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

1. Harold Esteban Piñeros Monroy, h.pineros@uniandes.edu.co, 202316402

2. Carlos Alberto Poveda Riaño, ca.povedar1@uniandes.edu.co, 202315546

3. Luis Sebastián Contreras Diaz, ls.contreras@uniandes.edu.co, 202311819

Requerimiento 1

```
elif int(inputs) == 2:
    number_matchs =int( input("Ingrese el numero de partidos: "))
    name_team = input("Ingrese el nombre del Equipo: ")
    condition_team = input("Ingrese la condicion del equipo (local, visitante o indiferente): ")
    total_matchs = controller.sortName(control['model']['results'], name_team, condition_team, number_matchs)
    printSimpleTable(total_matchs,['date','home_team','away_team','country','city','home_score','away_score'])
```

```
def sortName(data,name_team, condition_team, number_matchs):
    e =0
    total_indices = []
    if condition_team.lower() == "local":
        f ="home_team"
        indices, NameSort= searchname(data,name_team, condition_team, number_matchs,f)
        total_indices = indices

    elif condition_team.lower() == "visitante":
        f ="away_team"
        indices, NameSort= searchname(data,name_team, condition_team, number_matchs,f)
        total_indices = indices


    else:
        e =1
        f ="home_team"
        indices1, NameSort1= searchname(data,name_team, condition_team, number_matchs,f)
        z ="away_team"
        indices2, NameSort2= searchname(data,name_team, condition_team, number_matchs, z)
    if e==1:
        total_teams = lt.newList('ARRAY_LIST')
        answerSort = lt.newList('ARRAY_LIST')
        for i in indices1:
            element= lt.getElement(NameSort1,i+1)
            lt.addFirst(total_teams,element)
        for i in indices2:
            element= lt.getElement(NameSort2,i+1)
            lt.addFirst(total_teams,element)
        answer= sa.sort(total_teams, compare_shootouts)
        answerSort = getFirstNum(number_matchs,answer)
        return answerSort

    else:
        total_teams = lt.newList('ARRAY_LIST')
        answerSort = lt.newList('ARRAY_LIST')
        for i in total_indices:
            element= lt.getElement(NameSort,i+1)
            lt.addFirst(total_teams,element)
        answer= sa.sort(total_teams, compare_shootouts)
```

Descripción

La función sortname() recibe una estructura de datos “data” y la “condición” y”numero de partidos”. Su objetivo es encontrar y retornar una lista de partidos según la condición de acuerdo con el numero de partidos que desea el usuario,

Entrada	numero de partidos el nombre del equipo la condicion del equipo
Salidas	lista de partidos
Implementado (Sí/No)	Sí fue implementado por Carlos Poveda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear una lista para almacenar los datos (“ArrayList”)	O(1)
Ordenar la lista	O(N log (N))
Recorrer la lista de home_team o away_team para convertirlos a listas	O(N)
busqueda binaria para la lista	O(N log (N))
TOTAL	***O(***N***)***

Pruebas Realizadas

Las pruebas fueron realizadas con entrada de búsqueda “italy”, y buscando siempre los 15 primeros partidos. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable. Se

Procesadores	**Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz 2.70 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Tamaño	**Tiempo (ms)**
small	16.81
5pct	99.42
10pct	211.73
20pct	441.47
30pct	653.351
50pct	981.74

80pct	1524.73
-large	1769.39

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño	Salida	Tiempo (ms)
small		16.81
5 pct		99.42
10 pct		211.73
20 pct		441.47
30 pct		653.351
50 pct		981.74

80 pct	========== Req No. 1 Inputs ========== Ingrese el numero de partidos: 15 Ingrese el nombre del Equipo: Italy Ingrese la condicion del equipo (local, visitante o indiferente): local ========== Req No.1 Results ========== +------------+-----------+-------------+---------+----------+------------+-------------+ | date | home_team | away_team | country | city | home_score | away_score | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2023-03-23 | Italy | England | Italy | Napoli | 1 | 2 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2022-06-07 | Italy | Hungary | Italy | Cesena | 2 | 1 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2022-06-01 | Italy | Argentina | France | Paris | 0 | 3 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2021-11-12 | Italy | Switzerland | Italy | Rome | 1 | 1 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2021-10-06 | Italy | Spain | Italy | Milan | 1 | 2 | +------------+-----------+-------------+---------+----------+------------+-------------+	1524.73
large	========== Req No. 1 Inputs ========== Ingrese el numero de partidos: 15 Ingrese el nombre del Equipo: Italy Ingrese la condicion del equipo (local, visitante o indiferente): local ========== Req No.1 Results ========== +------------+-----------+-------------+---------+----------+------------+-------------+ | date | home_team | away_team | country | city | home_score | away_score | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2023-03-23 | Italy | England | Italy | Napoli | 1 | 2 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2022-09-23 | Italy | England | Italy | Milan | 1 | 0 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2022-06-07 | Italy | Hungary | Italy | Cesena | 2 | 1 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2022-06-04 | Italy | Germany | Italy | Bologna | 1 | 1 | +------------+-----------+-------------+---------+----------+------------+-------------+ | 2022-06-01 | Italy | Argentina | France | Paris | 0 | 3 | +------------+-----------+-------------+---------+----------+------------+-------------+	1769.39

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Los resultados de las pruebas respaldan el análisis de complejidad. A medida que aumenta el tamaño del conjunto de datos, se observa un aumento significativo en el tiempo de ejecución. Esto se debe a que la función debe recorrer toda la lista de partidos para convertirlos en una lista O(n). El comportamiento no es completamente eficiente y tiende a ser lineal en relación con el tamaño del conjunto de datos, como se puede ver en las pruebas "50pct" y "large".

Requerimiento 2

```python
def iter_get_first_n_goals_by_player(data_structs, player_name, n):
    player_goals = lt.newList('ARRAY_LIST')
    total_goals = 0
    sa.sort(data_structs["goalscore"], cmp_partidos_by_fecha_y_pais)
    # Recorremos la lista de goles y seleccionamos los que coincidan con el jugador
    for goal in lt.iterator(data_structs['goalscore']):
        if goal['scorer'].lower() == player_name.lower():
            lt.addLast(player_goals, goal)
            total_goals += 1
            if total_goals == n:
                break
    return total_goals, player_goals
```

.

Descripción

La función iter_get_first_n_goals_by_player recibe una estructura de datos data_structs, el nombre de un jugador, player_name, y un número n. Su objetivo es encontrar y retornar los primeros n goles anotados por el jugador especificado, ordenados por fecha y país. Además, la función registra el número total de goles encontrados.

Entrada	La lista de datos, el nombre del jugador y los n goles
Salidas	Número total de goles, y la lista con las especificaciones de los goles
Implementado (Sí/No)	Sí fue implementado por Luis Sebastián Contreras

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear una lista para almacenar los datos (“ArrayList”)	O(1)
Se crea una variable para almacenar la cantidad de goles	O(1)
Ordenar la lista de goles	O(N log (N))
Recorrer la lista de goles y seleccionar los primeros 'n' goles	O(N)
TOTAL	***O(N)***

Pruebas Realizadas

Las pruebas fueron realizadas con entrada de búsqueda de los goles de Michael Ballack, y buscando siempre los 6 primeros goles. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable y datetime. Se creó un ti

Procesadores	**12th Gen Intel(R) Core(TM) i5-1235U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

Tamaño	Tiempo (ms)
small	21.77
5pct	481.98
10pct	1044.15
20pct	2287.90
30pct	1066.41
50pct	7207.83
80pct	11579.32
-large	15350.27

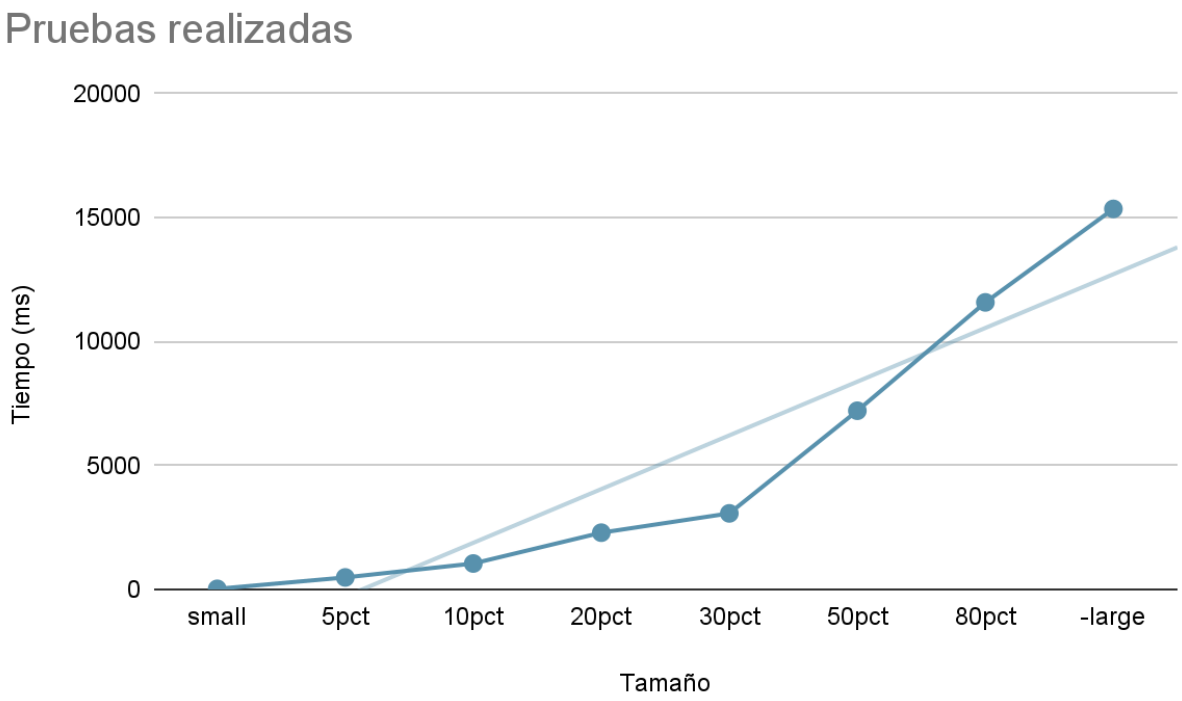
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño	Salida	Tiempo (ms)
small	Michael Ballack, 2 primeros	21.77
5 pct	Michael Ballack, 6 primeros	481.98
10 pct	Clint Dempsey, 5 primeros	312.2
20 pct	Michel Platini, 4 primeros	1987.17
30 pct	Oleg Salenko, 5 primeros	3277.81
50 pct	Okan Buruk, 4 primeros	7207.83
80 pct	Roque Júnior, primer gol	10624.35
large	Ricardo Aquino, primer gol	16966.15

