ANÁLISIS DEL RETO

Andrés Romero, af.romerop1@uniandes.edu.co, 202312399

Juan José Penha, J.penha@uniandes.edu.co, 202312307

Jerónimo Vásquez, j.vasquezp2@uniandes.edu.co, 202223824

Requerimiento <<1>>

Descripción

```python
def req_1(control,numero_partidos:int, nombre_equipo: str, condicion_equipo:int):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1

    if condicion_equipo == 1:
        control_r = control['hteam']
        map = mp.get(control_r,nombre_equipo)
        lista = me.getValue(map)
    if condicion_equipo == 2:
        control_r = control['ateam']
        map = mp.get(control_r,nombre_equipo)
        lista = me.getValue(map)

    quk.sort(lista,compareDates)


    return lista
```

Este es el requerimiento uno que busca mostrar los últimos N de cierto equipo, donde se filtra si dicho equipo jugó de local o visitante, todo se realiza mediante comparaciones haciendo que su complejidad no incremente

Entrada	Control, numero partidos, nombre equipo, condición equipo
Salidas	Lista
Implementado (Sí/No)	Si, implementado por Jerónimo Vázquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Comparación de los datos de entrada con los mapas	O(1)
Implementación de Quick sort	O(n log n)
TOTAL	***O(n log 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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	**Tiempo (ms)**
small	0.9
5 pct	1.98
10 pct	3.29
20 pct	6.44
30 pct	8.61
50 pct	14.84
80 pct	24.89
large	30.36

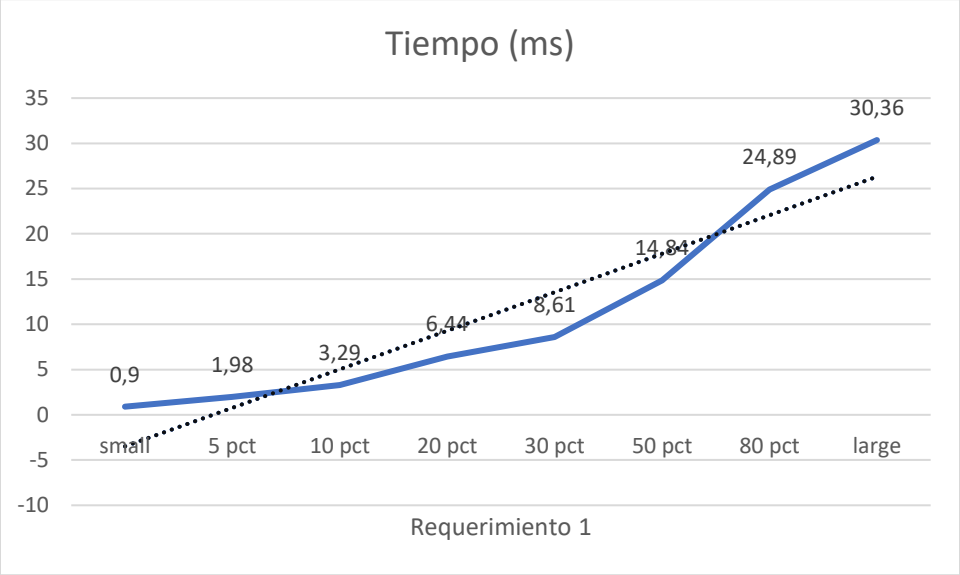
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	0.9
5 pct	Dato2	1.98
10 pct	Dato3	3.29
20 pct	Dato4	6.44
30 pct	Dato5	8.61
50 pct	Dato6	14.84
80 pct	Dato7	24.89
large	Dato8	30.36

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad del requerimiento es constante, donde su complejidad es O (n log n), ya que se está observando el peor caso para el Quick sort en donde el pivote quede completamente desordenado haciendo que se cree la mayor cantidad posible de pivotes para ordenar el resultado. Este algoritmo de ordenamiento fue necesario para organizar los datos en orden cronológico. Finalmente, podemos saber que la complejidad es constante debido a que las diferentes entradas (desde small hasta large) hicieron que los tiempos de ejecución aumentaran de una manera casi lineal, haciendo que los puntos no sean dispersos con respecto a la línea de tendencia lineal.

Requerimiento <<2>>

Descripción

```
def req_2(control,goles,goleador):
    """
    Función que soluciona el requerimiento 2
    """
        # TODO: Realizar el requerimiento 2
    r=[]
    controlg=control["goleadores_m"]
    map=mp.get(controlg,goleador)

    lista= me.getValue(map)
    quk.sort(lista,compareDates)
    return lista
```

Este es el requerimiento 2 en donde se busca toda la información relacionada con N cantidad de goles de cierto jugador, en este requerimiento solo se toma el mapa controlg el tiene como etiqueta el nombre del jugador y se delimita por la cantidad de goles.

Entrada	Control, goles, goleador
Salidas	Lista
Implementado (Sí/No)	Si, implementado por Juan José Penha

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Acceso en el diccionario para sacar la información	O(1)
Implementación de Quick sort	O(n log n)
TOTAL	***O(n log 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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	Tiempo (ms)
small	0.12
5 pct	0.12
10 pct	0.19
20 pct	0.45
30 pct	0.64
50 pct	0.71
80 pct	0.54
large	1.01

