ANÁLISIS DEL RETO

Ximena Lopez, 202312848, ax.lopez@uniandes.edu.co

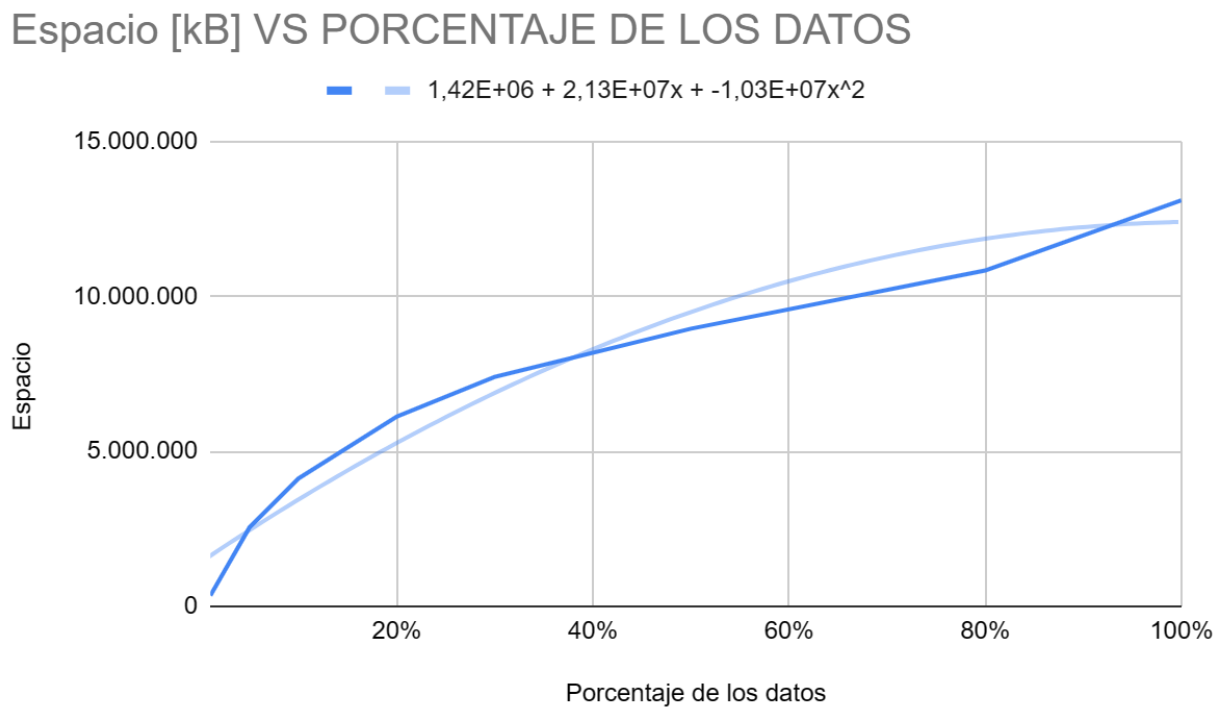
Juan David Torres, 202317608, jd.torresa1@uniandes.edu.co

Sofia Losada M, 202221008, s.losadam@uniandes.edu.co

Carga de datos

Tabla de memoria

PORCENTAJE DATOS	PROMEDIO[kB]
SMALL	352 157
5%	2 549 352
10%	4 125 833
20%	6 123 112
30%	7 405 657
50%	8 956 843
80%	10 832 940
LARGE	13 102 004



Requerimiento <<1>>

```
def req_1(data_structs, team, condition):
"""
Función que soluciona el requerimiento 1
"""
# TODO: Realizar el requerimiento 1
if condition!="MatchResults":
resultado=
me.getValue(mp.get(me.getValue(mp.get(data_structs["model"]["teams"],team))["matchResultsByCondition"],condition))
else:
resultado= me.getValue(mp.get(data_structs["model"]["teams"],team))["MatchResults"]["list"]
if resultado:
total_teams= mp.size(data_structs["model"]["teams"])
total_partidos= lt.size(me.getValue(mp.get(data_structs["model"]["teams"],team))["MatchResults"]["list"])
return merg.sort(resultado,results_sort_criteria),total_teams,total_partidos
else:
```

```
return "El equipo no tiene partidos en esa condicion",0,0
```

Descripción

En este requerimiento se desean conocer los últimos partidos jugados más recientemente por un equipo según su condición en el encuentro (local, visitante, o indiferente).

Entrada	• El número (N) de partidos de consulta • Nombre del equipo (país de la selección nacional en inglés). • Condición del equipo en los partidos consultados (Local, Visitante, o Indiferente).
Salidas	• El total de equipos con información disponible. • El total de partidos en que participó el equipo (sin importar su condición). • El subtotal de los partidos en que participó el equipo según su condición (local, visitante, indiferente). • Para cada uno de los partidos en la consulta se debe desplegar la siguiente información: ○ Fecha del partido. o Equipo local. o Equipo visitante. o País del encuentro. ○ Ciudad donde se disputa el encuentro. ○ Marcador del equipo local (goles del local). ○ Marcador del equipo visitante (goles del visitante).
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Primer condicional que adquiere los valores del mapa ”teams”	O(1)
Paso 2 Segundo condicional que de acuerdo al valor de resultado resultado calcula el total de equipos y partidos.	O(1)
TOTAL	***O(1)***