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Los resultados de las pruebas respaldan el análisis de complejidad. A medida que aumenta el tamaño del conjunto de datos, se observa un aumento significativo en el tiempo de ejecución. Esto se debe a que la función debe recorrer toda la lista de goles para buscar y seleccionar los goles del jugador específico O(n). El comportamiento no es completamente eficiente y tiende a ser lineal en relación con el tamaño del conjunto de datos, como se puede ver en las pruebas "large" y "80pct".

Requerimiento 3

Descripción

```
elif int(inputs) == 4:
    print("========================= Req No. 3 Inputs ===============")
    team_name = input("Ingrese el nombre del equipo: ")
    fecha_inicio = input("Ingrese la fecha de inicio (YYYY-MM-DD):")
    fecha_fin = input("ingrese la fecha de fin (YYYY-MM-DD):")
    print("======================== Req No.3 Results ================")
    time, total_games, total_home_games, total_away_games, games_played = controller.consulta_partidos_equipo_periodo(control, team_name,
                                                                                                                       fecha_inicio, fecha_fin)
    print("Delta de tiempo fue:", str(time))
    print_games_over_a_period_of_time(total_games, total_home_games, total_away_games, games_played)
```

```
def iter_consultar_partidos_equipo_periodo(data_structs, team_name, fecha_inicio, fecha_fin):

    games_played = lt.newList("ARRAY_LIST")


    sa.sort(data_structs["results"], cmp_date)
    total_games = 0
    total_home_games = 0
    total_away_games = 0
    fecha_inicio = datetime.datetime.strptime(fecha_inicio, '%Y-%m-%d')
    fecha_fin = datetime.datetime.strptime(fecha_fin, '%Y-%m-%d')

    for partido in lt.iterator(data_structs["results"]):
        goal_date = datetime.datetime.strptime(partido['date'], '%Y-%m-%d')
        if fecha_inicio <= goal_date <= fecha_fin and (partido["home_team"].lower() == team_name.lower() or partido["away_team"].lower() == team_name.lower()):
            total_games += 1
            penalty = buscar_penaltis(data_structs["goalscore"],partido['date'], partido['home_team'], partido['away_team'])
            own_goal = buscar_autogoles(data_structs["goalscore"],partido['date'], partido['home_team'], partido['away_team'])
            if partido["home_team"].lower() == team_name.lower():
                total_home_games += 1
            if partido["away_team"].lower() == team_name.lower():
                total_away_games += 1

            partido["penalty"] = penalty
            partido["own_goal"] = own_goal
            lt.addLast(games_played, partido)
    return total_games , total_home_games, total_away_games, games_played
```

La función iter_consultar_partidos_equipo_periodo, toma como entrada una estructura de datos data_structs que contiene información sobre partidos, goles y torneos, el nombre del equipo team_name, así como el intervalo de tiempo de las fechas de inicio y fin en las que se desea consultar partidos jugados por el equipo. Luego, realiza una serie de acciones para buscar y recopilar información sobre los partidos jugados por el equipo en el período de tiempo especificado.

Entrada	La lista de datos, el nombre del equipo, fecha de inicio y fecha final
Salidas	Número total de partidos disputados. Número total de partidos disputados como local. Número total de partidos disputados como visitante. El listado de los partidos disputados ordenados cronológicamente
Implementado (Sí/No)	Sí fue implementado por Harold Esteban Piñeros Monroy

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Crear una lista para almacenar los datos (“ArrayList”)	O(1)
Se crea una variable para almacenar la cantidad de partidos	O(1)
Ordenar la lista de partidos	O(N log (N))
Recorrer la lista de partidos y seleccionas los que cumplan la condicion	O(N)
TOTAL	***O(N)***

Pruebas Realizadas

Las pruebas fueron realizadas con entrada de búsqueda de los goles de Michael Ballack, y buscando siempre los 6 primeros goles. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable y datetime. Se creó un ti

Procesadores	
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

Tamaño	**Tiempo (ms)**
small	249.23
5pct	4410.74
10pct	19054.30
20pct	77215.52
30pct	267836.02
50pct	725376.44
80pct	1611050.31
-large	2336426.75

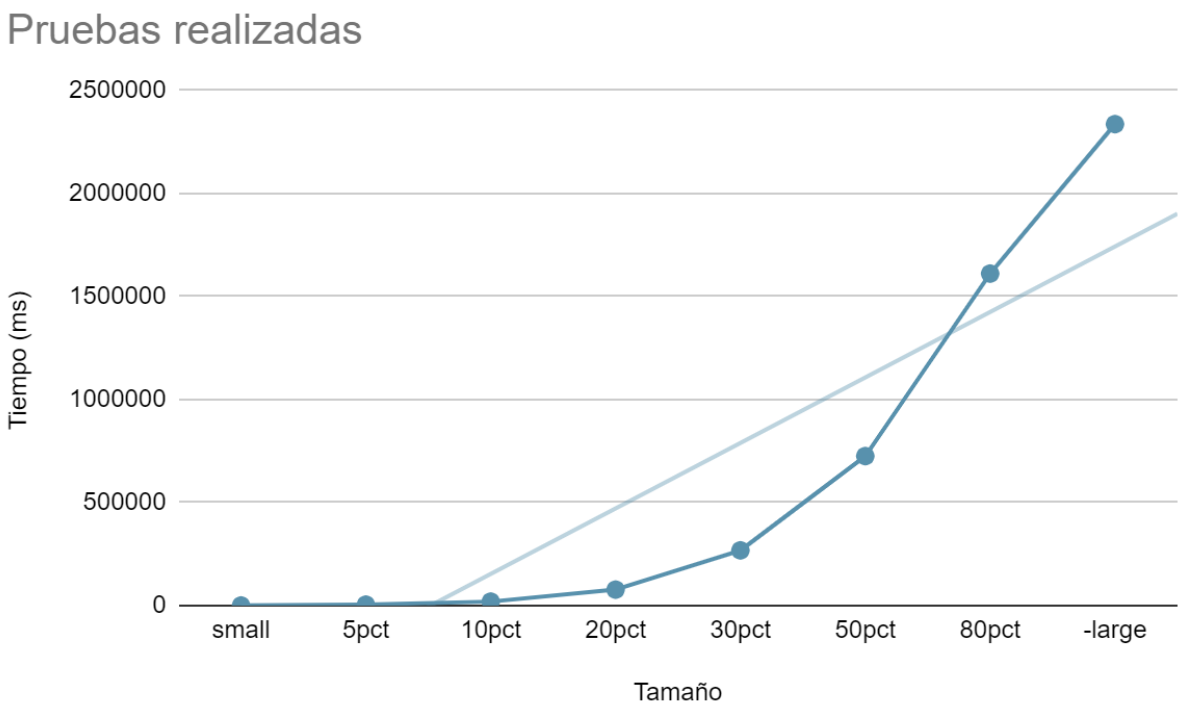
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Tamaño	Salida	Tiempo (ms)
small	partidos jugados por el equipo: 5 partidos jugados como local: 3 partidos jugados como visitante: 2	249.23
5 pct	partidos jugados por el equipo: 28 partidos jugados como local: 11 partidos jugados como visitante: 17	4410.74
10 pct	partidos jugados por el equipo: 69 partidos jugados como local: 30 partidos jugados como visitante: 39	19054.30
20 pct	partidos jugados por el equipo: 152 partidos jugados como local: 72 partidos jugados como visitante: 80	77215.52
30 pct	partidos jugados por el equipo: 275 partidos jugados como local: 133 partidos jugados como visitante: 142	267836.02
50 pct	partidos jugados por el equipo: 451 partidos jugados como local: 224 partidos jugados como visitante: 227	725376.44
80 pct	partidos jugados por el equipo: partidos jugados como local: partidos jugados como visitante:	1611050.31
large	partidos jugados por el equipo: partidos jugados como local: partidos jugados como visitante:	2336426.75

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La función principal, iiter_consultar_partidos_equipo_periodo, utiliza una estrategia eficiente para ordenar la lista de objetivos por fecha, lo que permite búsquedas más rápidas por fecha. Sin embargo, la complejidad de esta operación de clasificación es O(n * log(n)), donde "n" es el número de objetivos en la lista. Iterar a través de la lista de objetivos tiene una complejidad lineal O(n) porque hay que comprobar cada objetivo para determinar si coincide con la fecha y los criterios del nombre del equipo. Además, la función buscar_penalti y buscar_autogol realiza una búsqueda lineal de la lista de resultados, lo que significa complejidad O(m), donde "m" es el número de resultados.

