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

María Juliana Ballesteros, 202313216, mj.ballesteros@uniandes.edu.co

Santiago Pineda, 202023262, s.pinedaq@unaindes.edu.co

Ian Velandia, 202312488, i.velandiag@uniandes.edu.co

Requerimiento 1

```python
def req_1(data_structs, n_partidos, equipo, condicion):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    results = sort_req1(data_structs)
    size = results["size"]
    i = 1
    lista = []
    while len(lista) <= n_partidos and i < size:
        actual = lt.getElement(results, i)
        if condicion == "Local":
            if actual["home_team"] == equipo:
                lista.append(actual)
        elif condicion == "Visitante":
            if actual["away_team"] == equipo:
                lista.append(actual)
        elif condicion == "Indiferente":
            if actual["home_team"] == equipo or actual["away_team"] == equipo:
                lista.append(actual)
        i += 1
    return lista
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Estructura de datos, número de partidos, equipo, condición
Salidas	Lista de diccionarios que contiene el total de partidos que participó el equipo según su condición.
Implementado (Sí/No)	Si. Implementado por todos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Ordenar la estructura de datos.	O(n Log n)
Inicializar y asignar variables	O(1)
Verificar que la lista creada sea menor o igual a el número de partidos y que sea menor que el tamaño de la estructura de datos.	O(n)
Obtener el elemento (getElement)	O(1)
	O()
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).

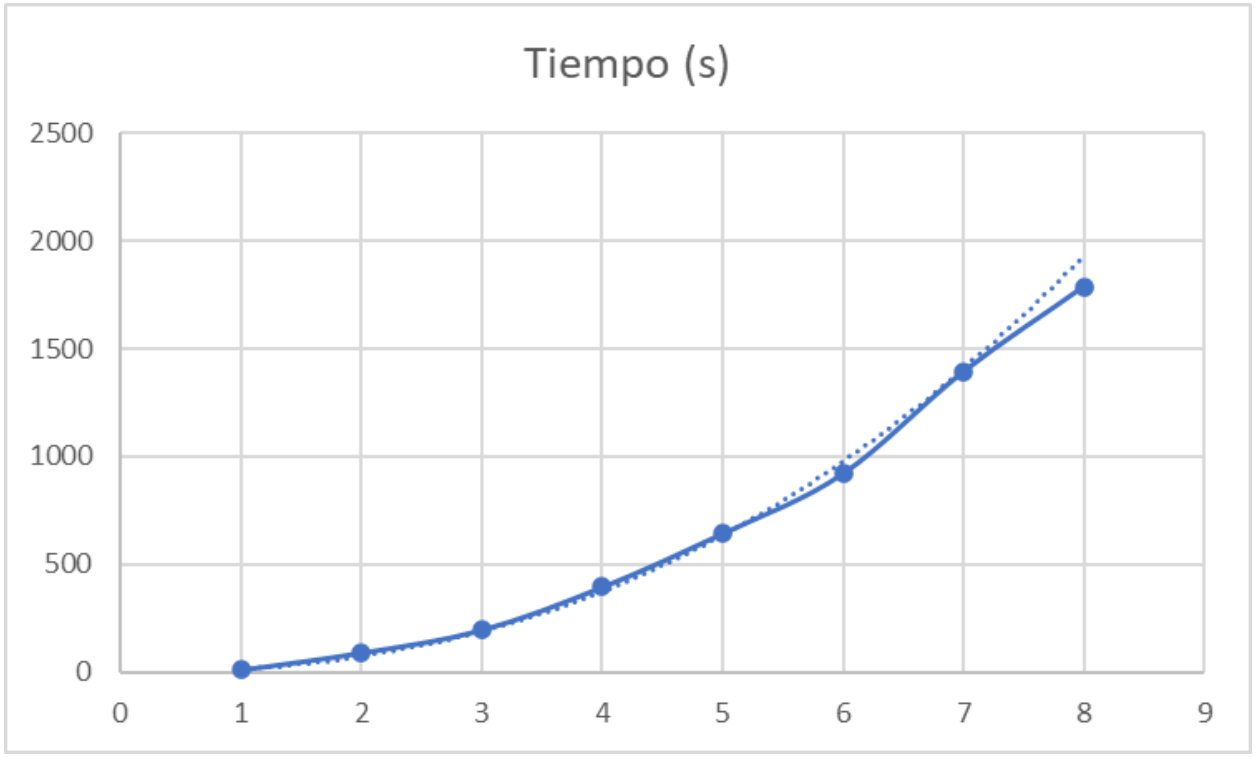
Entrada	Tiempo (ms)
small	11.73
5ptc	91.4
10ptc	196.62
20ptc	395.04
30ptc	643.01
50ptc	920.38
80ptc	1393.37
large	1787.49

Tablas de datos

Entrada	Tiempo (ms)
small	11.73
5ptc	91.4
10ptc	196.62
20ptc	395.04
30ptc	643.01
50ptc	920.38
80ptc	1393.37
large	1787.49

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al implementar el requerimiento 1 la complejidad del algoritmo es O(n Log n) lo que quiere decir que el tiempo de ejecución aumenta logarítmicamente respecto al tamaño de los datos proporcionados, esto se puede ver en la gráfica.

Requerimiento 2

```python

def req_2(data_structs, n_goles, jugador):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    goalscorers = sort_req2(data_structs)
    size = goalscorers["size"]
    i = 1
    lista = []