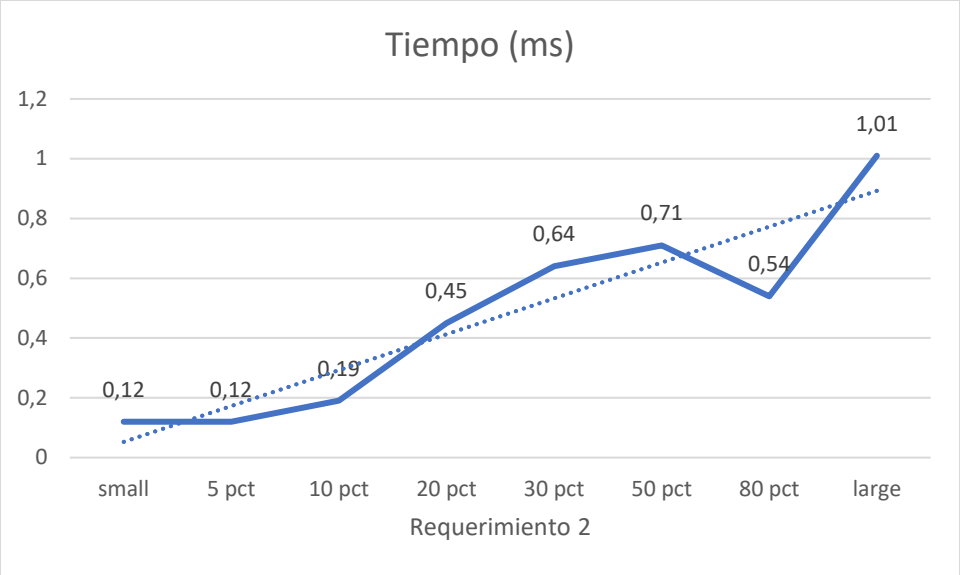
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0.12
5 pct	Dato2	0.12
10 pct	Dato3	0.19
20 pct	Dato4	0.45
30 pct	Dato5	0.64
50 pct	Dato6	0.71
80 pct	Dato7	0.54
large	Dato8	1.01

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad del requerimiento es constante, donde su complejidad es O (n log n), ya que se está observando el peor caso para el Quick sort en donde el pivote quede completamente desordenado haciendo que se cree la mayor cantidad posible de pivotes para ordenar el resultado. Este algoritmo de ordenamiento fue necesario para organizar los datos en orden cronológico. Finalmente, este requerimiento cambió ligeramente puesto que, en su 80pct, disminuyó el tiempo de ejecución, esto puede tratarse solo de la maquina y que en ese ensayo logró medir menos que su versión mas ligera, se puede evidencias que en este requerimiento todos los datos son bastante cercanos a comparación del requerimiento 1, esto se debe a que no se realizan comparaciones, solo se hace un llamado a un diccionario y se extrae cierta información del mismo.

Requerimiento <<3>>

Descripción

```python
def req_3(control, nombre_equipo, fecha_final, fecha_inicial):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3

    filtro = lt.newList('ARRAY_LIST')
    respuesta = lt.newList('ARRAY_LIST')

    control_h = control['hteam']
    control_a = control['ateam']
    control_t = control['penalty']

    map_h = mp.get(control_h, nombre_equipo)
    map_a = mp.get(control_a, nombre_equipo)
    mapa = union(map_h, map_a) ; del mapa[0]; del mapa[1]
    for f in mapa:
        for j in f['elements']:
            lt.addLast(filtro,j)

    respuesta = filtrar_fechas(filtro['elements'], fecha_inicial, fecha_final)

    return respuesta, len(filtro['elements']), len(respuesta['elements']), len(map_h), len(map_a)
```

Este es el requerimiento 3 el cual busca mostrar los partidos que se jugaron en un rango de tiempo para cierto equipo, para ello, se sacan 3 mapas, dos de ellos son necesarios para saber si el equipo solicitado está en away team o home team y el ultimo mapa sirve para poder agregar las columnas faltantes para poder dar toda la información necesaria para el requerimiento. Se usaron dos funciones externas, “union” y “filtrar_fechas” estas dos funciones son necesarias para 1. Unir control_h con control_a y 2. Para comparar las fechas, tanto fecha inicial como final.

Entrada	Control, nombre equipo, fecha inicial, fecha final
Salidas	respuesta
Implementado (Sí/No)	Si, implementado por Andrés Romero

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
creación de lista	O (1)
Acceso a diccionarios	O (1)
Funciones .get	O (1)
Primero bucle for	O (N)
Función “filtrar_fechas”	O (M)
Calcular longitudes de la lista	O (1)
TOTAL	**O (N) + O(M)**

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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	**Tiempo (ms)**
small	9.81
5 pct	11.95
10 pct	15.4
20 pct	23.26
30 pct	27.86
50 pct	34.54
80 pct	43.92
large	52.43

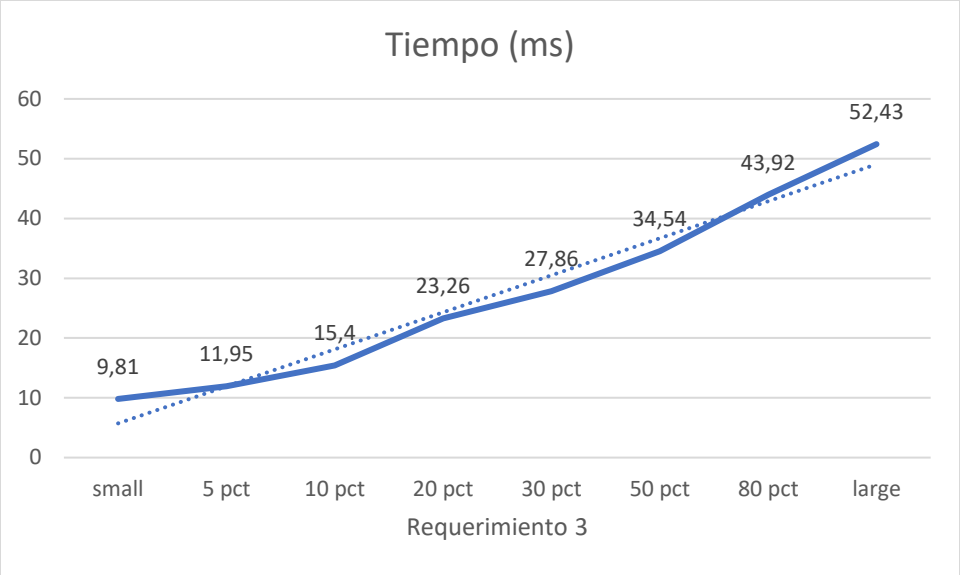
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	9.81
5 pct	Dato2	11.95
10 pct	Dato3	15.4
20 pct	Dato4	23.26
30 pct	Dato5	27.86
50 pct	Dato6	34.54
80 pct	Dato7	43.92
large	Dato8	52.43

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad del requerimiento es constante, donde su complejidad es O (n) + O (m), se puede ver lo constante que es la gráfica puesto que la desviación estándar no es alta, todos los puntos de la gráfica están cerca de la misma línea de tendencia lo que demuestra que esta función es linear y es directamente proporcional el tiempo de carga con las entradas.