Los resultados respaldan este complejo análisis. A medida que aumenta el tamaño del conjunto de datos, especialmente cuando aumenta el número de objetivos y resultados, se observa un aumento significativo en el tiempo de ejecución del algoritmo. Esto se debe a la necesidad de desplazarse por la lista completa de resultados y goles y buscar coincidencias. Aunque ordenar la lista de objetivos mejora el rendimiento de la búsqueda por fecha, el comportamiento general suele ser lineal o incluso ligeramente superlineal con respecto al tamaño del conjunto de datos.

Requerimiento 4

```
elif int(inputs) == 5:
    """PConsultar los partidos relacionados
    con un torneo durante un
    periodo especifico."""
    print("========================= Req No. 4 Inputs ==============")
    name_tournament = input(" Ingrese el nombre del Torneo: ")
    start_date = input("Ingrese la fecha de inicio del periodo a consultar (YYYY-MM-DD): ")
    end_date = input("Ingrese la fecha de final del periodo a consultar (YYYY-MM-DD): ")
    matchs,total_coutries, total_cities, size , sizematches= controller.queryMatchsbyPeriod(name_tournament, start_date, end_date ,cont
    print("========================= Req No. 4 Inputs ==============")
    print(f"tournament name : {name_tournament}")
    print(f"Start date: {start_date}")
    print(f"End date: {end_date}")
    print("========================= Req No. 4 Inputs ==============")
    print(f"{name_tournament} total matches: ",sizematches)
    print(f"{name_tournament} total countries: ",total_coutries)
    print(f"{name_tournament} total cities: ",total_cities)
    if size >= 6:
        print("\n the Tournament results has more than 6 records")
    else:
        print("\n the Tournament results has more than 6 records")
    printSimpleTable(matchs,['date','tournament','country','city','home_team','away_team','home_score','away_score','winner'])
    print(" delta tiempo fue:", str(time))
```

```
def queryMatchsbyPeriod(name_tournament, start_date, end_date,goalscore, results):
    """
    Función que soluciona el requerimiento 4
    """
    newarray = lt.newList('ARRAY_LIST')
    goals =sa.sort(shootouts,compare_home)
    total_paises = set()
    total_ciudades = set()
    sa.sort(shootouts,compare_away)
    start_date= datetime.datetime.strptime(start_date,"%Y-%m-%d")
    end_date = datetime.datetime.strptime(end_date,"%Y-%m-%d")
    r = lt.newList('ARRAY_LIST')
    for i in lt.iterator(results):
        date = datetime.datetime.strptime(i['date'],"%Y-%m-%d")
        if date<= end_date and date>= start_date and i['tournament']== name_tournament:
            country = i['country']
            city = i['city']
            total_paises.add(country)
            total_ciudades.add(city)
            if i['home_score'] == i['away_score']:
                winner = buscar_ganador(shootouts, date, i['home_team'], i ['away_team'])
            else:
                winner = 'Unknown'
            i['winner'] = winner
            lt.addLast(newarray, i)
```

Descripción

La función “queryMatchsbyPeriod” recibe dos estructura de datos “results” y “shootouts” y el nombre del torneo y la fecha de inicio y la fecha final de partidos”. Su objetivo es encontrar y retornar una lista de partidos de acuerdo al torneo dentro el rango de fechas,

Entrada	nombre de torneo fecha inicial fecha final
Salidas	lista de partidos
Implementado (Sí/No)	Sí fue implementado por Carlos Poveda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crea una lista vacía llamada games_playeds para almacenar lospartidos jugados por el quipo en el período.	O(1)
Ordena la lista de partidos en data_structs['results'] por fecha, utilizando una función de comparación personalizada llamada cmp_date.	O(N log (N))
Inicializa contadores para el número total de partidos (total_games), el número de partidos jugados de local (home_games), el número de partidos jugados de visitante (away_games).	O(1)
Recorrer la lista para buscar los partidos	O(N)
TOTAL	***O(***N***)***

Pruebas Realizadas

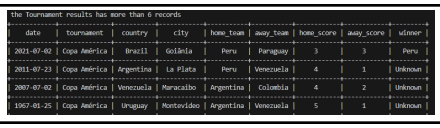
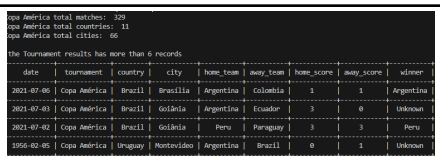
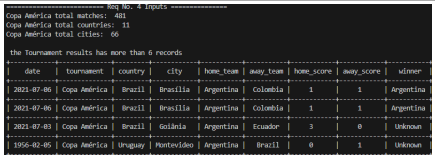
Las pruebas fueron realizadas con entrada de búsqueda del torneo es:Copa América, entre el periodo 1955-06-01 hasta 2022-06-30. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable. Se

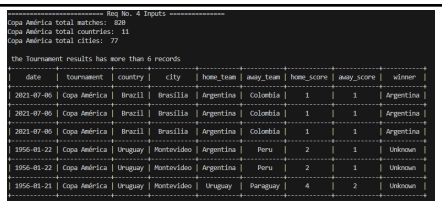
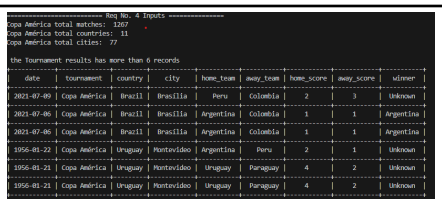
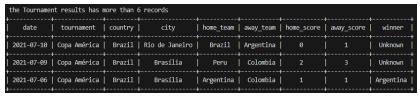
Procesadores	**Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz 2.70 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Tamaño	Tiempo (ms)
small	10.2
5pct	53.86
10pct	107.95
20pct	201.99
30pct	324.65
50pct	532.21
80pct	842.38
-large	1040.667

Tablas de datos

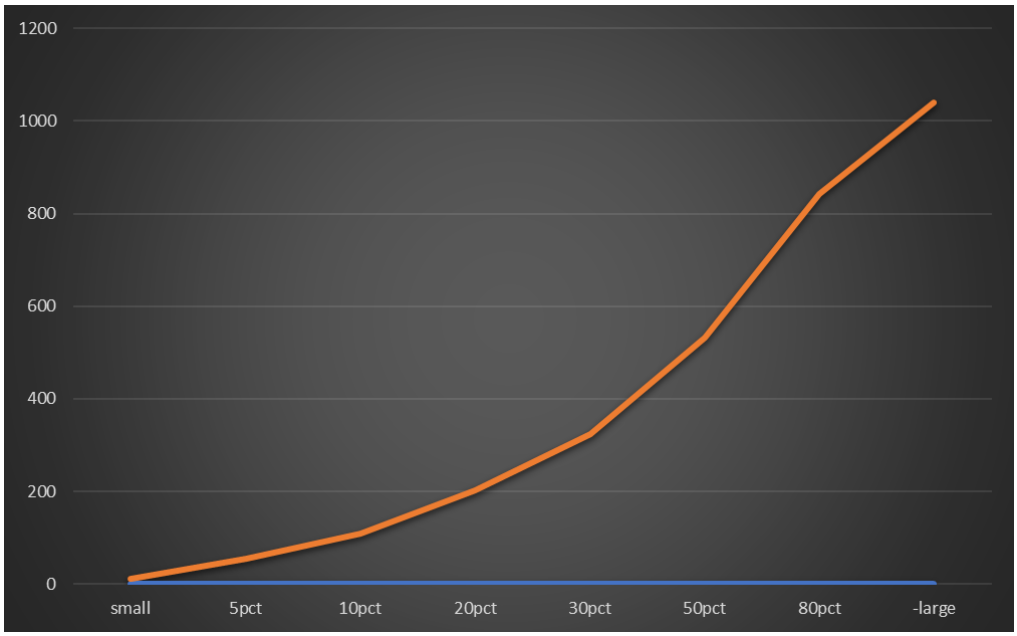
Las tablas con la recopilación de datos de las pruebas.