    while len(lista) <= n_goles and i < size:
        actual = lt.getElement(goalscorers, i)
        if jugador == actual["scorer"]:
            lista.append(actual)
        i += 1
    return lista

```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, numero de goles, jugador
Salidas	Lista de diccionarios que contiene el total de anotaciones que hizo el jugador
Implementado (Sí/No)	Si. implementado por todos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
La primera línea de código llama a la función sort_req2(data_structs).	O(n log n), donde 'n' es el tamaño de data_structs.
variable size	O(1)
Luego, se inicializan las variables i y lista	O(1)

while que tiene dos condiciones: len(lista) <= n_goles y i < size. Al igual que en el caso anterior, la condición len(lista) <= n_goles depende de cuántos elementos se agreguen a lista, mientras que la condición i < size se verifica en cada iteración del bucle. Dentro del bucle, se llama a lt.getElement(goalscorers, i). Supondremos que esto tiene una complejidad de tiempo de O(1) para simplificar. Luego, se verifica si jugador es igual al valor de "scorer" en actual. Esto es una operación O(1). Si la condición es verdadera, se agrega actual a la lista lista, lo cual es una operación O(1). Finalmente, se incrementa i en 1 en cada iteración del bucle, lo cual es una operación O(1)	O(1)
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).

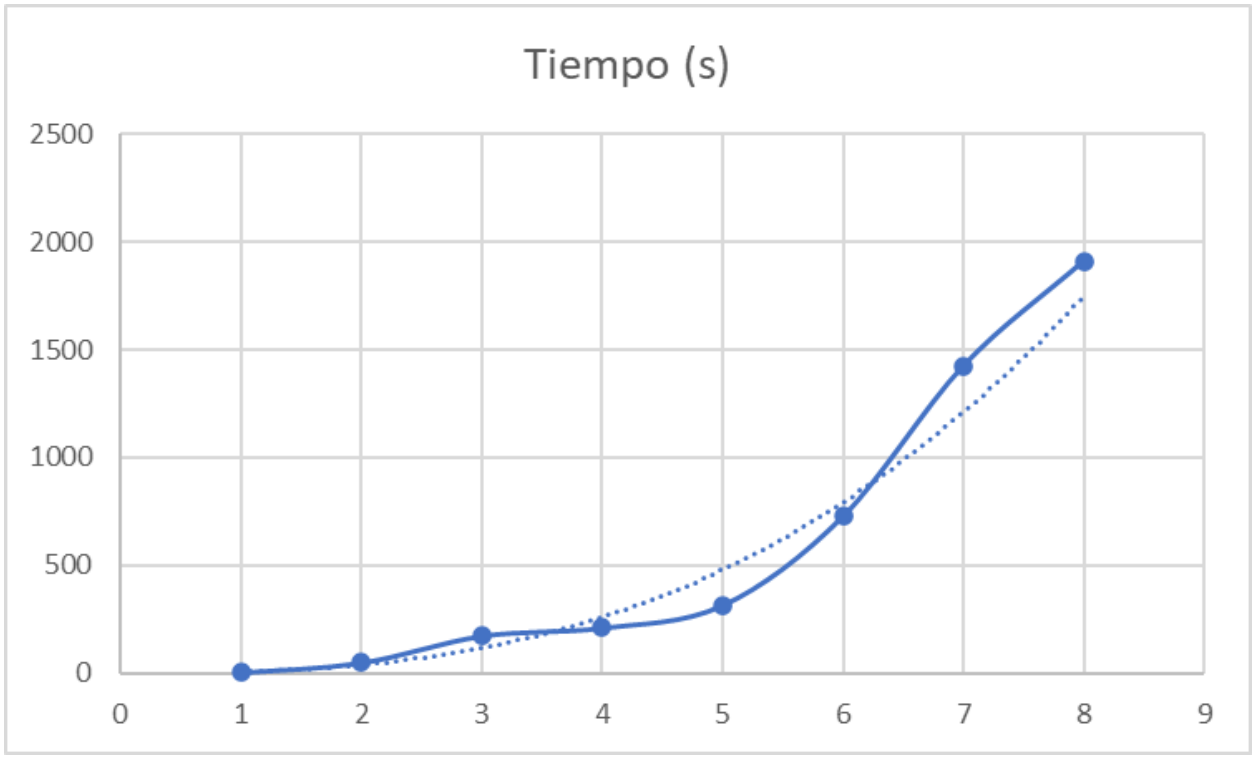
Entrada	Tiempo (ms)
small	4.68
5ptc	50.06
10ptc	174.15
20ptc	211.63
30ptc	315.65
50ptc	727.74
80ptc	1424.83
large	1910.11

Tablas de datos

Entrada	Tiempo (ms)
small	4.68
5ptc	50.06
10ptc	174.15
20ptc	211.63
30ptc	315.65
50ptc	727.74
80ptc	1424.83
large	1910.11

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al implementar el requerimiento 2 da una complejidad de O(n Log n), al igual que en el requerimiento 1, el tiempo de ejecución aumenta respecto al tamaño de los datos como se observa en la gráfica.

Requerimiento 3

```
169  def req_3(data_structs, equipo, fecha_i, fecha_f ):
170      """
171      Función que soluciona el requerimiento 3
172      """
173      # TODO: Realizar el requerimiento 3
174      lista = []
175      results = sort_req3_results(data_structs)
176      i = 1
177      size = results["size"]
178
179      while i < size:
180          actual = lt.getElement(results, i)
181          if actual["date"] >= fecha_i and actual["date"] <= fecha_f:
182              if actual["home_team"] == equipo or actual["away_team"] == equipo:
183                  actual.pop("neutral")
184                  j = 1
185                  goalscorers = sort_req3_goalscorers(data_structs)
186                  size_2 = goalscorers["size"]
187                  encontro = False
188                  while j < size_2:
189                      actual_2 = lt.getElement(goalscorers, j)
190
191                      if actual["date"] == actual_2["date"]:
192
193                          actual["penalty"] = actual_2["penalty"]
194                          actual["own_goal"] = actual_2["own_goal"]
```

```
                    if actual["date"] == actual_2["date"]:

                        actual["penalty"] = actual_2["penalty"]
                        actual["own_goal"] = actual_2["own_goal"]
                        encontro = True
                        lista.append(actual)

                    j += 1

                if encontro == False:

                    actual["penalty"] = "unknown"
                    actual["own_goal"] = True
                    lista.append(actual)