Requerimiento <<4>>

Descripción

```
def req_4(control, nombre_torneo,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    r = []
    control_r = control['torneo']
    control_t = control['tiros']

    map = mp.get(control_r, nombre_torneo)
    lista = me.getValue(map)
    quk.sort(lista,compareDates)

    for i in lista['elements']:
        if int(i['date'][:4]) == int(fecha_inicial[:4]) and int(i['date'][:4]) == int(fecha_final[:4]):
            if int(i['date'][5:7]) == int(fecha_inicial[5:7]) and int(i['date'][5:7]) == int(fecha_final[5:7]):
                if int(i['date'][8:10]) >= int(fecha_inicial[8:10]) and int(i['date'][8:10]) <= int(fecha_final[8:10]):
                        r.append(i)
            elif int(i['date'][5:7]) >= int(fecha_inicial[5:7]) and int(i['date'][5:7]) <= int(fecha_final[5:7]):
                        r.append(i)
        elif int(i['date'][:4]) >= int(fecha_inicial[:4]) and int(i['date'][:4]) <= int(fecha_final[:4]):
                        r.append(i)

    for e in r:
        for fila in control_t['elements']:
            if e['date'] == fila['date'] and (e['home_team'] == fila['home_team'] or e['away_team'] == fila['away_team']) :
                e['winner'] = fila['winner']
        if "winner" not in e.keys():
            e['winner'] = "Desconocido"


    return r
```

Este es el requerimiento 4 el cual brinda toda la información para saber todos los partidos de un torneo para cierto rango de tiempo, para tal necesitamos dos mapas los cuales tienen como tags torneo y tiros, estos dos mapas son necesarios para poder sacar toda la información necesaria. Primero se toma la comparación de fechas para luego hacer un if el cual compare fechas, y el nombre del torneo. Finalmente, para agregar toda la información, necesitábamos recorrer 2 csv, para tal fin hacemos un ultimo if el cual nos ayuda a filtrar la misma información (esto se logra mediante if que consulte home_team, away_team y date)

Entrada	Control, nombre torneo, fecha inicial, fecha final
Salidas	R (es una lista)
Implementado (Sí/No)	Si, implementado por Jerónimo Vázquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
creación de lista	O (1)
Acceso a diccionarios	O (1)
Funciones .get	O (1)
Primero bucle for	O (N)
Segundo buble for	O (N^2)
Condicional if [“winner”]	O (1)
TOTAL	**O (N^2)**

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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	**Tiempo (ms)**
small	0.42
5 pct	0.94
10 pct	1.92
20 pct	4.16
30 pct	6.23
50 pct	10.24
80 pct	12.26
large	14.93

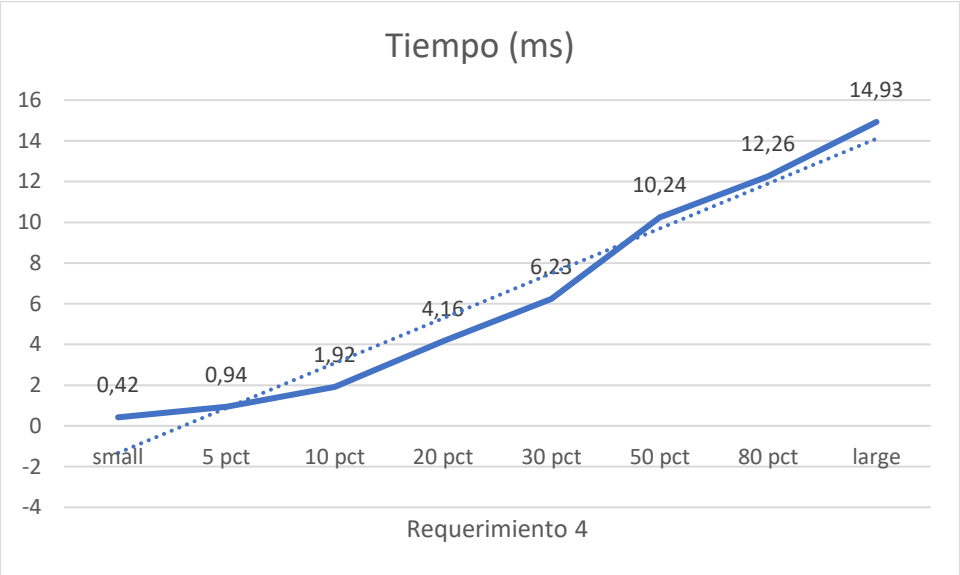
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0.42
5 pct	Dato2	0.94
10 pct	Dato3	1.92
20 pct	Dato4	4.16
30 pct	Dato5	6.23
50 pct	Dato6	10.24
80 pct	Dato7	12.26
large	Dato8	14.93

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad del requerimiento es acorde a la complejidad descrita previamente (O (N^2)) esto se evidencia porque el incremento es casi exponencial, lo cual cumple con su complejidad en el peor de los casos, esta complejidad se debe a que hay bucles anidados que hace que el programa recorra n veces un recorrido que ya buscaba n veces (Es por esto por lo que es n x n). Los datos están bastante cerca de la línea de tendencia, dándole así exactitud y constancia al requerimiento