Pruebas Realizadas

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

Procesadores	AMD Ryzen 7 4800HS with Radeon Graphics
Memoria RAM	8 GB
Sistema Operativo	IOS 11.7.10

Comparación de tiempo con Reto #1

TAMAÑO ENTRADA	RETO 1 O(n)	RETO2 O(1)
small	1.625	0,5014333333
5pct	3.480	1,599
10pct	9.195	1,776666667
20pct	11.508	2,58
30pct	15.942	5,766666667
50pct	17.940	10,05666667
80pt	47.085	5,73
large	38.026	15,53333333

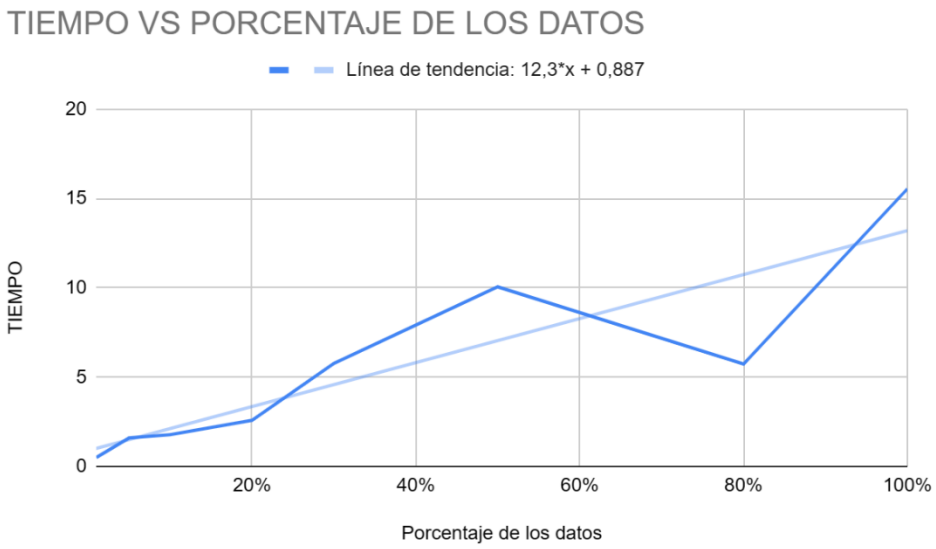
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (s)
small	0,5014333333
5pct	1,599
10pct	1,776666667
20pct	2,58
30pct	5,766666667
50pct	10,05666667
80pt	5,73
large	15,53333333

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.

El requerimiento implementado con mapas baja mucho la complejidad porque al momento de realizar la carga de datos se va haciendo lo necesario y solo queda una complejidad de o(1) al consultar. Es mucho más eficiente con mapas que con listas. Un mapa generalmente tiene una complejidad temporal menor que una lista debido a su capacidad de acceso rápido por clave en lugar de índice. La complejidad promedio de acceso en un mapa es O(1), mientras que en una lista es O(n), lo que significa que el tiempo de acceso en una lista aumenta linealmente con el tamaño. Los mapas son más flexibles en términos de claves y son eficientes para inserciones y eliminaciones, lo que los hace una elección acertada teniendo en cuenta los requisitos del requerimiento.

Requerimiento <<2>>

```
def req_2(data_structs, nombre, cant_goles):
"""
Función que soluciona el requerimiento 2
"""
# TODO: Realizar el requerimiento 2
if mp.contains(data_structs["jugador_goles"], nombre):
goles_entry = mp.get(data_structs["jugador_goles"],nombre)
goles = me.getValue(goles_entry)
```

```
merg.sort(goles,cmp_crit_goal_req_2)
if lt.size(goles) >= cant_goles:
return lt.subList(goles, 1, cant_goles)
else:
return goles
else:
return "El jugador no existe"
```

Descripción

En este requerimiento se desean conocer los partidos jugados por un jugador en específico en la historia y además conocer una cantidad específica de goles marcados

Entrada	• El número (N) de goles de consulta • Nombre completo del jugador.
Salidas	• El total de jugadores con anotaciones registradas. • El total de anotaciones obtenidas por el jugador. • El subtotal de anotaciones desde el punto penal • Para cada una de las anotaciones en la consulta se debe desplegar la siguiente información: ○ Fecha del partido. ○ Equipo local o Equipo visitante. ○ Equipo del jugador. ○ Minuto en el que se marcó el gol. ○ Tipo de anotación, si fue por falta desde el penal. ○ Tipo de anotación, si fue autogol.
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Primer condicional donde se pregunta si el mapa creado específicamente para el requerimiento contiene el nombre del jugador. Si es verdadero obtiene las llaves y valores del mapa correspondiente a la información concreta del jugador, se guarda en una lista/arreglo y se organiza mediante un merge	O(1)

Paso 2 Si es verdadero obtiene las llaves y valores del mapa correspondiente a la información concreta del jugador, se guarda en una lista/arreglo y se organiza mediante un merge	O(nlogn)
Paso 3 Condicional anidado que pregunsta si el tamaño de la lista dada es mayor a la de la cantidad de goles que se dan por parámetro. Si esverdadero retrona una sublista	O(1)
TOTAL	***O(nlogn)***

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 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	IOS 11.7.10