        i += 1
    local = 0
    visitante = 0
    for partido in lista:
        if partido["home_team"] == equipo:
            local += 1
        elif partido["away_team"] == equipo:
            visitante += 1
    return lista, local, visitante
```

Descripción

Breve descripción de cómo abordaron la implementación del requerimiento

Entrada	Estructura de datos, equipo, fecha inicial, fecha final
Salidas	El listado de los partidos disputados ordenados cronológicamente.
Implementado (Sí/No)	Si. Implementado por Juliana Ballesteros

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación variables	O(1)
getElement para obtener primer diccionario	O(1)
Verificar que i sea menor que el tamaño de la primera estructura de datos(results)	O(n)
Verificar que la fecha del diccionario actual sea mayor o igual a la fecha inicial y menor o igual a la fecha final.	O(1)

Verificar que el equipo local o el equipo visitante sean iguales al equipo que entra por parametro.	O(1)
Verificar que j sea menor que el tamaño de la segunda estructura de datos(goalscorers)	O(n^2)
get element para obtener otro diccionario	O(1)
Comparar que la fecha que esta en el primer diccionario y en el segundo sean iguales.	O(1)
Crear 2 llaves y agregarlas al diccionario actual	O(1)
Agregar el diccionario a la lista	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).

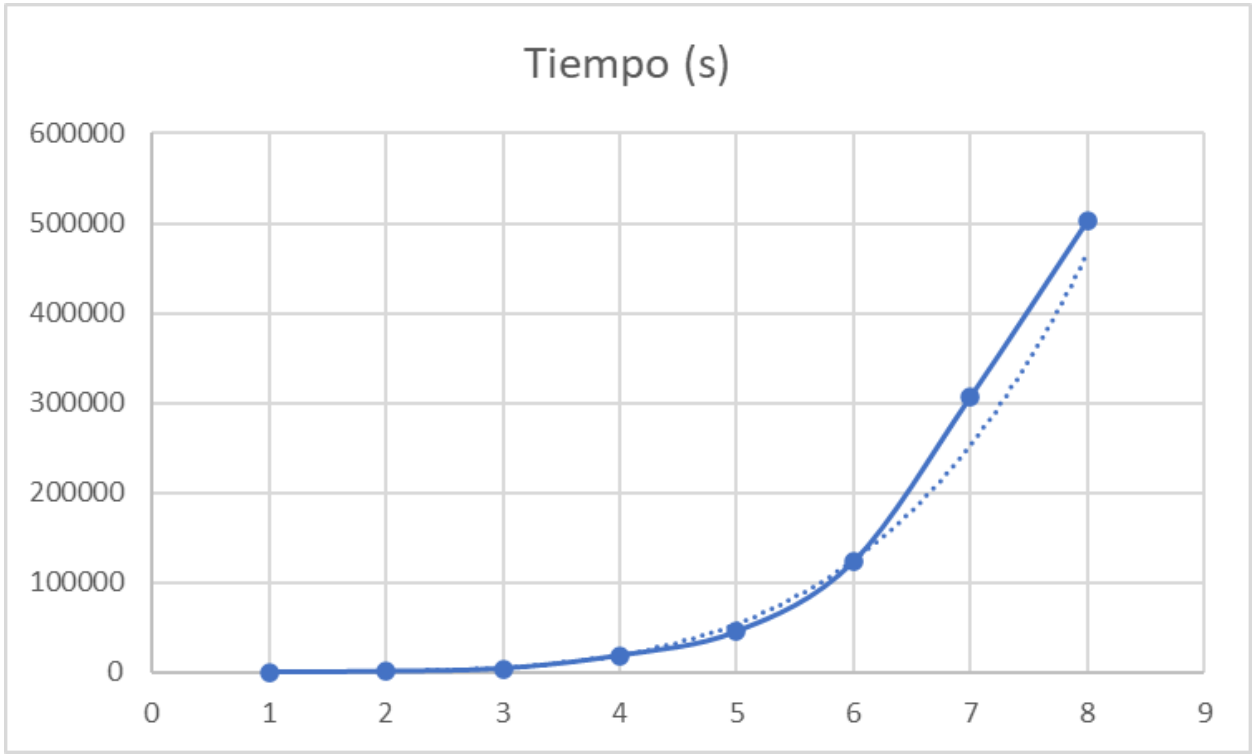
Entrada	Tiempo (ms)
small	34.97
5ptc	988.2
10ptc	4151.41
20ptc	18624.62
30ptc	45266.17
50ptc	122920.25
80ptc	306254.36
large	502193.01

Tablas de datos

Entrada	Tiempo (ms)
small	34.97
5ptc	988.2
10ptc	4151.41
20ptc	18624.62
30ptc	45266.17
50ptc	122920.25
80ptc	306254.36
large	502193.01

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al implementar el requerimiento 3 da una complejidad de n^2, lo que significa que el tiempo de ejecución aumenta cuadráticamente, esto se puede observar en la grafica y en la tabla de datos, ya que mientras hacíamos las pruebas el tiempo aumentaba respecto a los datos.

Requerimiento 4

```python
def req_4(data_structs,torneo,fecha_i,fecha_f):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    results = sort_req1(data_structs)
    size = results["size"]
    lista = []
    i = 1
    centinela = True
    while centinela and i < size:
        actual = lt.getElement(results,i)
        fecha = actual["date"]
        if actual["tournament"] == torneo:
            if fecha_i <= fecha and fecha <= fecha_f:
                if "neutral" in actual:
                    actual.pop("neutral")
                if actual["home_score"] == actual["away_score"]:
                    size_s = lt.size(data_structs["shootouts"])
                    shootouts = lt.subList(data_structs["shootouts"],1,size_s)
                    j = 1
                    bandera = True
                    while j < size_s and bandera:
                        penales = lt.getElement(shootouts,j)
                        if penales["date"] == fecha and actual["home_team"] == penales["home_team"]:
                            actual["winner"] = penales["winner"]
                            bandera = False
                        j += 1
                else:
                    actual["winner"] = "Unknown"
                lista.append(actual)
            elif fecha < fecha_i:
                centinela = False
        i += 1
    paises = []
    ciudades = []
    penales = 0
```

```python
    for partido in lista:
        if partido["home_team"] not in paises:
            paises.append(partido["home_team"])
        if partido["away_team"] not in paises:
            paises.append(partido["away_team"])
        if partido["city"] not in ciudades:
            ciudades.append(partido["city"])
        #if partido["winner"] != "Unknown":
            # penales += 1

    return lista, len(paises), len(ciudades), penales
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, torneo, fecha inicial, fecha final

Salidas	El listado de los partidos disputados ordenados cronológicamente por fecha, nombre del por país y ciudad en que se disputaron los encuentros.
Implementado (Sí/No)	Si. Implementado por Santiago Pineda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Inicialización de variables (results, size, lista, i, centinela, paises, ciudades, penales)	O(1)
Dentro del primer bucle, se realizan varias operaciones condicionales y bucles anidados, incluyendo un segundo bucle while (while j < size_s) que se ejecuta en función de size_s.	O(n^2)
Luego, hay un bucle for que recorre lista. La complejidad de este bucle depende del tamaño de lista.	O(n)
for, donde hay operaciones que involucran la lista paises	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).

