt.newList("ARRAY_LIST")
    for partido in lt.iterator(partidos):
        if partido["scorers"] != "unknown":
            total_matches += 1
            for scorer in lt.iterator(partido['scorers']):
                if scorer['name'] not in nombres:
                    nombres.append(scorer['name'])
                    lt.addLast(scorers,{"scorer": str(scorer['name']), "total_points":0, "total_goals":0, "penalty_goals":0, "own_goals":0, "avg_time [min]":0, "total_tournaments":0, "
                pos = posicion_scorer(scorers, scorer)
                sum_scorer2(partido,pos, scorers, scorer)
                #Funcion encargada de sumar estadisticas para info arriba de la tabla
                total_penalties, total_own_goals = sum_estadisticas(total_penalties, total_own_goals, scorer)
                total_goals += int(partido['home_score']) + int(partido['away_score'])


for scorer in lt.iterator(scorers):
    if scorer['total_points'] == puntos:
        scorer["avg_time [min]"] = (scorer["avg_time [min]"]/scorer["total_goals"])
        ultimo = scorer["last_goal"]
        scorer["last_goal"] = print_tabulate(ultimo,["date","tournament","home_team","away_team","home_score","away_score","minute","penalty","own_goal"])
        lt.addLast(filtrados,scorer)

ordenados = sort_List(filtrados, ordenar_puntos2)

torneos_con_info = mp.size(torneos)
total_players = lt.size(scorers)

total_p_2points = lt.size(filtrados)
return ordenados, torneos_con_info, total_players, total_matches, total_goals, total_penalties, total_own_goals,total_p_2points
```

Descripción

Entrada	Estructura de datos, año inicial, año final y número de jugadores.
Salidas	Array list conformado por con los N mejores jugadores en la delta de tiempo y los otros datos que nos pide.
Implementado (Sí/No)	Si. Implementado por Felipe Gaitán.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

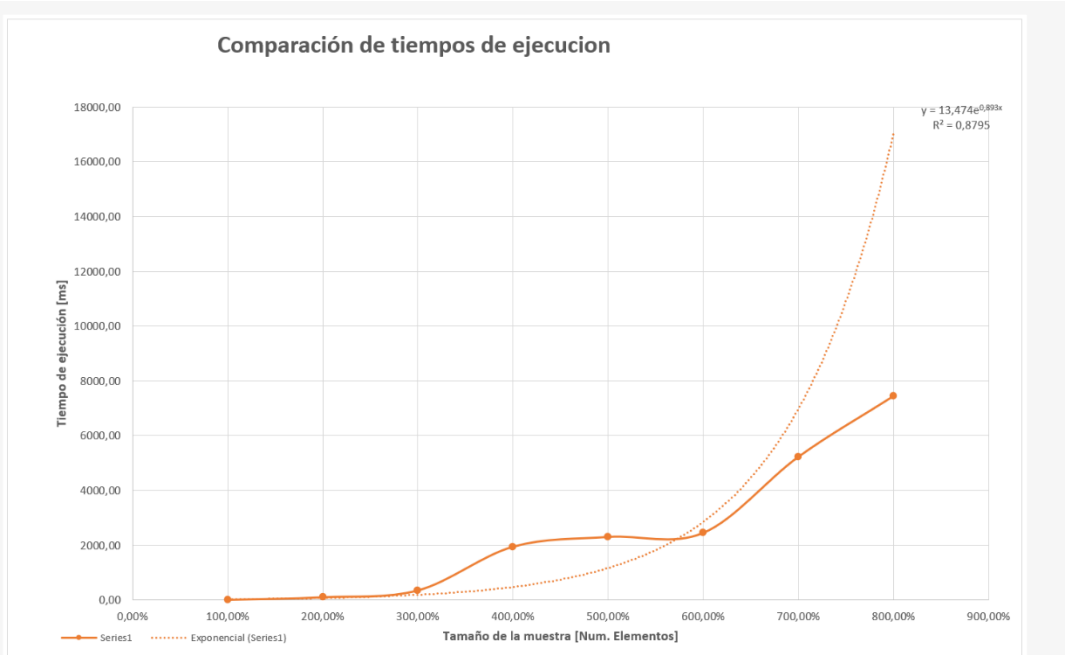
Pasos	**Complejidad**
Crear las diferente variables	O(1)
Obtener los el mapa torneos y los valores de ese mapa, por la llave q necesitamos	O(1)
Crear un array list	O(1)
Recorrer los elementos de partidos	O(n)

Evaluar si el elemento cumple con las condiciones por medio de un “if”	O(1)
Iterar partidos[“scorers”]	O(n)
Lt.addLast	O(1)
Posicion_scorer	O(m)
For scorer in lt.itarator	O(m)
Comprobar los datos que necesita	O(1)
Sort_list	O(k log (k))
TOTAL	**O(n+m + k log(k))**

Pruebas Realizadas

Procesadores	**Intel core i7-12700K**
Memoria RAM	32GB
Sistema Operativo	Windows 11 Pro – 64 bits

Entrada	**Tiempo (ms)**
small	6,54
5 pct	100,34
10 pct	353,41
20 pct	1948,14
30 pct	2304,23
50 pct	2452,87
80 pct	5231,32
large	7.447



Análisis

Este código tiene una complejidad de O(n+m + k log(k)), que significa que se demora aproximadamente el númerode elementos de la carga de datos, n, * n. Así mismo, esto puede ser evidenciado a través de las gráficas, que nos muestran que los tiempos obtenidos por las máquinas son acotados por una función cuadrática.

Como se puede ver en la gráfica llega a tender hacia n +m + k log(k) pero no lo alcanza de todo.

Requerimiento 8