Comparación de tiempo con Reto #1

TAMAÑO ENTRADA	**RETO 1 O(nlogn)**	**RETO2 O(nlogn)**
small	12.497	0,5566666667
5pct	76.249	1,423333333
10pct	145.699	1,563333333
20pct	393.988	2,766666667
30pct	604.956	4,206666667
50pct	951.204	10,28
80pt	3052.140	12,10333333
large	6441.864	15,5

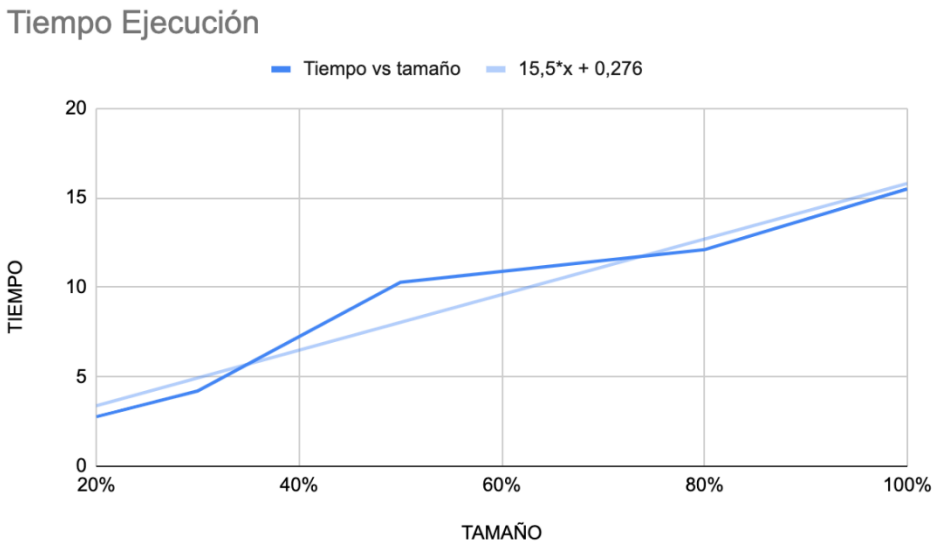
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	0,5566666667
5pct	1,423333333
10pct	1,563333333

20pct	2,766666667
30pct	4,206666667
50pct	10,28
80pt	12,10333333
large	15,5

Graficas



Análisis

En la gráfica se demuestra una tendencia lineal en el comportamiento de los datos, por lo que responde al análisis de complejidad realizado. De esta manera, el algoritmo responde de manera linearítmica al crecimiento del volumen de datos. Este comportamiento linerarítimico se debe, principalmente, al comportamiento que experimenta merge sort en cuanto al volumen de datos, es decir, nlogn.

Requerimiento <<3>>

```
def req_3(data_structs, date_in, date_f,team):
"""
Función que soluciona el requerimiento 3
"""
away_team= 0
home_team=0
```

```
lista=lt.newList("ARRAY_LIST")
todos=lt.newList("ARRAY_LIST")
date_inicial=date.fromisoformat(date_in).year
date_final=date.fromisoformat(date_f).year
date_inicial_fecha=date.fromisoformat(date_in)
date_final_fecha=date.fromisoformat(date_f)
mapa= data_structs["model"]["results"]
resultado= mp.keySet(mapa)
for key in lt.iterator(resultado):


if date_final>=key and key>=date_inicial:
valor= me.getValue(mp.get(mapa,key))
for cada in lt.iterator(valor):
fecha= date.fromisoformat(cada["date"])
if date_final_fecha>=fecha and fecha>=date_inicial_fecha:


#para todos
away=cada["away_team"]
todos = its_present_partidos(todos,away)
home= cada["home_team"]
todos = its_present_partidos(todos,home)
if away==team:
away_team+=1
auto,penal=its_present_in_goal(data_structs,fecha,away,"away_team")
cada["penalty"]=penal
cada["own_goal"]=auto
lt.addLast(lista,cada)
elif home==team:
home_team+=1
```

```
auto,penal=its_present_in_goal(data_structs,fecha,home,"home_team")
cada["penalty"]=penal
cada["own_goal"]=auto
lt.addLast(lista,cada)
return away_team,home_team,(away_team + home_team), lt.size(todos),merg.sort(lista,results_sort_criteria)
def its_present_partidos(lista,team):
n= lt.isPresent(lista,team)
if n==0:
lt.addLast(lista,team)
return lista
def its_present_in_goal(data_structs,fecha,equipo,condicion):
auto="unknown"
penal="unknown"
entry= mp.get(data_structs["model"]["goal_scorers_by_year"],fecha)
if entry:
entry= me.getValue(entry)
for cada in lt.iterator(entry):
equi= cada[condicion]
if equi==equipo:
if penal!="True":
penal=cada["penalty"]
if auto!="True":
auto= cada["own_goal"]
return auto,penal
```

Descripción

Se desea consultar los partidos que tuvo un equipo utilizando su nombre y un periodo entre dos fechas especificas

Entrada	• El nombre del equipo. • La fecha inicial del periodo a consultar • La fecha final del periodo a consultar

Salidas	• El total de equipos con partidos registrados.
	• El total de partidos disputados por el equipo.
	• El subtotal de partidos disputados como local.
	• El subtotal de partidos disputados como visitante.
	• El listado de los partidos disputados ordenados cronológicamente.
	  ○ Fecha del partido.
	  ○ Marcador del equipo local (goles del local).
	  ○ Marcador del equipo visitante (goles del visitante).
	  ○ Equipo local.
	  ○ Equipo visitante.
	  ○ País del encuentro.
	  ○ Ciudad donde se disputa el encuentro.
	  ○ Nombre del torneo asociado.
	  ○ Anotación, si el partido presento goles por faltas desde el punto penal.
	  ○ Anotación, si el partido presento autogoles.