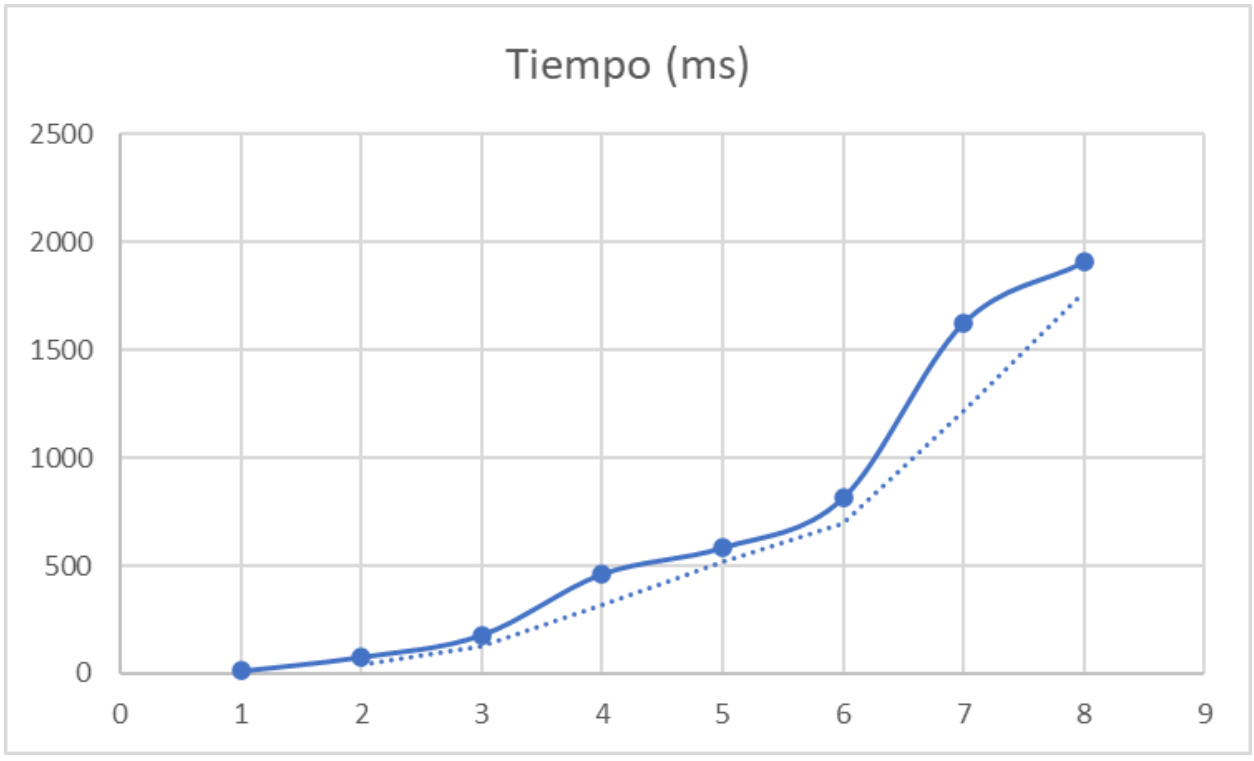
Entrada	**Tiempo (ms)**
small	10.94
5ptc	75.66
10ptc	177.73
20ptc	461.14
30ptc	583.09
50ptc	813.15
80ptc	1624.73
large	1907.65

Tablas de datos

Entrada	Tiempo (ms)
small	10.94
5ptc	75.66
10ptc	177.73
20ptc	461.14
30ptc	583.09
50ptc	813.15
80ptc	1624.73
large	1907.65

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

Requerimiento 5

```python
def req_5(data_structs, nom_player, fecha_i, fecha_f):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    goalscorers = sort_req5(data_structs)
    results = sort_req1(data_structs)
    sizeg = goalscorers["size"]
    sizer = results["size"]
    lista = []
    i = 1
    while i < sizeg:
        actual = lt.getElement(goalscorers, i)
        if actual["date"] >= fecha_i and actual["date"] <= fecha_f:
            if actual["scorer"] == nom_player:
                j = 1
                while j < sizer:
                    data = lt.getElement(results, j)
                    if data["date"] == actual["date"]:
                        if data["home_team"] == actual["home_team"]:
                            actual["tournament"] = data["tournament"]
                            lista.append(actual)
                    j += 1
        i +=1
```

```python
    torneos = []
    penales = 0
    autogoles = 0
    for goles in lista:
        if goles["tournament"] not in torneos:
            torneos.append(goles["tournament"])
        if goles["penalty"] != "False":
            penales += 1
        elif goles["own_goal"] != "False":
            autogoles += 1

    return lista, len(torneos), penales, autogoles

```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, jugador, fecha inicial, fecha final.
Salidas	El listado de las anotaciones del jugador ordenadas cronológicamente por fecha y minuto en que se marcó el gol en el encuentro.
Implementado (Sí/No)	Si. implementado por Ian Velandia .

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad

Inicialización de variables: goalscorers, results, sizeg, sizer, lista, i, torneos, penales, autogoles	O(1)
Primer ciclo while (while i < sizeg): El ciclo se ejecuta hasta que i sea igual a sizeg, y i se incrementa en cada iteración. En el peor caso, el bucle se ejecuta sizeg veces.	O(N)
Dentro del primer ciclo, se realizan varias operaciones condicionales y ciclos anidados, incluyendo un segundo ciclo while (while j < sizer) que se ejecuta en función de sizer. La complejidad de esta parte del código depende de cuántas veces se ejecuta este segundo ciclo.	O(N^2)
Luego, hay un ciclo for que recorre la lista. La complejidad de este bucle depende del tamaño de lista	O(N)
Dentro del bucle for, hay operaciones que involucran las listas torneos, penales, autogoles, y variables de acceso a los elementos de lista. La complejidad de estas operaciones depende del tamaño de las listas y de cuántos elementos únicos haya en ellas.	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).