Requerimiento <<5>>

Descripción

```
def req_5(control,nombre_jugador,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    r=[]
    controlg=control["goleadores_m"]
    controlr=control["resultados"]
    map = mp.get(controlg, nombre_jugador)
    lista = me.getValue(map)
    quk.sort(lista,compareDates)
    for i in lista["elements"]:
        if int(i['date'][:4]) == int(fecha_inicial[:4]) and int(i['date'][:4]) == int(fecha_final[:4]):
            if int(i['date'][5:7]) == int(fecha_inicial[5:7]) and int(i['date'][5:7]) == int(fecha_final[5:7]):
                if int(i['date'][8:10]) >= int(fecha_inicial[8:10]) and int(i['date'][8:10]) <= int(fecha_final[8:10]):
                        r.append(i)
            elif int(i['date'][5:7]) >= int(fecha_inicial[5:7]) and int(i['date'][5:7]) <= int(fecha_final[5:7]):
                        r.append(i)
        elif int(i['date'][:4]) >= int(fecha_inicial[:4]) and int(i['date'][:4]) <= int(fecha_final[:4]):
                        r.append(i)
    for e in r:
        for fila in controlr['elements']:
            if e['date'] == fila['date'] and (e['home_team'] == fila['home_team'] or e['away_team'] == fila['away_team']) :
                e['tournament'] = fila['tournament']
                e['home_score'] = fila['home_score']
                e['away_score'] = fila['away_score']



    return r
```

Esta es el requerimiento 5 el cual mantiene una estructura parecida a los anteriores requerimientos, puesto que este también busca delimitar toda la información de 2 mapas en un rango de fechas y el nombre del jugador; se necesita el segundo mapa (el mapa de resultados) para agregar la información: “tournament”, “home_score” y “away_score”.

Entrada	Control, nombre_jugador, fecha_inicial, fecha_final
Salidas	lista
Implementado (Sí/No)	Si, implementado por Juan José Penha

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Acceso a diccionarios	O (1)
Funciones. get	O (1)
Llamadas a quick sort	O (1)
Primero bucle for	O (N)
Segundo buble for	O (N^2)
Condicional if [“winner”]	O (1)
TOTAL	**O (N^2)**

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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	**Tiempo (ms)**
small	1.28
5 pct	8.31
10 pct	26.56
20 pct	98.22
30 pct	240.41
50 pct	470.76
80 pct	907.13
large	1151.44

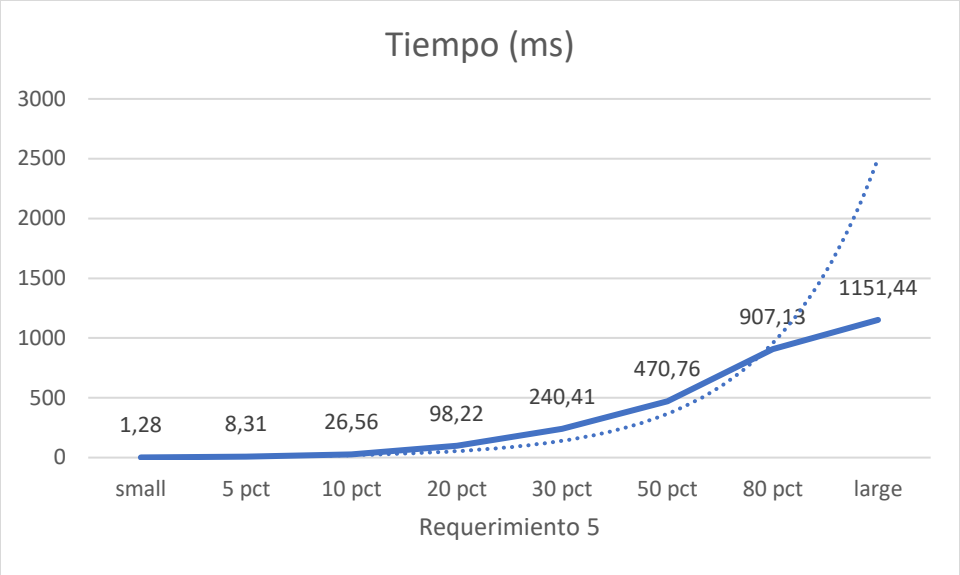
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	1.28
5 pct	Dato2	8.31
10 pct	Dato3	26.56
20 pct	Dato4	98.22
30 pct	Dato5	240.41
50 pct	Dato6	470.76
80 pct	Dato7	907.13
large	Dato8	1151.44

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad de este requerimiento es exponencial, esto se logra apreciar a que los tiempos de ejecución cargan de manera exponencial, esto se debe a que se utiliza un ciclo anidado dentro de otro ciclo, haciendo que la función itere n x n veces para poder agregar los elementos del mapa cuyo csv es resultados; la complejidad vista en la grafica va acorde al valor teórico de la misma, puesto que entre mas largo sea el archivo mas grande será el n por el cual tiene que iterar el bucle inicial para agregar toda la información necesaria.