Implementado (Sí/No)	Ximena Lopez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Primer for que itera sobre el mapa construido results ya previamente filtrado por team, cuya llave es la fecha. Contiene un if anidado que compara las llaves y obtiene los valores asociados. A su vez contiene un for que itera sobre los valores de las llaves	O(m)
Paso 2 Después hay un for que itera sobre los valores de las llaves y obtiene la fecha	O(n)
Paso 3 La fecha es utilizada por medio de un condicional para establecer el periodo de tiempo y crea las variables correspondientes a away_team, home_team y otras dos para almacenar los datos respectivamente.	O(1)
Paso 4 Condicional anidado que pregunta si el valor de away y team es el mismo, de ser así la variable away incrementa en uno, se pregunta si la variable autopenal está presente en goal, se asigna un valor a penal y a la variable correspondiente a almacenar los valores de autogol, finalmente se añade a lista.	O(1)
Paso 5	O(1)

Elif donde se pregunta si home es igual al valor en equipo de ser asi se repite el proceso realizado anteriormente con away, pero con las respectivas variables y valores.	
TOTAL	***O(n*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 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	IOS 11.7.10

Comparación de tiempo con Reto #1

TAMAÑO ENTRADA	**RETO 1 O(n**2)**	**RETO2 O(n*m)**
small	6,216	85,16333333
5pct	74,181	246,2
10pct	152,830	300,1633333
20pct	336,790	743
30pct	521,876	901,4066667
50pct	928,480	1523
80pt	1699.890	1626,966667
large	2289.890	1660,433333
small		

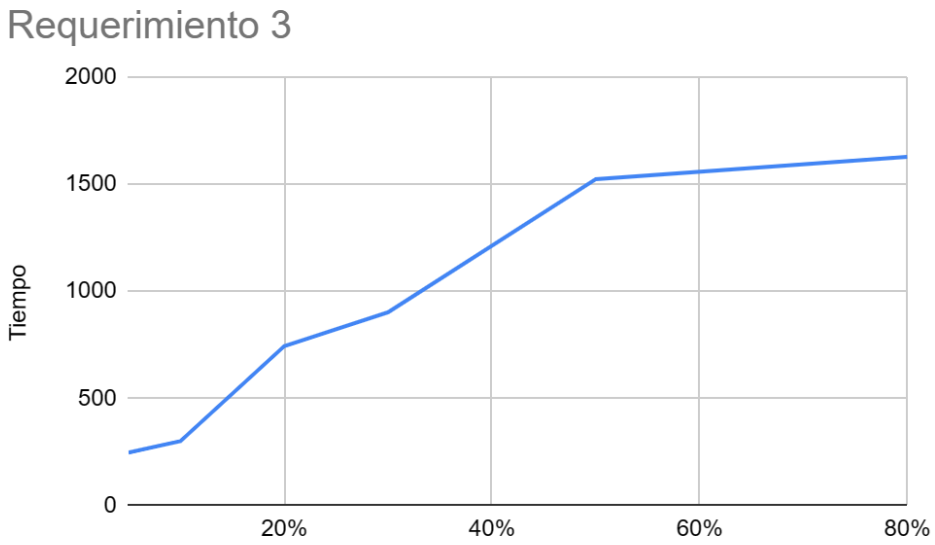
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	85,16333333
5pct	246,2
10pct	300,1633333
20pct	743
30pct	901,4066667
50pct	1523
80pt	1626,966667
large	1660,433333
small	

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

Este requerimiento implementado con mapas tiene menor complejidad que en el reto 1, para la carga de datos de large. Sin embargo, para la otra cantidad de datos es casi lo mismo. Al usar la carga de datos para que realice el prefiltrado por torneo es mucho más rápido de consultar los datos. Inicialmente, el crecimiento es mayor debido a que a medida que aumentan los datos, aumenta el número de equipos y de años con registro. No obstante, al fijarse en la segunda parte, es posible darse cuenta que la pendiente de la gráfica disminuye. Esto podría deberse a que, precisamente, la complejidad O(n*m) viene dado por el número de fechas y por el número de equipos. Es posible que, después de cierto punto, no se presenten muchos nuevos equipos o muchas nuevas fechas sobre las cuales se tenga que iterar. Debido a esto, podría explicarse la reducción en el crecimiento.