Tamaño	Salida	Tiempo (ms)
small		10.2
5 pct		53.86
10 pct		107.95
20 pct		201.99

30 pct		324.65
50 pct		532.21
80 pct		842.38
large		1040.667

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Los resultados de las pruebas respaldan el análisis de complejidad. A medida que aumenta el tamaño del conjunto de datos, se observa un aumento significativo en el tiempo de ejecución. Esto se debe a que la

función debe recorrer toda la lista de partidos para convertirlos en una lista O(n). El comportamiento no es completamente eficiente y tiende a ser lineal en relación con el tamaño del conjunto de datos, como se puede ver en las pruebas "50pct" y "large".

Requerimiento 5

.

```python
def iter_consultar_anotaciones_jugador_periodo(data_structs, jugador_nombre, fecha_inicio, fecha_fin):
    """
    Consulta las anotaciones de un jugador en un período de tiempo.
    Devuelve una lista de goles del jugador en ese período.
    """
    player_goals = lt.newList('ARRAY_LIST')

    # Ordena la lista de goles por fecha y minuto
    sa.sort(data_structs['goalscore'], cmp_date_and_minute)

    total_goals = 0
    total_tournaments = set()
    penalties = 0
    own_goals = 0
    fecha_inicio = datetime.datetime.strptime(fecha_inicio, '%Y-%m-%d')
    fecha_fin = datetime.datetime.strptime(fecha_fin, '%Y-%m-%d')
    for goal in lt.iterator(data_structs['goalscore']):
        goal_date = datetime.datetime.strptime(goal['date'], '%Y-%m-%d')
        if fecha_inicio <= goal_date <= fecha_fin and goal['scorer'].lower() == jugador_nombre.lower():
            total_goals += 1
            # Obtener el nombre del torneo desde la lista de resultados
            tournament = buscar_torneo(data_structs['results'], goal['date'], goal['home_team'], goal['away_team'])
            if tournament:
                total_tournaments.add(tournament)
            if goal['penalty'] == 'True':
                penalties += 1
            if goal['own_goal'] == 'True':
                own_goals += 1

            # Incluir el nombre del torneo en el gol
            goal['tournament'] = tournament
            lt.addLast(player_goals, goal)

    return total_goals, len(total_tournaments), penalties, own_goals, player_goals
```

```python
def buscar_torneo(results, goal_date, home_team, away_team):
    """
    Busca el nombre del torneo en la lista de resultados según la fecha y los equipos.
    """
    for result in lt.iterator(results):
        result_date = datetime.datetime.strptime(result['date'], '%Y-%m-%d')
        if result_date == datetime.datetime.strptime(goal_date, '%Y-%m-%d') and result['home_team'] == home_team and result['away_team'] == away_team:
            return result['tournament']
    return 'Desconocido'

#Req 6
```

Descripción

La función iter_consultar_anotaciones_jugador_periodo, toma como entrada una estructura de datos data_structs que contiene información sobre partidos, goles y torneos, el nombre del jugador jugador_nombre, así como las fechas de inicio y fin en las que se desea consultar las anotaciones del jugador. Luego, realiza una serie de acciones para buscar y recopilar información sobre los goles anotados por el jugador en el período de tiempo especificado.

La función buscar_torneo, busca el nombre del torneo en la lista de resultados según la fecha del gol (goal_date) y los equipos que jugaron el partido (home_team y away_team). Es una función auxiliar utilizada por iter_consultar_anotaciones_jugador_periodo para obtener el nombre del torneo en el que se anotó un gol específico.

Entrada	data_structs: lista en la que se encuentran los datos; jugador_nombre: El nombre del jugador del cual se desea consultar; fecha_inicio La fecha de inicio del período de tiempo en el que se desea buscar; fecha_fin: La fecha final del período de tiempo en el que se desea buscar
Salidas	Respuesta esperada del algoritmo; total_goals: número total de anotaciones realizadas por el jugador; len(total_tournaments); número de torneos en que anotó el jugador, penalties; número de anotaciones obtenidas desde el punto penal; own_goals; número total de autogoles cometidos; player_goals: el listado de las anotaciones del jugador ordenadas cronológicamente por fecha y minuto.
Implementado (Sí/No)	Sí fue implementado por Luis Sebastián Contreras

Análisis de complejidad

Pasos	Complejidad
Crea una lista vacía llamada player_goals para almacenar los goles anotados por el jugador en el período.	O(1)
Ordena la lista de goles en data_structs['goalscore'] por fecha y minuto, utilizando una función de	O(N log (N))

comparación personalizada llamada cmp_date_and_minute.	
Inicializa contadores para el número total de goles (total_goals), el número de torneos diferentes en los que el jugador anotó (total_tournaments), el número de penales anotados (penalties) y el número de autogoles anotados (own_goals).	O(1)
Convierte las fechas de inicio y fin proporcionadas en formato de cadena en objetos datetime.	O(1)
Itera a través de los goles en data_structs['goalscore'] y verifica si el gol pertenece al jugador especificado y si su fecha está dentro del período de tiempo dado.	O(N)
Si se cumple la condición, actualiza los contadores y busca el nombre del torneo en el que se anotó el gol utilizando la función buscar_torneo.	O(N)
Agrega información sobre el torneo al gol y agrega el gol a la lista player_goals.	O(1)
Si se cumple la condición de que penalty sea True, suma 1 a la variable de penalties, si se cumple la condición de que se haya anotado por autogol, se suma 1	O(1)
Devuelve el número total de goles, el número de torneos distintos, el número de penales y autogoles, y la lista de goles del jugador.	O(1)
TOTAL	***O(N log (N))***

Pruebas Realizadas

Las pruebas fueron realizadas con entrada de búsqueda de los goles de del jugador Ali Daei con fecha de inicio 1999-03-25 y con fecha final 2021-11-23. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable y datetime. Se creó un ti

Procesadores	**12th Gen Intel(R) Core(TM) i5-1235U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

Tamaño	Tiempo (ms)
small	99.94
5pct	714.80
10pct	1435.55
20pct	3541.65
30pct	6181.79
50pct	12288.77
80pct	23560.61
-large	47800.28

Tablas de datos

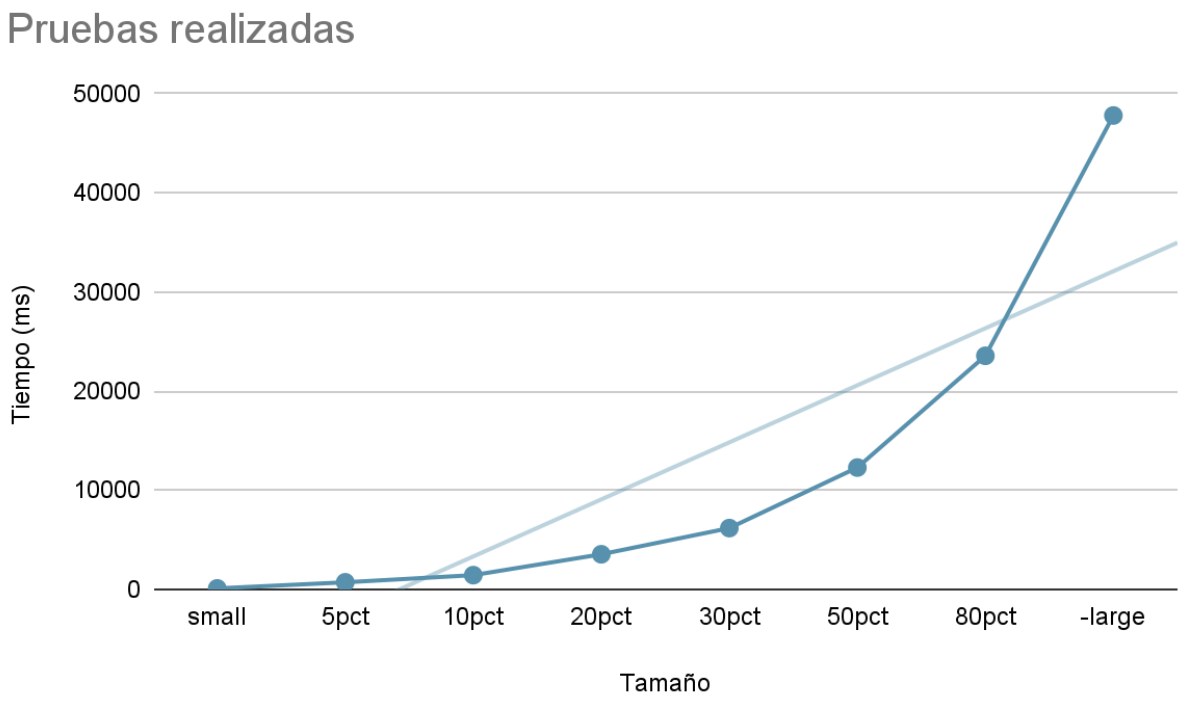
Las tablas con la recopilación de datos de las pruebas.