Requerimiento <<6>>

Descripción

```
def req_6(control, nombre_torneo, anio):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    control_t = control['torneo']
    control_r = control['resultados']
    control_g = control['goleadores']
    control_ti = control['tiros']

    respuesta = lt.newList("ARRAY_LIST")
    equipos = lt.newList("ARRAY_LIST")
    paises = lt.newList("ARRAY_LIST")
    ciudades = {}
    total_equipos = lt.newList("ARRAY_LIST")

    lista = mp.get(control_t, nombre_torneo)
    torneo = me.getValue(lista)

    for i in lt.iterator(torneo):
        if int(i['date'][:4]) == anio:
            lt.addLast(respuesta,i)

    for e in lt.iterator(respuesta):
        for fila in lt.iterator(control_ti):
            if e['date'] == fila['date'] and (e['home_team'] == fila['home_team'] or e['away_team'] == fila['away_team']) :
                e['winner'] = fila['winner']
        if "winner" not in e.keys():
            e['winner'] = "Desconocido"

    for e in lt.iterator(respuesta):
        for fila in lt.iterator(control_g):
            if e['date'] == fila['date'] and (e['home_team'] == fila['home_team'] or e['away_team'] == fila['away_team']) :
                e['team'] = fila['team']
                e['scorer'] = fila['scorer']
                e['minute'] = fila['minute']
                e['penalty'] = fila['penalty']
                e['own_goal'] = fila['own_goal']
        if "team" not in e.keys():
            e['team'] = "Desconocido"
        if "scorer" not in e.keys():
            e['scorer'] = "Desconocido"
        if "penalty" not in e.keys():
            e['penalty'] = "Desconocido"
        if "own_goal" not in e.keys():
            e['own_goal'] = "Desconocido"
```

```
    for elemento in lt.iterator(respuesta):
        equipo = elemento['home_team']
        if equipo not in equipos['elements']:
            lt.addLast(equipos,equipo)
        equipo = elemento['away_team']
        if equipo not in equipos['elements']:
            lt.addLast(equipos,equipo)
        pais = elemento['country']
        if pais not in paises['elements']:
            lt.addLast(paises,pais)

        ciudad = elemento['city']
        if ciudad not in ciudades:
            ciudades[ciudad] = 1
        else:
            ciudades[ciudad] += 1

    ocurrencias_maximas = 0

    for ciudad, ocurrencias in ciudades.items():
        if ocurrencias > ocurrencias_maximas:
            ciudad_comun = ciudad
            ocurrencias_maximas = ocurrencias

    for team in lt.iterator(equipos):
        lt.addLast(total_equipos,iberibe_req_6(team, respuesta))



    respuesta_general = {'Total años del historial ': total_anios,
                         'Total torneos en el año seleccionado ':total_torneos,
                         'Total equipos del torneo ': equipos['size'],
                         'Total encuentros disputados ': respuesta['size'],
                         'Total paises involucrados ': paises['size'],
                         'Total ciudades involucradas ': len(ciudades),
                         'Ciudad con mas partidos ': ciudad_comun
                        }

    return respuesta_general, lt.iterator(total_equipos)
```

```
def iberibe_req_6(equipo, lista):

    # Datos en Results discriminados

    goles_hechos = 0
    goles_recibidos = 0
    autogoles_away = 0
    ganado_por_penalty = 0

    partidos_jugados = 0
    partidos_ganados = 0
    partidos_empatados = 0
    partidos_perdidos = 0

    jugadores = lt.newList('ARRAY_LIST')
    # calcula los datos anteriores
    for p in lt.iterator(lista):
        home_n = str(p["home_team"])
        away_n = str(p["away_team"])
        home_p = int(p["home_score"])
        away_p = int(p["away_score"])

        if (home_n == equipo) or (away_n == equipo):
            partidos_jugados += 1

            if (home_n == equipo) and (home_p > away_p):
                    partidos_ganados += 1
                    goles_hechos += home_p
                    goles_recibidos += away_p

            elif (away_n == equipo) and (home_p < away_p):
                    partidos_ganados += 1
                    goles_hechos += away_p
                    goles_recibidos += home_p

            elif home_p == away_p:
                partidos_empatados += 1
                goles_hechos += away_p
                goles_recibidos += home_p

            elif (home_n == equipo) and (home_p < away_p):
                partidos_perdidos += 1
                goles_hechos += home_p
                goles_recibidos += away_p
```

```
        elif (away_n == equipo) and (home_p > away_p):
            partidos_perdidos += 1
            goles_hechos += away_p
            goles_recibidos += home_p

        if p['winner'] == equipo:
            ganado_por_penalty +=1

        if p['team'] == equipo:
            lt.addLast(jugadores, p['scorer'])


    fechas_part_jugador = lt.newList('ARRAY_LIST')
    num_partidos = 0
    suma_min = 0
    goles_totales = 0
    if not lt.isEmpty(jugadores):
        l = jugadores['elements']
        nombre_prominente = max(l, key = (l.count))
        for w in lt.iterator(lista):
            if nombre_prominente == w['scorer']:
                suma_min += float(w['minute'])
                if not lt.isPresent(fechas_part_jugador,w['date']):
                    lt.addLast(fechas_part_jugador,w['date'])
                    num_partidos += 1
                goles_totales += 1
    else:
        nombre_prominente = ""

    Promedio_tiempo = 0
    if goles_totales != 0:
        Promedio_tiempo = suma_min/goles_totales

    respuesta = {'Nombre equipo': equipo,
                 'Puntos totales': 3*partidos_ganados + (partidos_empatados),
                 'Diferencia de goles': goles_hechos - goles_recibidos,
                 'Partidos': partidos_jugados,
                 'Puntos penalties': 3*ganado_por_penalty,
                 'Puntos Autogol': autogoles_away,
                 'Victorias': partidos_ganados,
                 'Empates':partidos_empatados,
                 'Derrotas': partidos_perdidos,
                 'Goles Hechos': goles_hechos,
                 'Goles Recibidos':goles_recibidos,
                 'Mejor Jugador': {
                     'Nombre': nombre_prominente,
                     'Goles Jugador': goles_totales,
                     'Partidos con Gol':num_partidos,
```

El requerimiento 6 busca mediante dos funciones “req_6” y “iberibe_req_6” donde req_6 llama a la otra función para poder generar un diccionario dentro de una lista.