Requerimiento <<4>>

```
def req_4(data_structs, tournament, start_d, end_d):
start_y = date.fromisoformat(start_d).year
end_y = date.fromisoformat(end_d).year
map1 = data_structs['tournaments_by_year']
year_i = start_y
```

```
countries = lt.newList("ARRAY_LIST", compare_string)
cities = lt.newList("ARRAY_LIST", compare_string)
tournament_matches = lt.newList("ARRAY_LIST",compare_results_list)
shootout_matches = 0
while int(year_i)<=int(end_y):
if mp.contains(map1, str(year_i)):
map2= me.getValue(mp.get(map1, str(year_i)))
if mp.contains(map2, tournament):
map3 = me.getValue(mp.get(map2, tournament))
matches = me.getValue(mp.get(map3, 'total_matches'))
for match in lt.iterator(matches):
if start_d<match['date']<end_d:
match['winner']='Unavailable'
lt.addLast(tournament_matches, match)
if not lt.isPresent(countries, match['country']):
lt.addLast(countries, match['country'])
if not lt.isPresent(cities, match['city']):
lt.addLast(cities, match['city'])


year_i= str(int(year_i)+1)
shootouts = data_structs['shootouts_date']
merg.sort(tournament_matches, req4_sort_criteria)
for match in lt.iterator(tournament_matches):
m_date = date.fromisoformat(match['date'])
if mp.contains(shootouts, m_date):
shootout_list= me.getValue(mp.get(shootouts, m_date))
shootout = lt.isPresent(shootout_list,match)
if shootout:
shootout_matches+=1
```

```
winner=lt.getElement(shootout_list, shootout)['winner']

match['winner']=winner

n_tournaments = mp.size(data_structs['teams_tournament_year'])

n_matches = lt.size(tournament_matches)

n_countries = lt.size(countries)

n_cities = lt.size(cities)


return tournament_matches, n_tournaments, n_matches, n_countries, n_cities, shootout_matches
```

Descripción

Se desea consultar los partidos relacionados con un torneo utilizando el nombre respectivo y un periodo entre dos fechas especificadas.

Entrada	• Nombre del torneo. • La fecha inicial del periodo a consultar • La fecha final del periodo a consultar
Salidas	• El total de torneos con información disponible. • El total de partidos relevantes al torneo. • El total de países involucrados en el torneo. • El total de ciudades donde se disputan los partidos del torneo. • El total de partidos definidos por cobros de punto penal en el torneo. • El listado de los partidos disputados ordenados cronológicamente por fecha, nombre del por país y ciudad en que se disputaron los encuentros. ○ Fecha del partido. ○ País del encuentro. ○ Ciudad donde se disputa el encuentro. ○ Equipo local. ○ Equipo visitante. ○ Marcador del equipo local (goles del local). ○ Marcador del equipo visitante (goles del visitante). ○ Anotación, si el partido se definió por penales. ○ Nombre del equipo ganador por definiciones desde el punto penal (si existe).
Implementado (Sí/No)	Juan David Torres

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Comienza un while bajo la consigna de funcionamiento de que el año incial sea siempre menor al año final	O()
Paso 2 Contiene un if anidado donde se pregunta si el mapa1 contiene el año incial, de ser así se crea la variable mapa 2 donde se almacenan todas los valores del mapa en año 1	O(1)
Paso 3 Condicional anidado preguntando si el mapa2 contiene a torneo, de ser así se crea la variable mapa3 y matches, para almacenar los valores de total_matches en mapa3.	O(1)
Paso 4 For anidao que itera sobre matches.	O(M)
Paso 5 El if pregunta si la fecha en match es mayor que la fecha inicio y menor que la fecha final. De ser así crea la variable match[“winnne] como Unavailable y añade match a la lista de torneos. Contiene dos if anidados que preguntas si tanto countries como citites están presentes en match, de no ser así se agregan.	O(1)
Paso 6 Ordenamiento merge	O(nlogn)
Paso 7 Se realiza un for que itera sobre tournament_matches, que contiene dos if anidados. El primero pregunta sobre la contenencia de un valor dentro de un mapa y de ser así se crea la variable shootouts quien es una lista creada si está presente match en shootouts list.	
TOTAL	***O(NLOGN) ó O(NM), donde N es el número de años, y M es el número de partidos. La complejidad depende de qué tan grande es M o N. Sin embargo, debido a que M está previamente filtrado por los parámetros que entran como llave en el diccionario, será un número pequeño.***

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	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	12,0 GB
Sistema Operativo	Windows 11 Home Single Language

Comparación de tiempo con Reto #1

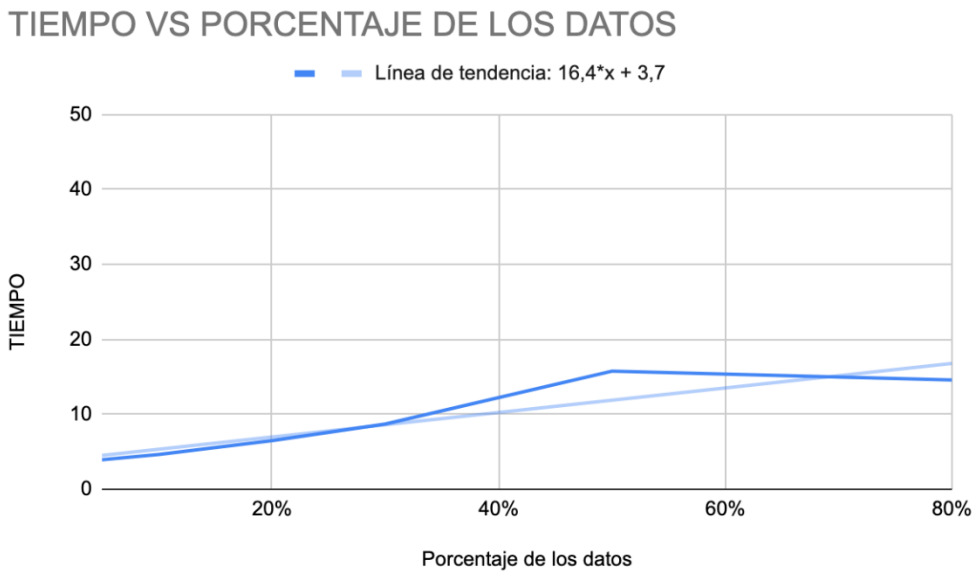
TAMAÑO ENTRADA	**RETO 1 O(nlogn)**	**RETO2 O(nlogn)**
small	0,73	1
5pct	3,80	3,950566649
10pct	9,02	4,633933425
20pct	11,85	6,491833289
30pct	17,22	8,697900017
50pct	27,29	16
80pt	36,16	14,57613329
large	51,03	41