Tamaño	Salida	Tiempo (ms)
small	Total de goles anotados por el jugador: 2 Total de torneos: 2 Total de penalties: 0 Total de autogoles: 0	99.94
5 pct	Total de goles anotados por el jugador: 4 Total de torneos: 2 Total de penalties: 0 Total de autogoles: 0	714.80
10 pct	Total de goles anotados por el jugador: 4 Total de torneos: 2 Total de penalties: 0 Total de autogoles: 0	1435.55
20 pct	Total de goles anotados por el jugador: 8 Total de torneos: 2 Total de penalties: 1 Total de autogoles: 0	3541.65

30 pct	Total de goles anotados por el jugador: 14 Total de torneos: 2 Total de penalties: 3 Total de autogoles: 0	6181.79
50 pct	Total de goles anotados por el jugador: 18 Total de torneos: 2 Total de penalties: 5 Total de autogoles: 0	12288.77
80 pct	Total de goles anotados por el jugador: 23 Total de torneos: 2 Total de penalties: 6 Total de autogoles: 0	23560.615
large	Total de goles anotados por el jugador: 25 Total de torneos: 2 Total de penalties: 6 Total de autogoles: 0 Detalles de los goles:	47800.285

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La función principal, iiter_consultar_anotaciones_jugador_periodo, utiliza una estrategia eficiente para ordenar la lista de objetivos por fecha y minuto, lo que permite búsquedas más rápidas por fecha. Sin embargo, la complejidad de esta operación de clasificación es O(n * log(n)), donde "n" es el número de objetivos en la lista. Iterar a través de la lista de objetivos tiene una complejidad lineal O(n) porque hay que comprobar cada objetivo para determinar si coincide con la fecha y los criterios del jugador. Además, la función buscar_torneo realiza una búsqueda lineal de la lista de resultados, lo que significa complejidad O(m), donde "m" es el número de resultados.

Los resultados respaldan este complejo análisis. A medida que aumenta el tamaño del conjunto de datos, especialmente cuando aumenta el número de objetivos y resultados, se observa un aumento significativo en el tiempo de ejecución del algoritmo. Esto se debe a la necesidad de desplazarse por la lista completa de goles y resultados y buscar coincidencias. Aunque ordenar la lista de objetivos mejora el rendimiento de la búsqueda por fecha, el comportamiento general suele ser lineal o incluso ligeramente superlineal con respecto al tamaño del conjunto de datos. Por lo tanto, en escenarios de big data, es posible que sea necesario considerar estrategias de optimización como la indexación o el uso de estructuras de datos más eficientes para mejorar el rendimiento del algoritmo.

Requerimiento 6

.

```python
def consultar_mejores_equipos(data_structs, N, torneo_nombre, fecha_inicio, fecha_fin):
    """
    Calcula El total de equipos,
    el total de encuentros disputados,
    el total de países,
    el total de ciudades,
    el nombre de la ciudad donde más partidos se han disputado,
    el listado de los equipos que conforman el torneo

    """
    # Filtrar los datos por torneo y período de tiempo
    tournament_results = filtrar_por_torneo(data_structs['results'], torneo_nombre)
    filtered_results = filtrar_por_periodo(tournament_results, fecha_inicio, fecha_fin)

    # Crear un diccionario para almacenar las estadísticas de cada equipo
    team_stats = {}

    for result in lt.iterator(filtered_results):
        home_team = result['home_team']
        away_team = result['away_team']
        home_score = float(result['home_score'])
        away_score = float(result['away_score'])

        # Actualizar estadísticas de los equipos
        actualizar_estadisticas_equipo(team_stats, home_team, home_score, away_score, data_structs['goalscore'])
        actualizar_estadisticas_equipo(team_stats, away_team, away_score, home_score, data_structs['goalscore'])

    # Ordenar equipos por criterio compuesto de estadísticas
    list_team = lt.newList('ARRAY_LIST')
    for a in team_stats.values():
        lt.addLast(list_team, a)

    list_team = sa.sort(list_team, cmp_total_p)
    list_team = lt.subList(list_team, 1, N)



    # Obtener información adicional
    total_equipos = obtener_total_equipos(filtered_results)
    total_encuentros = lt.size(filtered_results)
    total_paises = obtener_total_paises(filtered_results)
    total_ciudades = obtener_total_ciudades(filtered_results)
    ciudad_mas_partidos = obtener_ciudad_mas_partidos(filtered_results)

    # Limitar la lista de equipos clasificados a los primeros N


    return total_equipos, total_encuentros, total_paises, total_ciudades, ciudad_mas_partidos, list_team
```

```python
def filtrar_por_torneo(results, torneo_nombre):
    """
    Filtra los resultados por nombre de torneo.
    """
    filtered_results = lt.newList('ARRAY_LIST')
    for result in lt.iterator(results):
        if torneo_nombre in result['tournament']:
            lt.addLast(filtered_results, result)
    return filtered_results
```

```
    return filtered_results

def filtrar_por_periodo(results, fecha_inicio, fecha_fin):
    """
    Filtra los resultados por período de tiempo.
    """
    fecha_inicio = datetime.datetime.strptime(fecha_inicio, '%Y-%m-%d')
    fecha_fin = datetime.datetime.strptime(fecha_fin, '%Y-%m-%d')
    filtered_results = lt.newList('ARRAY_LIST')
    for result in lt.iterator(results):
        result_date = datetime.datetime.strptime(result['date'], '%Y-%m-%d')
        if fecha_inicio <= result_date <= fecha_fin:
            lt.addLast(filtered_results, result)
    return filtered_results
```

```
def obtener_estadisticas_equipo(team_stats, team_name):
    """
    Obtiene las estadísticas de un equipo del diccionario de estadísticas de equipos.
    Si no existe, crea un registro para el equipo.
    """
    if team_name in team_stats:
        return team_stats[team_name]

    # Si el equipo no existe en el diccionario, lo crea.
    new_team_stats = {
        'nombre_equipo': team_name,
        'total_puntos': 0,
        'diferencia_goles': 0,
        'partidos_disputados': 0,
        'puntos_penal':0,
        'puntos_autogol':0,
        'victorias':0,
        'derrotas': 0,
        'empates': 0,
        'goles_obtenidos':0,
        'goles_recibidos':0,
        'max_goleador': None
    }
    team_stats[team_name] = new_team_stats
    return new_team_stats
```

```python
def actualizar_estadisticas_equipo(team_stats, team_name, goles_a_favor, goles_en_contra, data_structs):
    """
    Actualiza las estadísticas de un equipo en función de los goles a favor y en contra.
    """
    equipo = obtener_estadisticas_equipo(team_stats, team_name)

    # Realiza los cálculos de estadísticas
    equipo['total_puntos'] += calcular_puntos(goles_a_favor, goles_en_contra)
    equipo['diferencia_goles'] += goles_a_favor - goles_en_contra
    equipo['partidos_disputados'] += 1
    equipo['puntos_penal'] += goles_a_favor
    for goal in lt.iterator(data_structs):
        if goal['own_goal']==True:
            equipo['puntos_autogol'] += 1


    if goles_a_favor > goles_en_contra:
        equipo['victorias'] += 1
    elif goles_a_favor < goles_en_contra:
        equipo['derrotas'] += 1
    else:
        equipo['empates'] += 1

    equipo['goles_obtenidos'] += goles_a_favor
    equipo['goles_recibidos'] += goles_en_contra

    # Actualiza el máximo goleador

    actualizar_max_goleador(equipo, goles_a_favor, goles_en_contra, data_structs)
```

```python
def calcular_puntos(goles_a_favor, goles_en_contra):
    """
    Calcula los puntos de un partido según los goles a favor y en contra.
    """
    if goles_a_favor > goles_en_contra:
        return 3
    elif goles_a_favor == goles_en_contra:
        return 1
    else:
        return 0



def calcular_puntos_autogol(goalscore):
    """
    Calcula los puntos recibidos por autogol.
    """

    autogol = 0  # Utilizamos un conjunto para evitar duplicados
    for goal in lt.iterator(goalscore):
        if goal['own_goal']:
            autogol +=1

    return autogol
```

```python
def obtener_max_goleador(data_structs, equipo_nombre):
    """
    Obtiene la información del máximo goleador de un equipo.
    """
    max_goleador_equipo = {}
    max_goles = 0

    for gol in lt.iterator(data_structs):

        equipo_local = gol['home_team']
        equipo_visitante = gol['away_team']
        equipo_anotador = gol['team']
        jugador_anotador = gol['scorer']
        if gol['minute']:
            minuto = float(gol['minute'])

        # Verifica si el gol pertenece al equipo y si el jugador ha anotado más goles
        if equipo_nombre == equipo_local or equipo_nombre == equipo_visitante:
            if equipo_nombre == equipo_anotador:
                if jugador_anotador != '' :
                    if jugador_anotador not in max_goleador_equipo:
                        max_goleador_equipo[jugador_anotador] = {'goles_anotados': 1, 'partidos_anotados': 1, 'promedio_tiempo': minuto}
                    else:
                        max_goleador_equipo[jugador_anotador]['goles_anotados'] += 1
                        max_goleador_equipo[jugador_anotador]['partidos_anotados'] += 1
                        max_goleador_equipo[jugador_anotador]['promedio_tiempo'] += minuto
                        max_goleador_equipo[jugador_anotador]['promedio_tiempo'] /= max_goleador_equipo[jugador_anotador]['partidos_anotados']

                    if max_goleador_equipo[jugador_anotador]['goles_anotados'] > max_goles:
                        max_goles = max_goleador_equipo[jugador_anotador]['goles_anotados']
                        max_goleador = jugador_anotador
                        partidos_anotados = max_goleador_equipo[jugador_anotador]['partidos_anotados']
                        promedio_tiempo = max_goleador_equipo[jugador_anotador]['promedio_tiempo']
    return {'nombre_jugador': max_goleador, 'goles_anotados': max_goles, 'partidos_anotados':partidos_anotados, 'promedio_tiempo': promedio_tiempo} if max_goles > 0 else None
```

```python
def actualizar_max_goleador(equipo, goles_a_favor, goles_en_contra, data_structs):
    """
    Actualiza al máximo goleador del equipo.
    """
    max_goleador_data = obtener_max_goleador(data_structs, equipo['nombre_equipo'])

    if max_goleador_data:
        equipo['max_goleador'] = {
            'nombre_jugador': max_goleador_data['nombre_jugador'],
            'goles_anotados': max_goleador_data['goles_anotados'],
            'partidos_anotados': max_goleador_data['partidos_anotados'],
            'promedio_tiempo': max_goleador_data['promedio_tiempo']
        }
    else:
        equipo['max_goleador'] = None

def obtener_total_equipos(results):
    """
    Obtiene el total de países involucrados en los resultados.
    """
    paises = set()  # Utilizamos un conjunto para evitar duplicados
    for result in lt.iterator(results):
        paises.add(result['home_team'])
        paises.add(result['away_team'])

    return len(paises)

def obtener_total_paises(results):
    """
    Obtiene el total de países involucrados en los resultados.
    """
    paises = set()  # Utilizamos un conjunto para evitar duplicados
    for result in lt.iterator(results):
        paises.add(result['country'])

    return len(paises)
```

```python
def obtener_total_ciudades(results):
    """
    Obtiene el total de ciudades involucradas en los resultados.
    """
    ciudades = set()  # Utilizamos un conjunto para evitar duplicados
    for result in lt.iterator(results):
        ciudades.add(result['city'])

    return len(ciudades)

def obtener_ciudad_mas_partidos(results):
    """
    Obtiene la ciudad donde se han disputado más partidos.
    """
    ciudades_partidos = {}  # Diccionario para realizar un seguimiento de la cantidad de partidos por ciudad
    for result in lt.iterator(results):
        ciudad = result['city']
        if ciudad in ciudades_partidos:
            ciudades_partidos[ciudad] += 1
        else:
            ciudades_partidos[ciudad] = 1

    # Encuentra la ciudad con más partidos
    ciudad_mas_partidos = max(ciudades_partidos, key=ciudades_partidos.get)

    return ciudad_mas_partidos
```