Entrada	Control, nombre_torneo, anio
Salidas	lista
Implementado (Sí/No)	Si, implementado por Andrés Romero

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Acceso a diccionarios	O (1)
Funciones. get	O (1)
Primero bucle for	O (N)
Segundo buble for	O (N^2)
If anidados	O (1)
Llamado a iberibe	O (N)
TOTAL	**O (N^2)**

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 5700U with Radeon Graphics**

Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

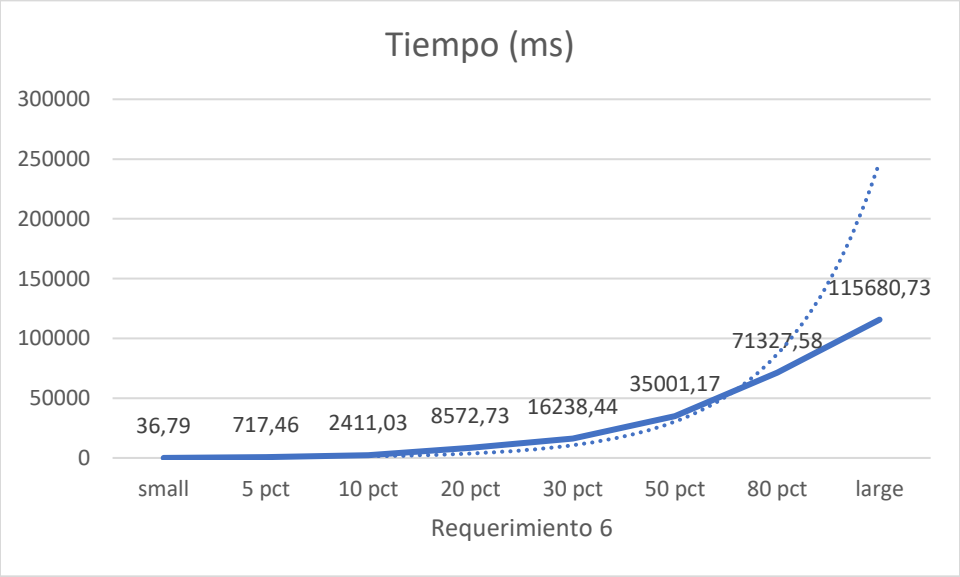
Entrada	**Tiempo (ms)**
small	36.79
5 pct	717.46
10 pct	2411.03
20 pct	8572.73
30 pct	16238.44
50 pct	35001.17
80 pct	71327.58
large	115680.73

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	36.79
5 pct	Dato2	717.46
10 pct	Dato3	2411.03
20 pct	Dato4	8572.73
30 pct	Dato5	16238.44
50 pct	Dato6	35001.17
80 pct	Dato7	71327.58
large	Dato8	115680.73

Graficas



Análisis

La complejidad del requerimiento es exponencial, esto se demuestra con la toma de datos de la gráfica, afortunadamente, la función no toma el peor de los casos puesto que en 80 pct y large no sigue con la curva exponencial que se ve como una línea interlineada, esto demuestra que el requerimiento fue desarrollado de una mejor manera para que no llegase a su peor caso.

Requerimiento <<7>>

Descripción

```python
def req_7(control, torneo, puntos):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    r = []
    lista_g = []
    lista_r = []
    lista_nombres = []

    control_r = control['torneo']
    map = mp.get(control_r, torneo)
    lista = me.getValue(map)


    control_g = control['goleadores']


    for fila in lista['elements']:
        for e in control_g['elements']:
            if fila['date'] == e['date'] and fila['home_team'] == e['home_team'] and fila['away_team'] == e['away_team']:
                lista_g.append(e)
                lista_r.append(fila)
    for i in lista_g:
        lista_nombres, r = iberibe_req_7(lista_nombres, r, i)

    for j in r:
        for g in lista_r:
            j['último_gol']['torneo'] = g['tournament']
            j['último_gol']['goles_local'] = g['home_score']
            j['último_gol']['goles_visitante'] = g['away_score']

            if g['home_team'] == j['team'] and j['fecha'] == g['date']:

                if g['home_score'] > g['away_score']:
                    j['anotaciones_v'] += 1
                elif g['home_score'] < g['away_score']:
                    j['anotaciones_p'] += 1
                elif g['home_score'] == g['away_score']:
                    j['anotaciones_e'] += 1

            if g['away_team'] == j['team'] and j['fecha'] == g['date']:
                if g['home_score'] > g['away_score']:
                    j['anotaciones_v'] += 1
                elif g['home_score'] < g['away_score']:
                    j['anotaciones_p'] += 1
                elif g['home_score'] == g['away_score']:
                    j['anotaciones_e'] += 1

    return r
```

```python
def iberibe_req_7(lista_nombres, lista, i):
    lista_n = lista_nombres
    listo = lista
    if i['scorer'] in lista_n:
        pos = lista_n.index(i['scorer'])
        dic_juagador = lista[pos]
        dic_juagador['puntos'] += 1
        dic_juagador['goles_totales'] += 1
        if i['penalty'] == 'True':
            dic_juagador['goles_penal'] += 1
            dic_juagador['puntos'] += 1
        if i['own_goal'] == 'True':
            dic_juagador['auto_goles'] += 1
            dic_juagador['puntos'] -= 1
        dic_juagador['tiempo_promedio'] = ((dic_juagador['tiempo_promedio']*(dic_juagador['goles_totales']-1)) + float(i['minute']))/dic_juagador['goles_totales']
    else:
        dic = {
            'fecha':i['date'],
            'team':i['team'],
            'nombre':i['scorer'],
            'puntos': 1,
            'goles_totales': 1,
            'goles_penal' : 0,
            'auto_goles' : 0,
            'tiempo_promedio' : float(i['minute']),
            'torneo': 0,
            'anotaciones_v':0,
            'anotaciones_p':0,
            'anotaciones_e':0,
            'último_gol': {
                'fecha' : i['date'],
                'torneo': "",
                'local' : i['home_team'],
                'visitante' : i['away_team'],
                'goles_local': 0,
                'goles_visitante': 0,
                'minuto' : float(i['minute']),
                'penal' : i['penalty'],
                'auto_gol' : i['own_goal']
            }
        }
```

```python
        }
        if i['penalty'] == 'True':
            dic['goles_penal'] += 1
            dic['puntos'] += 1
        if i['own_goal'] == 'True':
            dic['auto_goles'] += 1
            dic['puntos'] -= 1
        listo.append(dic)
        lista_n.append(i['scorer'])
    return lista_n, listo
```

Este es el requerimiento 7 cuyo propósito es brindar la información de los jugadores que hicieron x cantidad de puntos en cierto torneo, para tal fin creamos dos funciones, la original “req_7” y una de ayuda “iberibe_req_7” en la primera función, se realiza todo lo que se ha realizado en requerimientos anteriores, como sacar los valores de los mapas, filtrar y recorrer dichos mapas, ya sea para agregar o eliminar líneas que no sean necesarias para el requerimiento. La segunda función sirve para crear un diccionario nuevo para que este haga parte del tabulate en el segmento de “last_goal” en donde se agrega cierta información que no se encontraba antes.