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	1
5pct	3,950566649
10pct	4,633933425
20pct	6,491833289
30pct	8,697900017
50pct	16
80pt	14,57613329
large	41

Graficas



Análisis

En la gráfica se demuestra un comportamiento versátil. Inicialmente crece linearítmicamente, pero a medida que aumenta el volumen de datos el algoritmo se comporta de manera constante. El crecimiento inicial significativo se puede deber a que, a medida que aumenta el tamaño de los datos, aumentan los años presentes dentro del mapa con datos registrados, a su vez que el número de partidos guardados para cada año. Es posible que los datos específicos para las entradas usadas estuvieran, en su mayoría, presentes en los primeros archivos. Es decir, el "estancamiento" de la gráfica en la segunda parte podría deberse a que, a partir del archivo con el 50% de los datos, no hay muchas entradas nuevas que cumplan con los requisitos de búsqueda. Si analizamos la primera parte, es decir, de 1% a 50%, el crecimiento se asemeja a aquel de las funciones linearítmicas O(nlogn). Con esto, podríamos asumir que si en los archivos más grandes de datos aparecieran nuevos datos que coinciden con los criterios de búsqueda, la función seguiría creciendo de esta forma.

Requerimiento <<5>>

```
def req_5(data_structs, nombre, fecha_inicio, fecha_final):
"""
Función que soluciona el requerimiento 5
"""
anio_inicio = date.fromisoformat(fecha_inicio).year
anio_final = date.fromisoformat(fecha_final).year
```

```
periodo = lt.newList("ARRAY_LIST")
if mp.contains(data_structs["anotaciones_por_periodo"],nombre):
goles_entry = mp.get(data_structs["anotaciones_por_periodo"],nombre)
goles = me.getValue(goles_entry)
merg.sort(goles,cmp_crit_goal_req_2)
for fecha in lt.iterator(goles):
fecha_goles = date.fromisoformat(fecha["date"]).year
if fecha_goles >= anio_inicio and fecha_goles <= anio_final:
lt.addLast(periodo, fecha)
for cada_1 in lt.iterator(periodo):
fecha = cada_1["date"]
if mp.contains(data_structs["torneo_anio"],fecha):
fechas_entry = mp.get(data_structs["torneo_anio"], fecha)
fechas_values = me.getValue(fechas_entry)
for cada_2 in lt.iterator(fechas_values):
home_score = cada_2["home_score"]
away_score = cada_2["away_score"]
tournament = cada_2["tournament"]
cada_1["home_score"] = home_score
cada_1["away_score"] = away_score
cada_1["tournament"] = tournament
return periodo
else:
return "No se encuentran anotaciones realizadas por este jugador en el periodo de tiempo dado."
```

Descripción

Se desea consultar las anotaciones obtenidas por un jugador utilizando su nombre y un periodo entre dos fechas especificadas.

Entrada	• Nombre del anotador. • La fecha inicial del periodo a consultar

	• La fecha final del periodo a consultar
Salidas	• El total de jugadores con anotaciones registradas. • El total de anotaciones obtenidas por el jugador. • El total de torneos en que anoto el jugador. • El total de anotaciones obtenidas desde el punto penal. • El total de autogoles cometidos. • El listado de las anotaciones del jugador ordenadas cronológicamente por fecha y minuto en que se marcó el gol en el encuentro. ○ Fecha del partido. ○ Minuto en el que se marcó el gol. ○ Equipo local. o Equipo visitante. ○ Equipo del jugador. ○ Marcador del equipo local (goles del local). ○ Marcador del equipo visitante (goles del visitante). ○ Nombre del torneo donde se marcó el gol. ○ Tipo de anotación, si fue por falta desde el penal. ○ Tipo de anotación, si fue autogol.
Implementado (Sí/No)	Sofia Losada

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Comienza un if preguntando si el mapa anotaciones por periodo cuya llave es el nombre contiene al nombre del jugador que entra por parámetro. De ser así obtiene las llaves y valores de todas las anotaciones hechas por el jugador.	O(1)
Paso 2 Se realiza un ordenamiento merge	O(nlogn)
Paso 3 For anidado en donde se recorre la lista goles y se obtiene la fecha de la lista. Por medio de un if anidado se obtiene el periodo de tiempo, preguntando si la fecha es mayor a la fecha inicial y menor a la final. Finalmente se añaden los elementos que respondan a esta condición a una lista	O(N)
Paso 4 For que itera sobre la lista periodo que es donde se almacenan los valores de las anotaciones del jugador en el periodo establecido y se obtiene la fecha	O(M)
Paso 5	O(M)