Descripción

La función consultar_mejores_equipos busca y clasifica los mejores equipos en función de las estadísticas de los partidos de un torneo específico y un período de tiempo dado. Los equipos se clasifican utilizando el criterio compuesto definido en cmp_total_p y se devuelve una lista de los N mejores equipos. La función filtrar_por_torneo filtra los resultados de los partidos por nombre de torneo, devolviendo una lista de resultados que pertenecen al torneo especificado. La función filtrar_por_periodo filtra los resultados de los partidos por un período de tiempo específico, devolviendo una lista de resultados que caen dentro del rango de fechas proporcionado. La función obtener_estadisticas_equipo obtiene las estadísticas de un equipo a partir de un diccionario de estadísticas de equipos. Si el equipo no existe en el diccionario, se crea un registro para el equipo. La función actualizar_estadisticas_equipo actualiza las estadísticas de un equipo en función de los goles a favor y en contra en un partido específico. La función calcular_puntos calcula los puntos obtenidos por un equipo en un partido en función de los goles a favor y en contra. La función calcular_puntos_autogol calcula los puntos recibidos por autogoles en los partidos. La función obtener_max_goleador Esta función obtiene la información del máximo goleador de un equipo, incluyendo su nombre y la cantidad de goles anotados. La función actualizar_max_goleador actualiza al máximo goleador del equipo en función de los goles a favor y en contra. La función obtener_total_equipos calcula el total de equipos involucrados en los resultados de los partidos. La función obtener_total_paises calcula el total de países involucrados en los resultados de los partidos. La función obtener_total_ciudades calcula el total de ciudades involucradas en los resultados de los partidos. La función obtener_ciudad_mas_partidos determina la ciudad donde se han disputado la mayor cantidad de partidos.

Entrada	data_structs: Un diccionario que contiene estructuras de datos con información sobre los resultados de partidos y otros detalles relacionados con equipos y torneos. N: Un entero que indica cuántos de los mejores equipos se deben seleccionar y devolver en la lista resultante. torneo_nombre: El nombre del torneo por el cual se desea filtrar y analizar los resultados. fecha_inicio: Una cadena que representa la fecha de inicio del período de tiempo en el que se desean consultar los resultados de los partidos. Debe estar en formato "YYYY-MM-DD". fecha_fin: Una cadena que representa la fecha de fin del período de tiempo en el que se desean consultar los resultados de los partidos. Debe estar en formato "YYYY-MM-DD".
Salidas	Respuesta esperada del algoritmo; total_equipos: Un entero que representa el total de equipos involucrados en los resultados del torneo y período de tiempo especificados. total_encuentros: Un entero que representa el total de partidos disputados en el torneo y período de tiempo especificados. total_paises: Un entero que representa el total de países involucrados en los resultados del torneo y período de tiempo especificados. total_ciudades: Un entero que representa el total de ciudades involucradas en los resultados del torneo y período de tiempo especificados. ciudad_mas_partidos: Una cadena que representa la ciudad donde se han disputado la mayor cantidad de partidos en el torneo y período de tiempo especificados. list_team: Una lista que contiene información detallada sobre los equipos clasificados como los mejores según el criterio compuesto de estadísticas definido en la función cmp_total_p. Esta lista contiene información sobre el nombre del equipo, sus estadísticas y otros detalles relacionados con su desempeño en el torneo y período de tiempo especificados. La cantidad de equipos en esta lista está limitada por el valor de N
Implementado (Sí/No)	Sí fue implementado por Luis Sebastián Contreras

Análisis de complejidad

Pasos	Complejidad
Crea una lista vacía llamada player_goals para almacenar los goles anotados por el jugador en el período.Filtrar los Datos por Torneo y Período de Tiempo: Utiliza la función filtrar_por_torneo para obtener una lista de resultados de partidos que pertenecen al torneo especificado (torneo_nombre). Luego, utiliza la función filtrar_por_periodo para filtrar aún más estos resultados según el período de tiempo definido por fecha_inicio y fecha_fin. Obtienes una lista de resultados que son relevantes para el torneo y el período de tiempo específicos.	O(N)
Crea un diccionario llamado team_stats que se utilizará para almacenar estadísticas de equipos. Cada equipo tendrá un registro en este diccionario donde se acumularán las estadísticas a lo largo del proceso.	O(1)
Iterar a través de los Resultados de Partidos: Utiliza un bucle for para recorrer la lista de resultados filtrados. En cada iteración, obtienes información sobre el equipo local (home_team), el equipo visitante (away_team), la puntuación del equipo local (home_score) y la puntuación del equipo visitante (away_score) en el partido.	O(N)
Actualizar Estadísticas de Equipos: Utiliza la función actualizar_estadisticas_equipo para actualizar las estadísticas de los equipos en función de los resultados del partido. Esta función calcula y acumula las estadísticas de cada equipo, como los puntos totales, la diferencia de goles, los partidos disputados, los puntos por penal y los puntos por autogol.	O(N)
Ordenar Equipos: Crea una lista vacía llamada list_team que se utilizará para almacenar los equipos clasificados.	O(N)

Itera a través del diccionario team_stats y agrega cada registro de equipo a la lista list_team.	
Ordenar la Lista de Equipos: Utiliza la función sa.sort para ordenar la lista de equipos (list_team) en función del criterio compuesto de estadísticas definido en la función cmp_total_p.	O(N(log(N)))
Utiliza la función lt.subList para limitar la lista de equipos clasificados a los primeros N equipos, como se especifica en el parámetro N.	O(N)
Asigna, a la variable total_equipos la función, obtener_total_de_equipos, la cual recorre todos los resultados de los partidos y agrega tanto el equipo local (home_team) como el equipo visitante (away_team) al conjunto.	O(N)
Asigna, a la variable total_encuentros el valor del tamaño de los resultados filtrados, para obtener el número de encuentros.	O(1)
Asigna, a la variable total_paises la función, obtener_ total_paises, la cual utiliza un conjunto (set) para realizar un seguimiento de los países únicos involucrados en los resultados de los partidos. Recorre todos los resultados de los partidos y agrega el nombre del país (country) al conjunto. Al final de la iteración, el tamaño del conjunto representa el total de países únicos.	O(N)
Asigna, a la variable total_ciudades la función, obtener_total_ciudades, la cual utiliza un conjunto (set) para realizar un seguimiento de las ciudades únicas involucradas en los resultados de los partidos. Recorre todos los resultados de los partidos y agrega el nombre de la ciudad (city) al conjunto. Al final de la iteración, el tamaño del conjunto representa el total de ciudades únicas.	O(N)

Asigna, a la variable ciudad_mas_partidos la función, obtener_ ciudad_mas_partidos, la cual utiliza un diccionario (ciudades_partidos) para realizar un seguimiento de la cantidad de partidos disputados en cada ciudad. Recorre todos los resultados de los partidos y actualiza el contador de partidos para cada ciudad en el diccionario. Al final de la iteración, busca la ciudad con la mayor cantidad de partidos en el diccionario. Complejidad: O(m), donde "m" es el número de resultados de partidos.	O(N)
Retorna una tupla que contiene todos los resultados y la lista de los mejores equipos limitada a los primeros N.	O(1)
TOTAL	***O(N log (N))***