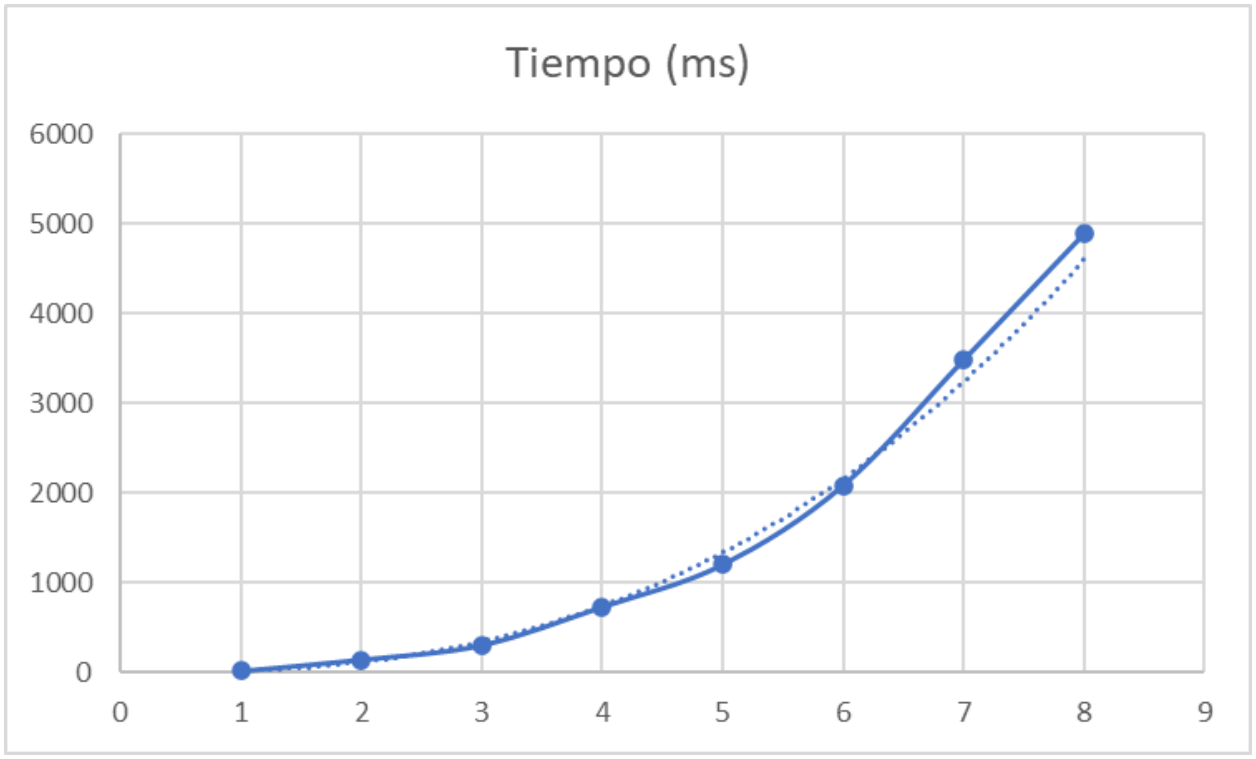
Entrada	Tiempo (ms)
small	18.8
5ptc	145.97
10ptc	307.46
20ptc	733.78
30ptc	1204.72
50ptc	2085.44
80ptc	3482.31
large	4888.29

Tablas de datos

Entrada	Tiempo (ms)
small	18.8
5ptc	145.97
10ptc	307.46
20ptc	733.78
30ptc	1204.72
50ptc	2085.44
80ptc	3482.31
large	4888.29