Dentro del for hay un if anidado donde se pregunta si el mapa torneo_anio contiene la fecha obtenida. De ser así se obtienen las llaves y valores correspondientes	
Paso 6 Se itera sobre los valores obtenidos en un arreglo y se optiene el home_score, away_score y tournament, para contruír un diccionario	O(M)
Paso 7 Se añade el diccionario a la lista de retorno	O(1)
TOTAL	***O(n*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 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	IOS 11.7.10

Comparación de tiempo con Reto #1

TAMAÑO ENTRADA	**RETO 1 O (n**2)**	**RETO2 O(n*m)**
small	10.545	1,070333333
5pct	81.122	2,876666667
10pct	40.696	4,216666667
20pct	94.966	5,281333333
30pct	269.580	6,261166667
50pct	729.837	8,493066667
80pt	1585.216	12,22033333
large	3281.788	14,771

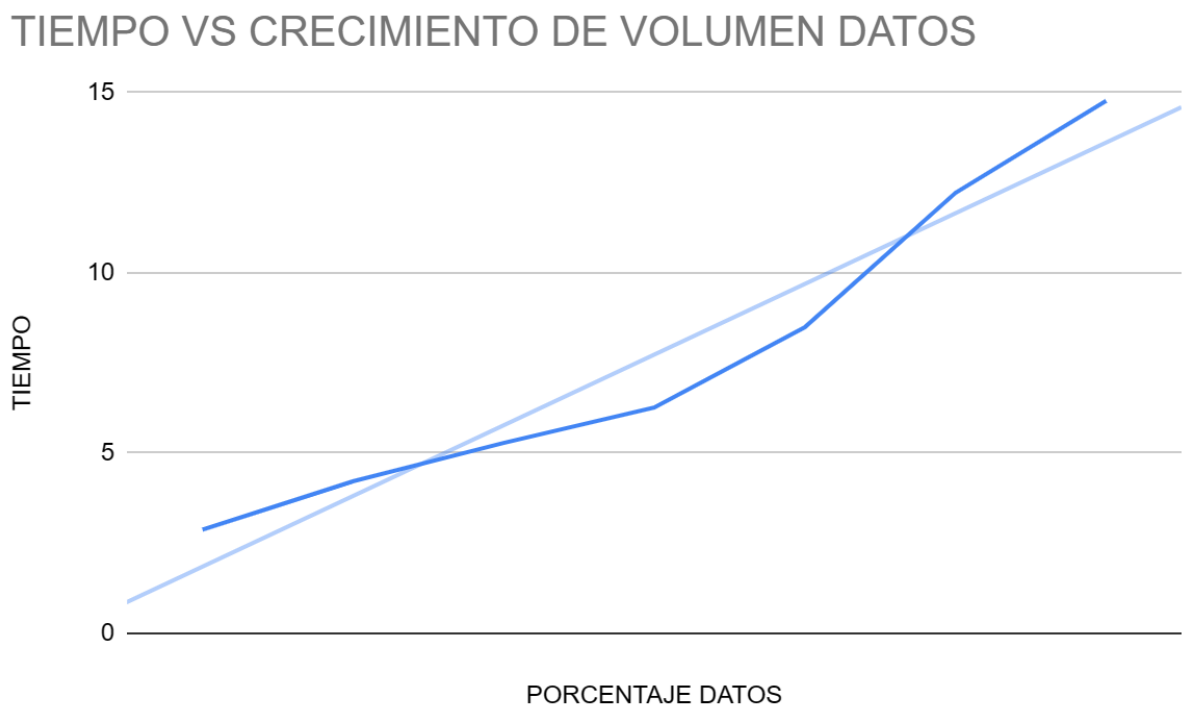
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	1,070333333
5pct	2,876666667
10pct	4,216666667

20pct	5,281333333
30pct	6,261166667
50pct	8,493066667
80pt	12,22033333
large	14,771

Graficas



Análisis

El comportamiento de la gráfica tiene una tendencia lineal. Esto se debe a que el algoritmo utiliza for iterativo anidado pero la cantidad de datos que recorre se reduce considerablemente mediante los condicionales. De esta manera, la longitud de m y n varía.