Entrada	Control, torneo, puntos
Salidas	Lista_n, listo
Implementado (Sí/No)	Si, implementado por Jerónimo Vázquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Acceso a diccionarios	O (1)
Funciones. get	O (1)
Primero bucle for	O (N)
Segundo buble for	O (N^2)
If anidados	O (1)
Llamado a iberibe	O (N)
TOTAL	**O (N^2)**

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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	**Tiempo (ms)**
small	34.75
5 pct	245.0
10 pct	1006.97
20 pct	4694.15
30 pct	12056.14
50 pct	28031.76
80 pct	57348.29
large	76101.55

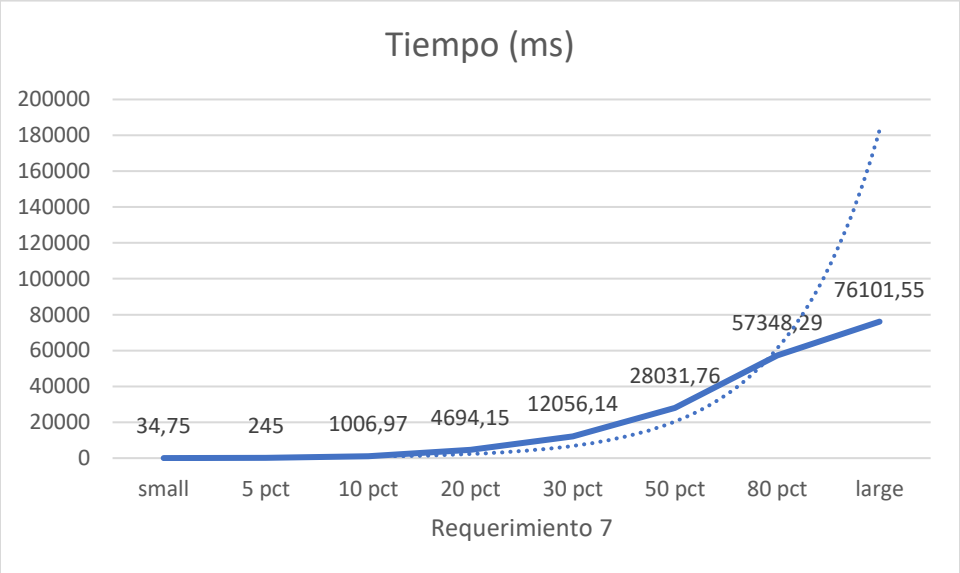
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	34.75
5 pct	Dato2	245.0
10 pct	Dato3	1006.97
20 pct	Dato4	4694.15
30 pct	Dato5	12056.14
50 pct	Dato6	28031.76
80 pct	Dato7	57348.29
large	Dato8	76101.55

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad del requerimiento es constante, donde su complejidad es O (n log n), ya que se está observando el peor caso para el Quick sort en donde el pivote quede completamente desordenado

haciendo que se cree la mayor cantidad posible de pivotes para ordenar el resultado. Este algoritmo de ordenamiento fue necesario para organizar los datos en orden cronológico. Finalmente, podemos saber que la complejidad es constante debido a que las diferentes entradas (desde small hasta large) hicieron que los tiempos de ejecución aumentaran de una manera casi lineal, haciendo que los puntos no sean dispersos con respecto a la línea de tendencia lineal.