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

Requerimiento 6

```python
def req_6(data_structs,n_equipos,torneo,fecha_i,fecha_f):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    mejores = lt.newList('ARRAY_LIST')
    paises = {}
    results = sort_req1(data_structs)
    goalscorers = sort_req2(data_structs)
    size = results["size"]
    size_goals = goalscorers["size"]
    jugadores = {}
    i = 1
    poderoso = True
    partidos = 0
    ciudades = {}
    while i < size and poderoso:
        data = lt.getElement(results,i)
        fecha = data["date"]
        pais_1 = data["home_team"]
        pais_2 = data["away_team"]
        if torneo == data["tournament"]:
            if fecha >= fecha_i and fecha <= fecha_f:
                partidos += 1
                if data["city"] not in ciudades:
                    ciudades[data["city"]] = 1
                else:
                    ciudades[data["city"]] += 1
                if pais_1 not in paises:
                    paises[pais_1] = {"team":None,
                    "points": 0,
                    "dif_goles": 0,
                    "Matches_played": 0,
                    "Goles x penal": 0,
                    "Autogoles": 0,
                    "Victorias": 0,
                    "Empates":0,
```

```python
                    "Derrotas":0,
                    "Goles":0,
                    "Goles Recibidos":0,
                    "Top Scorer": {"Nombre":None,
                                   "Goles":0,
                                   "Scored Matches": 0,
                                   "Promedio":0}}
                    paises[pais_1]["team"] = pais_1
                if pais_2 not in paises:
                    paises[pais_2] = {"team":None,
                    "points": 0,
                    "dif_goles": 0,
                    "Matches_played": 0,
                    "Goles x penal": 0,
                    "Autogoles": 0,
                    "Victorias": 0,
                    "Empates":0,
                    "Derrotas":0,
                    "Goles":0,
                    "Goles Recibidos":0,
                    "Top Scorer": {"Nombre":None,
                                   "Goles":0,
                                   "Scored Matches": 0,
                                   "Promedio":0}}
                    paises[pais_2]["team"] = pais_2
                if float(data["home_score"]) > float(data["away_score"]):
                    paises[pais_1]["points"] += 3
                    paises[pais_1]["Victorias"] += 1
                    paises[pais_1]["Matches_played"] += 1
                    paises[pais_1]["Goles"] += float(data["home_score"])
                    paises[pais_1]["Goles Recibidos"] += float(data["away_score"])

                    paises[pais_2]["Derrotas"] += 1
                    paises[pais_2]["Matches_played"] += 1
                    paises[pais_2]["Goles"] += float(data["away_score"])
                    paises[pais_2]["Goles Recibidos"] += int(data["home_score"])
                    j = 1
                    centinela = True
```

```
                    bandera = True
                    while centinela and j < size_goals:
                        actual = lt.getElement(goalscorers, j)
                        if fecha == actual["date"]:
                            if pais_1 == actual["home_team"]:
                                if actual["own_goal"]:
                                    paises[actual["team"]]["Autogoles"] += 1
                                if actual["penalty"]:
                                    paises[actual["team"]]["Goles x penal"] += 1
                                if actual["scorer"] not in jugadores:
                                    jugadores[actual["scorer"]] = {"pais":actual["team"],
                                                                   "goles":0,
                                                                   "Scored Matches":0,
                                                                   "promedio":0}
                                jugadores[actual["scorer"]]["goles"] += 1
                                jugadores[actual["scorer"]]["promedio"] += float(actual["minute"])
                                if bandera:
                                    jugadores[actual["scorer"]]["Scored Matches"] += 1
                                    bandera = False
                        elif fecha < actual["date"]:
                            centinela = False
                        j += 1

                elif int(data["home_score"]) < int(data["away_score"]):
                    paises[pais_2]["points"] += 3
                    paises[pais_2]["Victorias"] += 1
                    paises[pais_2]["Matches_played"] += 1
                    paises[pais_2]["Goles"] += int(data["away_score"])
                    paises[pais_2]["Goles Recibidos"] += int(data["home_score"])

                    paises[pais_1]["Derrotas"] += 1
                    paises[pais_1]["Matches_played"] += 1
                    paises[pais_1]["Goles"] += int(data["home_score"])
                    paises[pais_1]["Goles Recibidos"] += int(data["away_score"])
                    j = 1
                    centinela = True
                    bandera = True
                    while centinela and j < size_goals:
```

```
                        actual = lt.getElement(goalscorers, j)
                        if fecha == actual["date"]:
                            if pais_1 == actual["home_team"]:
                                if actual["own_goal"]:
                                    paises[actual["team"]]["Autogoles"] += 1
                                if actual["penalty"]:
                                    paises[actual["team"]]["Goles x penal"] += 1
                                if actual["scorer"] not in jugadores:
                                    jugadores[actual["scorer"]] = {"pais":actual["team"],
                                                                   "goles":0,
                                                                   "Scored Matches":0,
                                                                   "promedio":0}
                                jugadores[actual["scorer"]]["goles"] += 1
                                jugadores[actual["scorer"]]["promedio"] += float(actual["minute"])
                                if bandera:
                                    jugadores[actual["scorer"]]["Scored Matches"] += 1
                                    bandera = False
                        elif fecha < actual["date"]:
                            centinela = False
                        j += 1
                else:
                    paises[pais_2]["points"] += 1
                    paises[pais_2]["Empates"] += 1
                    paises[pais_2]["Matches_played"] += 1
                    paises[pais_2]["Goles"] += int(data["away_score"])
                    paises[pais_2]["Goles Recibidos"] += int(data["home_score"])

                    paises[pais_1]["points"] += 1
                    paises[pais_1]["Empates"] += 1
                    paises[pais_1]["Matches_played"] += 1
                    paises[pais_1]["Goles"] += int(data["home_score"])
                    paises[pais_1]["Goles Recibidos"] += int(data["away_score"])
                    j = 1
                    centinela = True
                    bandera = True
                    while centinela and j < size_goals:
                        actual = lt.getElement(goalscorers, j)
                        if fecha == actual["date"]:
```

```
                        if pais_1 == actual["home_team"]:
                            if actual["own_goal"]:
                                paises[actual["team"]]["Autogoles"] += 1
                            if actual["penalty"]:
                                paises[actual["team"]]["Goles x penal"] += 1
                            if actual["scorer"] not in jugadores:
                                jugadores[actual["scorer"]] = {"pais":actual["team"],
                                                               "goles":0,
                                                               "Scored Matches":0,
                                                               "promedio":0
                                                               }
                            jugadores[actual["scorer"]]["goles"] += 1
                            jugadores[actual["scorer"]]["promedio"] += float(actual["minute"])
                            if bandera:
                                jugadores[actual["scorer"]]["Scored Matches"] += 1
                                bandera = False
                    elif fecha < actual["date"]:
                        centinela = False
                    j += 1
            elif fecha < fecha_i:
                poderoso = False
            i += 1

        for jugador in jugadores:
            actual = jugadores[jugador]
            promedio = actual["promedio"]/actual["goles"]
            actual["promedio"] = promedio
        for pais in paises:
            actual = paises[pais]
            actual["dif_goles"] = actual["Goles"] - actual["Goles Recibidos"]
            mejor = {"pais":None,
                     "goles":0,
                    "Scored Matches":0,
                    "promedio":0
                              }
            max_g = 0
            nombre = None
            for jugador in jugadores:
                jug_act = jugadores[jugador]
                if jug_act["pais"] == pais:
                    if max_g < jug_act["goles"]:
                        max_g = jug_act["goles"]
                        mejor = jug_act
                        nombre = jugador
            paises[pais]["Top Scorer"] = {"Nombre":nombre,
                                          "Goles":mejor["goles"],
                                          "Scored Matches": mejor["Scored Matches"],
                                          "Promedio":mejor["promedio"]}
            lt.addLast(mejores,actual)
        mejores = sort_req_6(mejores)
        mejores = lt.subList(mejores,1,n_equipos)
        max_c = 0
        mejor = None
        for ciudad in ciudades:
            actual = ciudades[ciudad]
            if actual > max_c:
                max_c = actual
                mejor = ciudad
        return mejores, partidos, len(ciudades), mejor
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, número de equipos, torneo, fecha inicial, fecha final
Salidas	El listado de los equipos que conforman el torneo debe estar ordenado por el criterio compuesto de sus estadísticas
Implementado (Sí/No)	Si se implementó, grupal.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Inicialización de variables (mejores, paises, results, goalscorers, size, size_goals, jugadores, i, poderoso, partidos, ciudades)	O(1)
Primer bucle while (while i < size and poderoso): El bucle se ejecuta hasta que i sea igual a size o poderoso sea False, y i se incrementa en cada iteración. O(n)	O(n)
Dentro del primer bucle, se realizan varias operaciones condicionales y bucles anidados, incluyendo un segundo bucle while (while j < size_goals) O(n)	O(n)
Luego, hay bucles for que recorren jugadores y paises. La complejidad de estos bucles depende del tamaño de jugadores y paises. O(n)	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

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

Entrada	Tiempo (ms)
small	16.97
5ptc	209.64
10ptc	774.66
20ptc	2880.32
30ptc	5970.69
50ptc	3486.21
80ptc	26132.5
large	35950.18

Tablas de datos

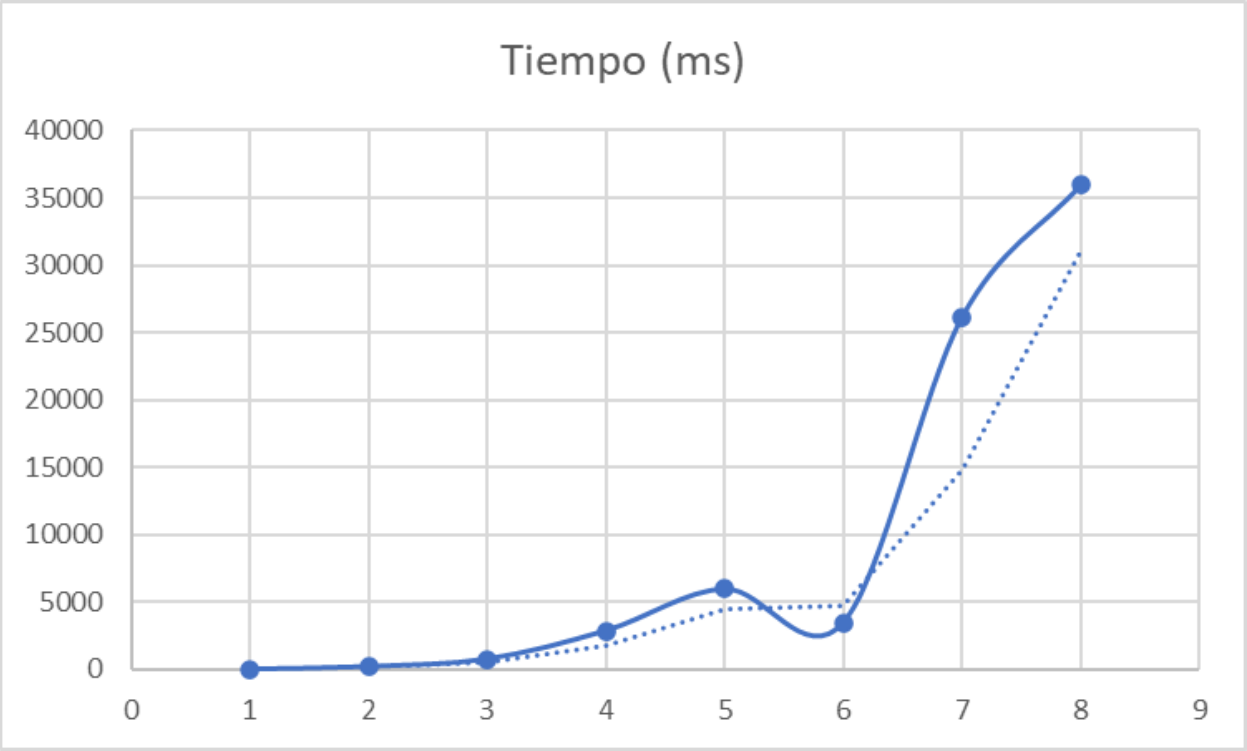
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)

small	16.97
5ptc	209.64
10ptc	774.66
20ptc	2880.32
30ptc	5970.69
50ptc	3486.21
80ptc	26132.5
large	35950.18

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

Requerimiento 7