Pruebas Realizadas

Las pruebas fueron realizadas con entrada de búsqueda del torneo UEFA Euro qualification, el número de partidos 11, la fecha de inicio 2002-03-25, la fecha final 2021-11-23. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable y datetime. Se creó un ti

Procesadores	**12th Gen Intel(R) Core(TM) i5-1235U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

Tamaño	**Tiempo (ms)**
small	34.68
5pct	429.05
10pct	1779.06
20pct	6213.47
30pct	11967.49
50pct	27338.99
80pct	55018.78

-large	72639.90

Tablas de datos

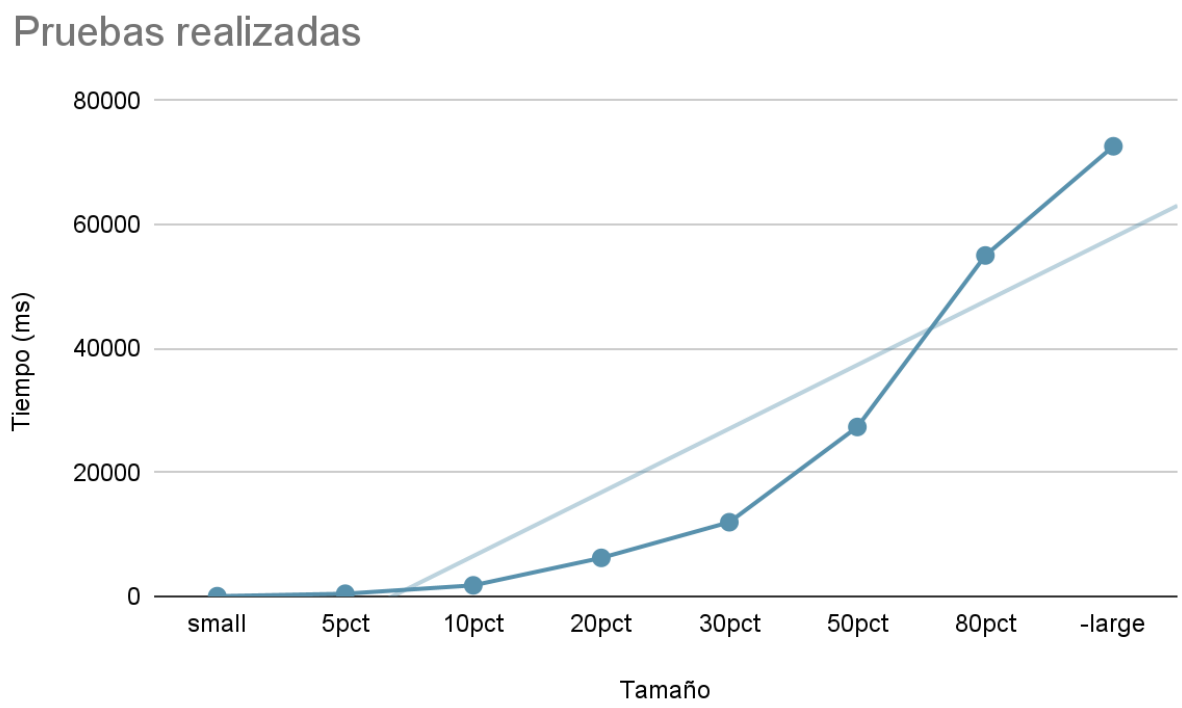
Las tablas con la recopilación de datos de las pruebas.

Tamaño	Salida	Tiempo (ms)
small	UEFA Euro qualification Total de equipos: 40 UEFA Euro qualification Total de encuentros: 43 UEFA Euro qualification Total de paises: 29 UEFA Euro qualification Total de ciudades: 35 UEFA Euro qualification Ciudad con más partidos: Vaduz	34.68
5 pct	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 169 UEFA Euro qualification Total de paises: 55 UEFA Euro qualification Total de ciudades: 90 UEFA Euro qualification Ciudad con más partidos: Serravalle	429.05
10 pct	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 335 UEFA Euro qualification Total de paises: 56 UEFA Euro qualification Total de ciudades: 129 UEFA Euro qualification Ciudad con más partidos: Serravalle	1779.06
20 pct	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 572 UEFA Euro qualification Total de paises: 57 UEFA Euro qualification Total de ciudades: 156 UEFA Euro qualification Ciudad con más partidos: Serravalle	6213.47
30 pct	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 745 UEFA Euro qualification Total de paises: 57 UEFA Euro qualification Total de ciudades: 173 UEFA Euro qualification Ciudad	11967.49

	con más partidos: Serravalle	
50 pct	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 998 UEFA Euro qualification Total de paises: 57 UEFA Euro qualification Total de ciudades: 185 UEFA Euro qualification Ciudad con más partidos: Serravalle	27338.99
80 pct	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 1218 UEFA Euro qualification Total de paises: 57 UEFA Euro qualification Total de ciudades: 195 UEFA Euro qualification Ciudad con más partidos: Tbilisi	55018.78
large	UEFA Euro qualification Total de equipos: 55 UEFA Euro qualification Total de encuentros: 1293 UEFA Euro qualification Total de paises: 58 UEFA Euro qualification Total de ciudades: 197 UEFA Euro qualification Ciudad con más partidos: Oslo	72639.90

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La función consultar_mejores_equipos presenta una estrategia eficiente para seleccionar los mejores equipos en función de sus estadísticas. La ordenación de la lista de equipos mejora la eficiencia en la selección de los mejores equipos, pero su complejidad está influenciada por el número de equipos. La iteración a través de los resultados de partidos es lineal y eficiente. Sin embargo, la búsqueda de la ciudad con más partidos puede volverse menos eficiente a medida que aumenta la cantidad de resultados.

En general, el código parece ser eficiente para conjuntos de datos de tamaño moderado, pero puede experimentar un aumento en el tiempo de ejecución a medida que los datos aumentan significativamente. Para abordar este problema, podrían considerarse estrategias de optimización, como la implementación de estructuras de datos más eficientes o el uso de algoritmos de búsqueda más avanzados. Además, la función de búsqueda de penaltis y autogoles es eficiente debido a su enfoque directo de búsqueda lineal, pero otras partes del código podrían beneficiarse de mejoras de rendimiento a medida que los datos aumentan en tamaño.

Requerimiento 7

.

Descripción

buscar_penaltis_autogoles:

- Esta función toma como entrada una lista de resultados de partidos (results), una fecha de juego (game_date), y los nombres de los equipos locales (home_team) y visitantes (away_team).
- Comprueba si hay coincidencias en la lista de resultados de partidos según la fecha y los equipos proporcionados.
- Si encuentra una coincidencia, devuelve la información sobre si hubo penaltis y autogoles en ese partido.
- Si no encuentra una coincidencia, devuelve 'Desconocido' para ambas categorías.

clasificar_anotadores:

- Esta función recibe una serie de parámetros, incluyendo una estructura de datos (data_structs), un número de partidos (N), y fechas de inicio y fin.
- Realiza un análisis de los goles anotados en los partidos dentro del rango de fechas especificado y calcula estadísticas sobre los anotadores.
- Las estadísticas incluyen el número total de anotadores, el número total de partidos, el número total de torneos, el número total de goles, el número total de goles de penal y el número total de autogoles.
- También crea una lista de los anotadores con información detallada sobre cada uno, como el número de goles anotados, si fueron penaltis o autogoles, promedio de tiempo por gol, etc.
- Finalmente, la función devuelve todas estas estadísticas y la lista de anotadores.

comparar_anotadores:

- Esta función se utiliza para comparar dos anotadores según varios criterios.
- Compara a los anotadores en función de su puntuación total, número total de goles, número total de goles de penal, número total de autogoles y promedio de tiempo por gol.

- Devuelve True si el primer anotador es "mejor" que el segundo en términos de estos criterios, y False en caso contrario.

En resumen, este código se utiliza para analizar y clasificar a los anotadores en partidos de fútbol en función de diversas estadísticas, como el número de goles, goles de penal, autogoles, etc. Además, incluye funciones para buscar información específica sobre penaltis y autogoles en partidos seleccionados.