```
def req_8(catalog, pais, inicial,final):
    teams = []
    scorers = []
    equipos_t = lt.newList("ARRAY_LIST")

    paises = catalog['paises']
    key_m =  pais

    entry = mp.get(paises,key_m)
    torneos = me.getValue(entry)

    for partido in lt.iterator(torneos):

        if comparardelta(partido["date"][0:4],inicial, final):
            if partido['tournament'] != "Friendly":
                if partido["date"][0:4] not in teams:
                    teams.append(partido["date"][0:4])
                    lt.addLast(equipos_t, {"year" : partido["date"][0:4], "puntos_totales" : 0, "goles_diferencia" : 0, "puntos_penalti" : 0, "encuentros": 0 , "puntos_autogol":0, "victorias": 0, "empates": 0, "derrotas": 0, "goles_anotados":0

                pos = posicion_fecha1(equipos_t,partido)


                sum_data2(equipos_t, int(partido["home_score"]),int(partido["away_score"]),  pos)



                if partido['scorers'] != "unknown":
                    for scorer in lt.iterator(partido['scorers']):
                        sum_penal_gol(equipos_t, partido["home_team"], scorer, pos)

                        if scorer['name']not in scorers:
                            scorers.append(scorer['name'])

                        if partido["home_team"] == scorer['team']:
                            lt.addLast(equipos_t["elements"][pos]["goleadores"], {"scorer": scorer["name"], "goals": 0, "matches":0, "avg_time [min]": 0})
                            pos_ = posicion_scorer(equipos_t["elements"][pos]["goleadores"],scorer)
                            sum_scorer(scorer,pos_, equipos_t["elements"][pos]["goleadores"])

                        elif  partido["away_team"] == scorer['team']:

                            lt.addLast(equipos_t["elements"][pos]["goleadores"], {"scorer": 'unavailable', "goals": 0, "matches":0, "avg_time [min]": 0})
                        else:
                            lt.addLast(equipos_t["elements"][pos]["goleadores"], {"scorer": 'unavailable', "goals": 0, "matches":0, "avg_time [min]": 0})
```

```
for equipo in lt.iterator(equipos_t):
    for goleador in lt.iterator(equipo['goleadores']):
        if goleador['scorer'] ≠ 'unavailable':
            if goleador["goals"] ⩾2:
                goleador["avg_time [min]"] = (goleador["avg_time [min]"]/goleador["matches"])

    if lt.size(equipo['goleadores']) ⩾ 2:
        scorers = sort_List(equipo["goleadores"], ordenar_goles)

    top_scorer = equipo["goleadores"]["elements"][0:1]
    equipo["goleadores"] = tabulate(top_scorer, headers="keys", tablefmt="grid")

primero = lt.firstElement(torneos)

primero_y = lt.firstElement(equipos_t)
ultimo_y = lt.lastElement(equipos_t)
year = int(primero_y["year"]) - int(ultimo_y["year"])
print(year)

size_t = lt.size(torneos)
print(size_t)
tablaInicio=lt.newList("ARRAY_LIST")
lt.addLast(tablaInicio,primero)