```python
def req_7(data_structs,n_jugadores,fecha_i,fecha_f):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    mejores = lt.newList('ARRAY_LIST')
    results = sort_req1(data_structs)
    goalscorers = sort_req2(data_structs)
    size = results["size"]
    size_goals = goalscorers["size"]
    jugadores = {}
    i = 1
    poderoso = True
    partidos = 0
    while i < size and poderoso:
        data = lt.getElement(results,i)
        fecha = data["date"]
        pais_1 = data["home_team"]
        if fecha >= fecha_i and fecha <= fecha_f:
            if data["tournament"] != "Friendly":
                partidos += 1
                j = 1
                centinela = True
                bandera = True
                while centinela and j < size_goals:
                    actual = lt.getElement(goalscorers, j)
                    jugador = actual["scorer"]
```

```python
                    jugador = actual["scorer"]
                    if fecha == actual["date"]:
                        if pais_1 == actual["home_team"]:
                            if jugador not in jugadores:
                                jugadores[jugador] = {"nombre":jugador,
                                                      "total_points":0,
                                                      "total_goals":0,
                                                      "penalty_goals":0,
                                                      "own_goals":0,
                                                      "avg_time(min)":0,
                                                      "total_tournaments":[],
                                                      "scored_in_wins":0,
                                                      "scored_in_losses":0,
                                                      "scored_in_draws":0,
                                                      "last_goal":{"date":None,
                                                                   "tournament":None,
                                                                   "home_team":None,
                                                                   "away_team":None,
                                                                   "home_score":0,
                                                                   "away_score":0,
                                                                   "minute":0,
                                                                   "penalty":"Unknown",
                                                                   "own_goal":"Unknown"}}
                            if actual["penalty"]:
                                jugadores[jugador]["total_points"] += 2
                                jugadores[jugador]["penalty_goals"] += 1
                            else:
                                jugadores[jugador]["total_points"] += 1
                            jugadores[jugador]["total_goals"] += 1
```

```
                    jugadores[jugador]["total_goals"] += 1
                    if actual["own_goal"]:
                        jugadores[jugador]["total_points"] -= 1
                        jugadores[jugador]["own_goals"] += 1
                    jugadores[jugador]["avg_time(min)"] += float(actual["minute"])
                    if data["tournament"] not in jugadores[jugador]["total_tournaments"]:
                        jugadores[jugador]["total_tournaments"].append(data["tournament"])
                    if actual["team"] == data["home_team"]:
                        if int(data["home_score"]) > int(data["away_score"]):
                            jugadores[jugador]["scored_in_wins"] += 1
                        elif int(data["home_score"]) < int(data["away_score"]):
                            jugadores[jugador]["scored_in_losses"] += 1

                    if actual["team"] == data["away_team"]:
                        if int(data["away_score"]) > int(data["home_score"]):
                            jugadores[jugador]["scored_in_wins"] += 1
                        elif int(data["home_score"]) > int(data["away_score"]):
                            jugadores[jugador]["scored_in_losses"] += 1
                    else:
                        jugadores[jugador]["scored_in_draws"] += 1
                    if bandera:
                        last = jugadores[jugador]["last_goal"]
                        last["date"] = fecha
                        last["tournament"] = data["tournament"]
                        last["home_team"] = pais_1
                        last["away_team"] = actual["away_team"]
                        last["home_score"] = data["home_score"]
                        last["away_score"] = data["away_score"]
```

```
                            last["away_score"] = data["away_score"]
                            last["minute"] = actual["minute"]
                            last["penalty"] = actual["penalty"]
                            last["own_goal"] = actual["own_goal"]
                        lt.addLast(mejores,jugadores[jugador])
                    j += 1
            i += 1
        mejores = sort_req_7(mejores)
        total_players = lt.size(mejores)
        goles = 0
        penaltis = 0
        autogoles = 0
        for jugador in lt.iterator(mejores):
            goles += jugador["total_goals"]
            penaltis += jugador["penalty_goals"]
            autogoles += jugador["own_goals"]
        mejores = lt.subList(mejores,1,n_jugadores)
        return mejores, total_players, partidos, goles, penaltis, autogoles
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos, jugador, fecha inicial, fecha final.
Salidas	El listado de anotadores debe estar ordenado por el criterio compuesto de sus estadísticas
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad

Asignación de variables (mejores, results, goalscorers, size, size_goals, jugadores, i, poderoso, partidos) O(1).	O(1)
Primer bucle while (while i < size and poderoso): El bucle se ejecuta hasta que i sea igual a size O(n)	O(n)
Dentro del primer bucle, se realizan varias operaciones condicionales y bucles anidados, incluyendo un segundo bucle while (while j < size_goals) que se ejecuta O(n) veces	O(n)
Luego, hay un for que recorre mejores.	O(n)
Al final, hay un bucle que ordena mejores y otro que crea una sublista mejores	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

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

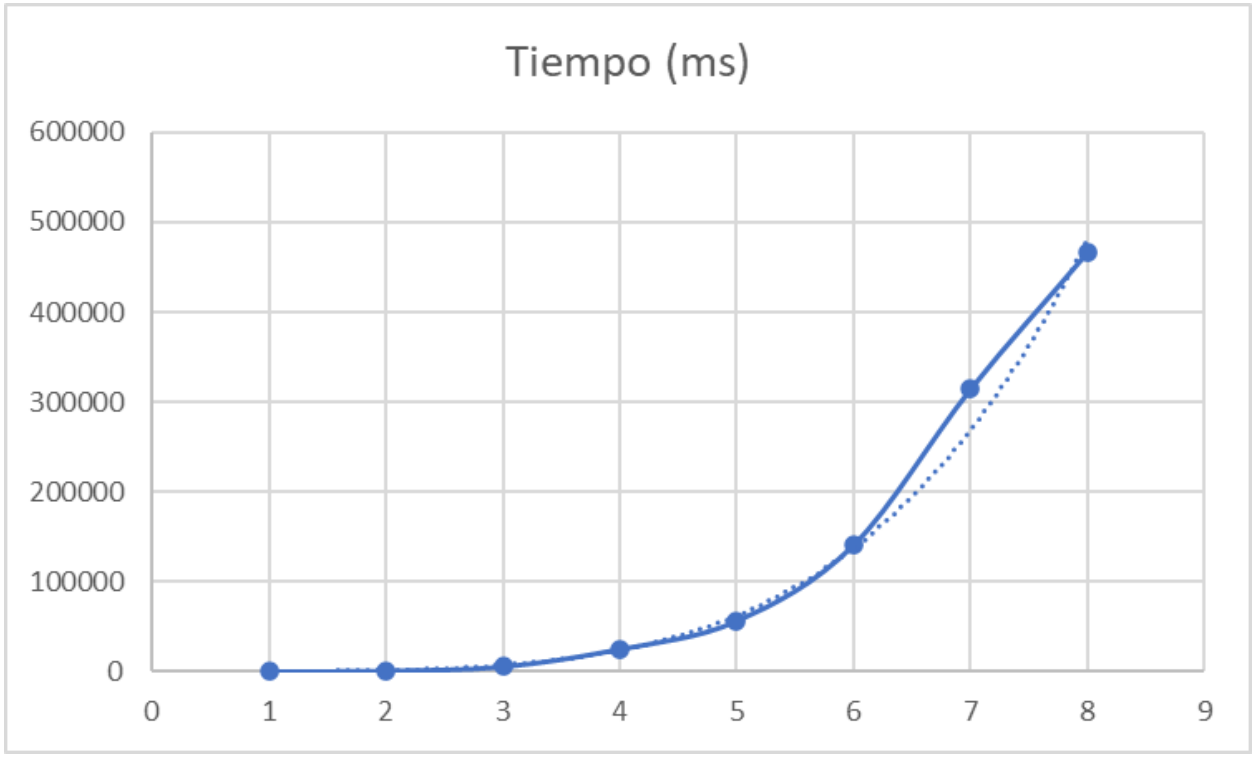
Entrada	Tiempo (ms)
small	61.65
5ptc	1120.95
10ptc	5651.81
20ptc	25186.25
30ptc	56458.48
50ptc	140955.14
80ptc	314263.51
large	466481.03

Tablas de datos

Entrada	Tiempo (ms)
small	61.65
5ptc	1120.95
10ptc	5651.81
20ptc	25186.25
30ptc	56458.48
50ptc	140955.14
80ptc	314263.51
large	466481.03

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

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

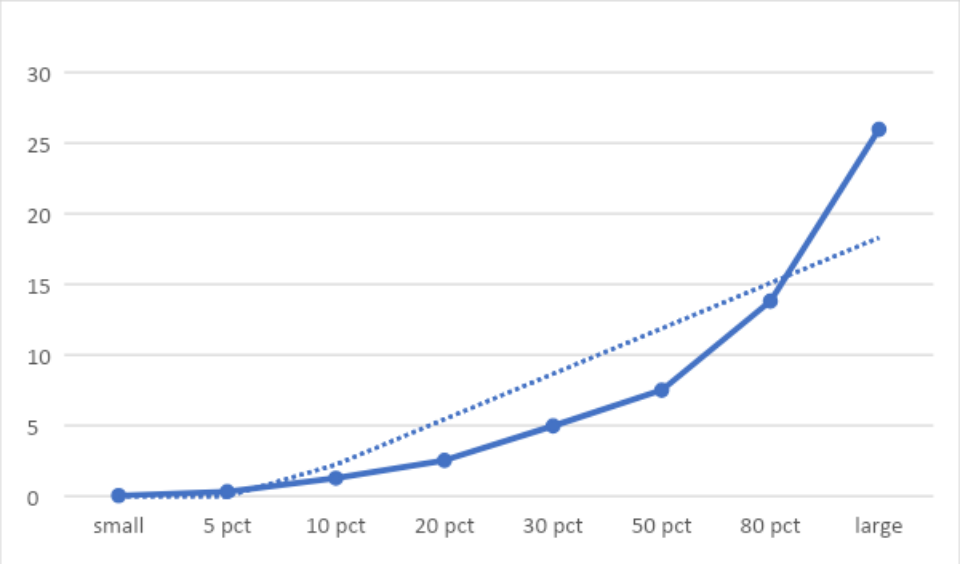
Las tablas con la recopilación de datos de las pruebas.

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.