Requerimiento <<8>>

Descripción

```python
def req_8(control, pais, anio_i, anio_f):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    r = []
    lista = []
    control_1 = control['hteam']
    control_2 = control['ateam']
    control_g = control['goleadores']

    map1 = mp.get(control_1, pais)
    map2 = mp.get(control_2, pais)

    lista1 = me.getValue(map1)
    lista2 = me.getValue(map2)

    lista = lista1['elements'] + lista2['elements']
    lista_anios = []


    for i in lista:
        if int(i['date'][:4]) >= anio_i and int(i['date'][:4]) <= anio_f:
            lista_anios,r = iberibe_req_8(lista_anios, r,pais, i)

    for e in r:
        lista_goleadores = []
        for fila in control_g['elements']:
            if e['año'] == fila['date'][:4] and (e['local'] == fila['home_team'] and e['visitante'] == fila['away_team']):
                if fila['penalty']:
                    e['goles_penal'] += 1
                if fila['own_goal']:
                    e['auto_goles'] += 1

    return r
```

```python
def iberibe_req_8(lista_anios, lista,nombre, i):
    lista_a = lista_anios
    listo = lista

    if int(i['date'][:4]) in lista_a:
        pos = lista_a.index(int(i['date'][:4]))
        dic_resultados = lista[pos]
        dic_resultados['partidos'] += 1
        if nombre == i['home_team']:
            if i['home_score'] > i['away_score']:
                dic_resultados['victorias'] += 1
                dic_resultados['puntos'] += 3
            elif i['home_score'] < i['away_score']:
                dic_resultados['derrotas'] += 1
            else:
                dic_resultados['empates'] += 1
                dic_resultados['puntos'] += 1
            dic_resultados['goles_favor'] += int(i['home_score'])
            dic_resultados['goles_contra'] += int(i['away_score'])
            dic_resultados['diferencia_goles'] += dic_resultados['goles_favor'] - dic_resultados['goles_contra']
        elif nombre == i['away_team']:
            if i['home_score'] > i['away_score']:
                dic_resultados['derrotas'] += 1
            elif i['home_score'] < i['away_score']:
                dic_resultados['victorias'] += 1
                dic_resultados['puntos'] += 3
            else:
                dic_resultados['empates'] += 1
                dic_resultados['puntos'] += 1
            dic_resultados['goles_favor'] += int(i['away_score'])
            dic_resultados['goles_contra'] += int(i['home_score'])
            dic_resultados['diferencia_goles'] += dic_resultados['goles_favor'] - dic_resultados['goles_contra']
    else:
        dic = {
```

```python
    else:
        dic = {
            'año':i['date'][:4],
            'local':i['home_team'],
            'visitante':i['away_team'],
            'partidos': 1,
            'puntos': 0,
            'diferencia_goles': 0,
            'goles_penal' : 0,
            'auto_goles' : 0,
            'victorias':0,
            'derrotas':0,
            'empates':0,
            'goles_favor':0,
            'goles_contra':0,
            'goleador': {
                'nombre' : '',
                'goles': 0,
                'partidos' :0,
                'minuto' :0
            }
        }
        if nombre == i['home_team']:
            if i['home_score'] > i['away_score']:
                dic['victorias'] += 1
                dic['puntos'] += 3
            elif i['home_score'] < i['away_score']:
                dic['derrotas'] += 1
            else:
                dic['empates'] += 1
                dic['puntos'] += 1
            dic['goles_favor'] += int(i['home_score'])
            dic['goles_contra'] += int(i['away_score'])
            dic['diferencia_goles'] += dic['goles_favor'] - dic['goles_contra']
        elif nombre == i['away_team']:
            if i['home_score'] > i['away_score']:
                dic['derrotas'] += 1
            elif i['home_score'] < i['away_score']:
                dic['victorias'] += 1
                dic['puntos'] += 3
            else:
                dic['empates'] += 1
                dic['puntos'] += 1
            dic['goles_favor'] += int(i['away_score'])
            dic['goles_contra'] += int(i['home_score'])
            dic['diferencia_goles'] += dic['goles_favor'] - dic['goles_contra']
        listo.append(dic)
        lista_a.append(int(i['date'][:4]))
    return lista_a, listo
```

El requerimiento 8 brinda toda la información de un equipo en ciertos periodos de tiempo (no es un rango) en donde se usa una estructura similar al requerimiento 7, puesto que también se usan 2 funciones, para generar un diccionario dentro del tabulate

Entrada	Control, país, año_i, año_f
Salidas	Control, nombre equipo, condición equipo
Implementado (Sí/No)	Si, implementado por Jerónimo Vázquez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Acceso a diccionarios	O (1)
Funciones. get	O (1)
Primero bucle for	O (N)
Segundo buble for	O (N^2)
If anidados	O (1)
Llamado a iberibe	O (N)
TOTAL	**O (N^2)**

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 5700U with Radeon Graphics**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11

Condiciones:

El requerimiento se realizó bajo un linear Probing con 0.5 loadfactor, y con un Shell sort

Entrada	**Tiempo (ms)**
small	2.08
5 pct	26.96
10 pct	92.41
20 pct	278.56
30 pct	503.09
50 pct	793.88
80 pct	1378.4
large	1699.34

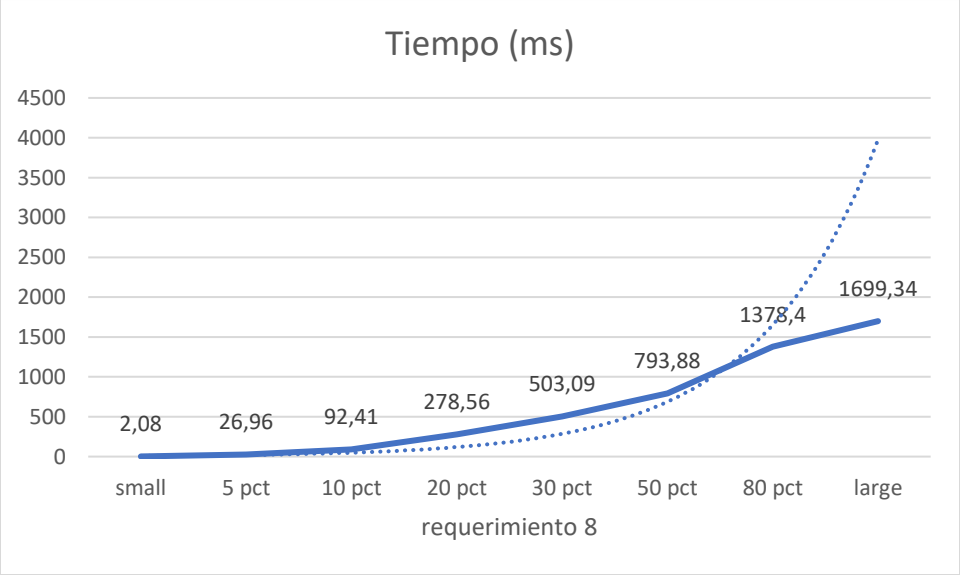
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	2.08
5 pct	Dato2	26.96
10 pct	Dato3	92.41
20 pct	Dato4	278.56
30 pct	Dato5	503.09
50 pct	Dato6	793.88
80 pct	Dato7	1378.4
large	Dato8	1699.34

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al ser una complejidad de O(n^2) se espera que la curva sea como lo indica la línea interlineada, pero gracias a un buen desarrollo del requerimiento, los datos de entrada de 80 pct y large no llegan a tal tiempo de ejecución, cabe recalcar que los datos son bastante altos debido a que se usan dos funciones para realizar este requerimiento, al igual que los requerimientos 6 y 7