return equipos_t,tablaInicio,torneos
```

Descripción

Entrada	Estructura de datos, año inicial y año final

Salidas	Array list con los subsectores con el mayor total de impuestos a cargo entre los años indicados por parámetro.
Implementado (Sí/No)	Si. Implementado por Carlos Said.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Aplicar un get con la llave pais	O(1)
Aplicar un GetValues con el get anterior	O(1)
Iterar getValues	O(n)
Comparar que la delta de tiempo este en el rango.	O(1)
Recorrer los elementos de la lista.	
Verificar que no este presente en la lista de equipos	O(1)
Agregar al final del Array_list	O(1)
Aplicar posición fecha1, para encontrar pos	O(n)
Sum data	O(1)
Ciclo para jugadores	O(n)
Agregar a la lista dentro del ciclo de jugadores	O(1)
Recorrer el Array_list y jugadores para ajustar la tabla	O(n*m)

TOTAL	O(n*m)

Pruebas Realizadas

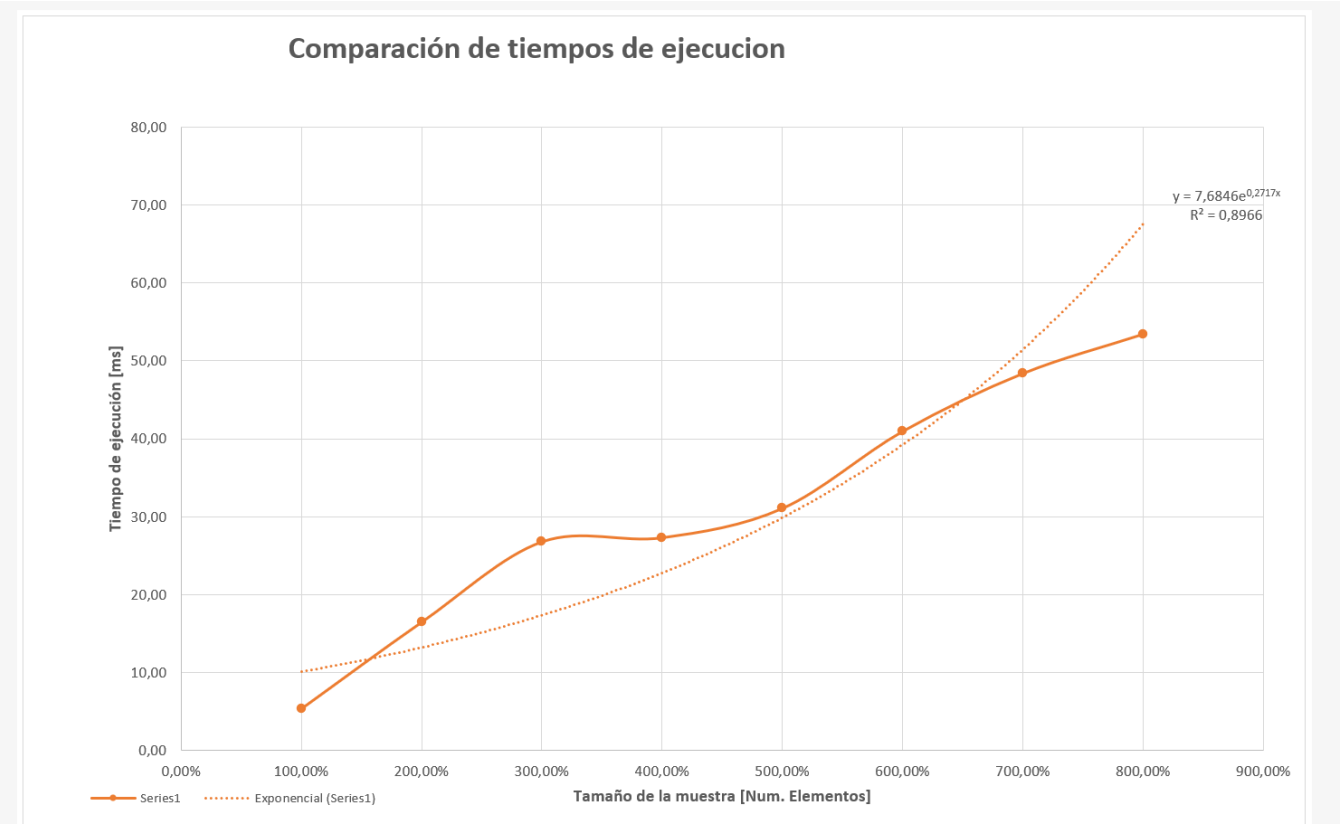
Procesadores	**AMD Ryzen 5 5700x 3.4GHz GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Home Single – 64 bits

Entrada	**Tiempo (ms)**
small	5,40
5 pct	16,51
10 pct	26,81
20 pct	27,33
30 pct	31,09
50 pct	40,98
80 pct	48,40
large	53

Graficas

Las gráficas con la representación de las pruebas realizadas.

- Máquina 1:



Análisis

El anterior código tiene una complejidad alta, gracias a que está acotado por O(N^4). Por lo tanto, se vuelve un código lento e ineficaz. No obstante, tiene la capacidad de mostrar la información pedida y no presenta errores en su ejecución.