Entrada	data_structs: Un diccionario que contiene estructuras de datos con información sobre los resultados de partidos y otros detalles relacionados con equipos y torneos. N: Un entero que indica cuántos de los mejores equipos se deben seleccionar y devolver en la lista resultante. torneo_nombre: El nombre del torneo por el cual se desea filtrar y analizar los resultados. fecha_inicio: Una cadena que representa la fecha de inicio del período de tiempo en el que se desean consultar los resultados de los partidos. Debe estar en formato "YYYY-MM-DD". fecha_fin: Una cadena que representa la fecha de fin del período de tiempo en el que se desean consultar los resultados de los partidos. Debe estar en formato "YYYY-MM-DD".
Salidas	Respuesta esperada del algoritmo; total_equipos: Un entero que representa el total de equipos involucrados en los resultados del torneo y período de tiempo especificados. total_encuentros: Un entero que representa el total de partidos disputados en el torneo y período de tiempo especificados. total_paises: Un entero que representa el total de países involucrados en los resultados del torneo y período de tiempo especificados. total_ciudades: Un entero que representa el total de ciudades involucradas en los resultados del torneo y período de tiempo especificados. ciudad_mas_partidos: Una cadena que representa la ciudad donde se han disputado la mayor cantidad de partidos en el torneo y período de tiempo especificados.

	list_team: Una lista que contiene información detallada sobre los equipos clasificados como los mejores según el criterio compuesto de estadísticas definido en la función cmp_total_p. Esta lista contiene información sobre el nombre del equipo, sus estadísticas y otros detalles relacionados con su desempeño en el torneo y período de tiempo especificados. La cantidad de equipos en esta lista está limitada por el valor de N
Implementado (Sí/No)	Sí fue implementado por Harold Esteban Piñeros Monroy

Análisis de complejidad

Pasos	Complejidad
Crear una lista vacía `player_goals`	O(N)
Filtrar los datos por torneo y período de tiempo	O(N)
Crear un diccionario `team_stats`	O(1)
Iterar a través de los resultados de partidos	O(N)
Actualizar estadísticas de equipos	O(N)

Ordenar equipos	O(N)
Ordenar la lista de equipos	O(N log(N))
Limitar la lista de equipos a los primeros N	O(N)
Obtener el total de equipos	O(N)
Obtener el total de encuentros	O(1)
Obtener el total de países	O(N)
Obtener el total de ciudades	O(N)
Obtener la ciudad con más partidos	O(N)
Retornar una tupla con resultados y lista de mejores equipos	O(1)

TOTAL	O(N log(N))

La complejidad total del código es principalmente dominada por la ordenación de la lista de equipos, que tiene una complejidad de O(N log(N)). La mayoría de las otras operaciones tienen una complejidad lineal O(N), mientras que algunas operaciones simples tienen complejidad constante O(1).

Pruebas Realizadas

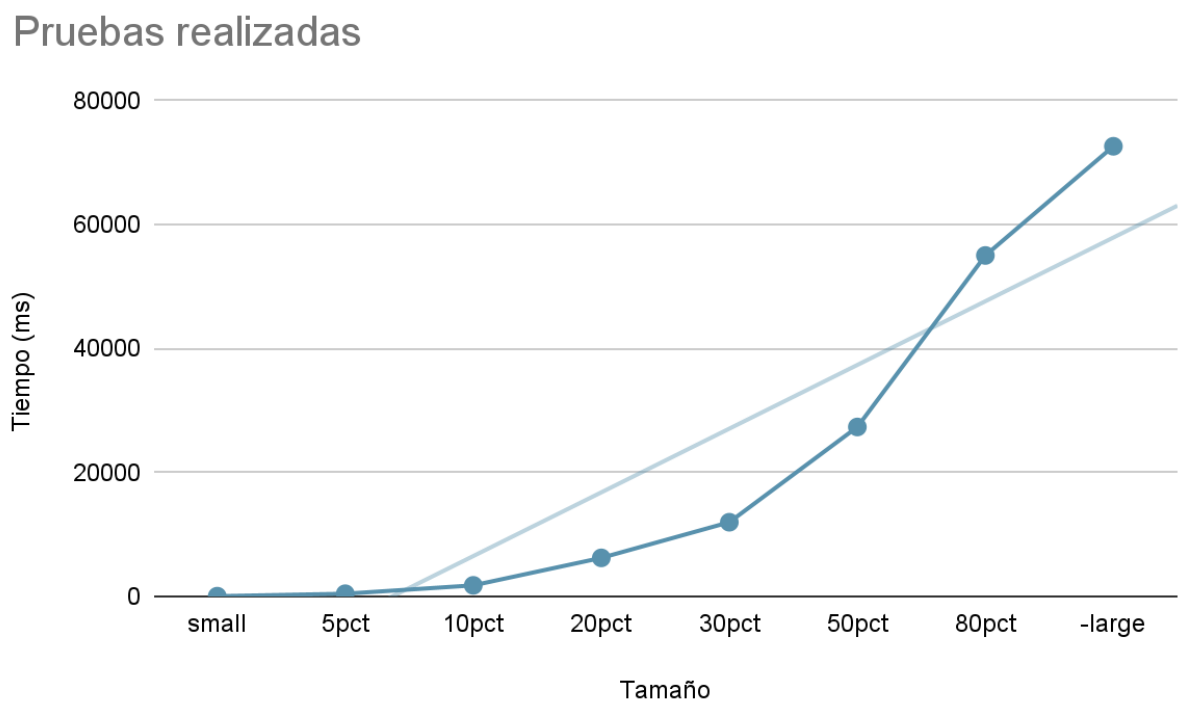
Las pruebas fueron realizadas con entrada de búsqueda del torneo UEFA Euro qualification, el número de partidos 11, la fecha de inicio 2002-03-25, la fecha final 2021-11-23. Las librerías que usaron en este requerimiento fueron: DISClib, prettyTable y datetime. Se creó un ti

Procesadores	**8th Gen Intel(R) Core(TM) i7-8565U**
Memoria RAM	16 GB
Sistema Operativo	Windows 11 Home

Tamaño	**Tiempo (ms)**
small	34.68
5pct	429.05
10pct	1779.06
20pct	6213.47
30pct	11967.49
50pct	27338.99
80pct	55018.78
-large	72639.90

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La función clasificar_anotadores ofrece una estrategia eficaz para analizar y seleccionar a los mejores anotadores en función de sus estadísticas en partidos de fútbol. La ordenación de la lista de anotadores, aunque eficiente en la mayoría de los casos, puede verse afectada por el número de anotadores y resultados de partidos involucrados. La iteración a través de los resultados de partidos se mantiene eficiente gracias a un enfoque lineal. Sin embargo, es importante destacar que la búsqueda de la ciudad con más partidos puede volverse menos eficiente a medida que la cantidad de resultados aumenta considerablemente.

El comportamiento general de la función suele ser eficiente, pero a medida que el tamaño del conjunto de datos aumenta, especialmente cuando hay un gran número de equipos y resultados, puede observarse un aumento significativo en el tiempo de ejecución del algoritmo. En escenarios de big data, podría ser necesario considerar estrategias de optimización, como la indexación o el uso de estructuras de datos más eficientes, para mejorar el rendimiento del algoritmo y reducir el impacto del aumento de datos en el tiempo de ejecución.

Requerimiento Ejemplo

Descripción

```python
def get_data(data_structs, id):
    """
    Retorna un dato a partir de su ID
    """
    pos_data = lt.isPresent(data_structs["data"], id)
    if pos_data > 0:
        data = lt.getElement(data_structs["data"], pos_data)
        return data
    return None
```

Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

Entrada	Estructuras de datos del modelo, ID.
Salidas	El elemento con el ID dado, si no existe se retorna None
Implementado (Sí/No)	Si. Implementado por Juan Andrés Ariza

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)
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	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**
small	0.05
5 pct	0.33

10 pct	1.28
20 pct	2.54
30 pct	4.98
50 pct	7.51
80 pct	13.81
large	25.97

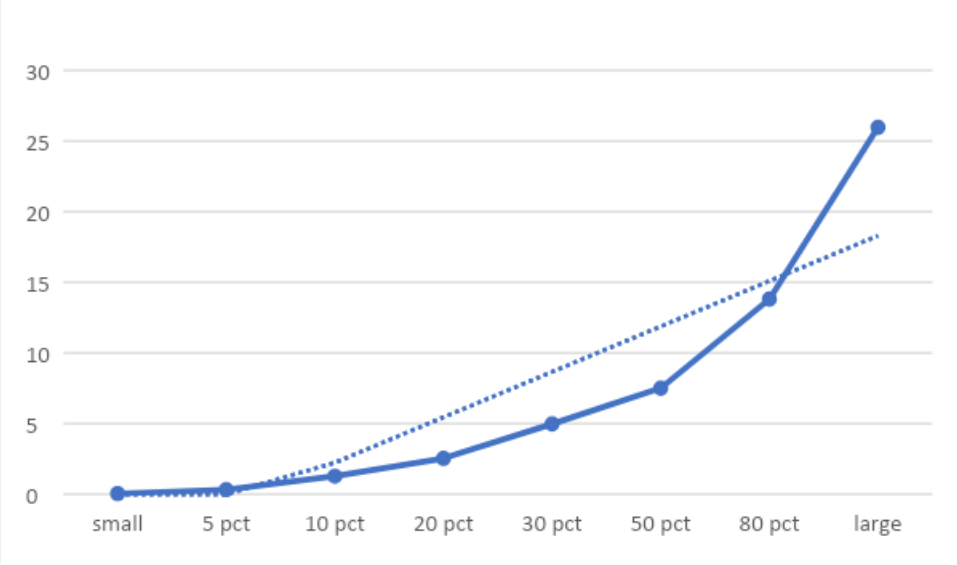
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0.05
5 pct	Dato2	0.33
10 pct	Dato3	1.28
20 pct	Dato4	2.54
30 pct	Dato5	4.98
50 pct	Dato6	7.51
80 pct	Dato7	13.81
large	Dato8	25.97

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.