Requerimiento <<6>>

```
def req_6(data_structs, n_teams, tournament, year):
```

```
"""
Función que soluciona el requerimiento 6
"""
# TODO: Realizar el requerimiento 6
map_tournament = me.getValue(mp.get(data_structs['teams_tournament_year'],tournament))
scores_date = data_structs['scores_date']
map_year = me.getValue(mp.get(map_tournament, year))
teams = values_to_array(map_year)
for team in lt.iterator(teams):
matches = lt.iterator(team['match_info'])
for match in matches:
home_team = match['home_team']
away_team = match['away_team']
m_date = date.fromisoformat(match['date'])
if mp.contains(scores_date,m_date):
#Se encarga de mirar quiénes marcaron un gol en un partido específico.
#Esto para asegurar que el número de goles y partidos sean distintos (un jugador pudo haber marcado más de un gol en un partido)
scorer_list=lt.newList("ARRAY_LIST")
scores =me.getValue(mp.get(scores_date, m_date))
for score in lt.iterator(scores):
if score['home_team']==home_team and score['away_team']==away_team:
scorer = score['scorer']
if not lt.isPresent(scorer_list, scorer):
lt.addLast(scorer_list, scorer)
if not mp.contains(team['top_scorer'],scorer):
elem={"scorer":scorer, "goals":1, 'matches':0, 'avg_time':float(score['minute'])}
mp.put(team['top_scorer'],scorer, elem)
else:
dic = me.getValue(mp.get(team['top_scorer'],scorer))
```

```
dic['goals']+=1
dic['avg_time']= ((dic['avg_time']*(dic['goals']-1))+float(score['minute']))/dic['goals']
mp.put(team['top_scorer'],scorer, dic)
if score['own_goal']=="True":
team['own_goal_points']+=1
if score['penalty']=="True":
team['penalty_points']+=1


for scorer in lt.iterator(scorer_list):
dic = me.getValue(mp.get(team['top_scorer'],scorer))
dic['matches']+=1
mp.put(team['top_scorer'],scorer, dic)
merg.sort(teams, req6_sort_criteria)
size = lt.size(teams)
if n_teams>size:
first_teams = teams
else:
first_teams = lt.subList(teams,1,n_teams)
total_years = mp.size(map_tournament)
year_info = me.getValue(mp.get(data_structs['tournaments_by_year'],str(year)))
total_tournaments = mp.size(year_info)
n_teams_y = lt.size(teams)
tournament_info = me.getValue(mp.get(year_info,tournament))
total_matches = lt.size(me.getValue(mp.get(tournament_info, 'total_matches')))
n_countries = lt.size(me.getValue(mp.get(tournament_info,'countries')))
cities = me.getValue(mp.get(tournament_info, 'cities'))
n_cities = mp.size(cities)
k_vs = lt.newList("ARRAY_LIST")
for key in lt.iterator(mp.keySet(cities)):
```

```
lt.addLast(k_vs, mp.get(cities,key))

merg.sort(k_vs, pop_city_sort_criteria)

pop_city = me.getKey(lt.firstElement(k_vs))

return first_teams,total_years, total_tournaments, n_teams_y, total_matches, n_countries, n_cities, pop_city
```

Descripción

Se desea clasificar los N mejores equipos de un torneo dentro de un periodo de tiempo. Esto puede entenderse como el TOP ranking de cierta cantidad de equipos del torneo.

Entrada	• El número (N) de equipos para consulta • Nombre del torneo que se desea consultar • El año de consulta
Salidas	• El total de años calendarios disponibles en el historial de partidos. • El total de torneos disputados en el año de consulta. • El total de equipos involucrados en el torneo. • El total de encuentros disputados en el año. • El total de países involucrados en el torneo. • El total de ciudades involucradas en el torneo. • El nombre de la ciudad donde más partidos se han disputado. • El listado de los equipos que conforman el torneo debe estar ordenado por el criterio compuesto de sus estadísticas. ○ El total de puntos obtenidos ○ La diferencia de goles ○ El total de partidos disputados. ○ El total de puntos obtenidos desde la línea penal. ○ El total de puntos recibidos por autogol. ○ El total de victorias. ○ El total de empates. ○ El total de derrotas. ○ El total de goles obtenidos por sus jugadores. ○ El total de goles recibidos por el equipo. ○ El jugador con más anotaciones en el equipo con la siguiente información: ♦ Nombre del jugador. ♦ El total de goles anotados. ♦ El total de partidos donde anoto un gol. ♦ El promedio de tiempo (en minutos) para anotar los goles.
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Obtener mapa dando el torneo como llave.	O(1)
Paso 2 : Dentro del mapa de torneos, obtener los datos para un años específico	O(1)
Paso 3: Función auxiliar values_to_array (convierte los valores de un mapa a un array list), en este caso, contiene el nombre de todos los equipos.	O(n)
Paso 4: Itera a través de los equipos.	O(n)
Paso 5: Itera a través de los partidos asociados a ese equipo.	O(n)
Paso 6: Dada la fecha del partido, mira si hay un gol asociado a esa fecha.	O(1)
Paso 7: En caso de que sí hayan goles asociados a la fecha, itera a través de los goles asociados a esa fecha.	O(n)
Paso 8: Mirar si un jugador está presente dentro de la lista que lleva registro de los jugadores asociados a un año.	O(n)
Paso 9: Añadir un jugador al final de un array list. Ó obtener dicho jugador en caso de que ya exista.	O(1)
Paso 10: Iterar a través de los jugadores en un array_list	O(n)
Paso 11:Ordenar los equipos de acuerdo al criterio de ordenamiento (merge sort).	O(nlogn)
Paso 12:Obtener el número de años dentro del mapa del torneo.	O(1)
Paso 13: Obtener los torneos dado un año como llave.	O(1)
Paso 14: Obtener el tamaño de un array_list	O(1)
Dentro del mapa de torneos por año, obtener la información de un torneo dado el nombre del torneo.	O(1)
Dada la llave countries, obtener el tamaño de la lista que tiene guardados todos los países y ciudades que participaron en un torneo dado.	O(1)
Merge sort de las ciudades para obtener la ciudad más popular	O(nlogn)

TOTAL	***O(nlogn), comparado a O(n**2) del reto pasado.***

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	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	12,0 GB
Sistema Operativo	Windows 11 Home Single Language

Comparación de tiempo con Reto #1

TAMAÑO ENTRADA	**RETO 1**	**RETO2**
small	10,51	6,555533369
5pct	56,96	10,16466665
10pct	93,48	11,27149995
20pct	201,55	27,99036662
30pct	273,67	49,64439996
50pct	430,87	58,35643331
80pt	722,82	65,60903343
large	916,60	111,5029667

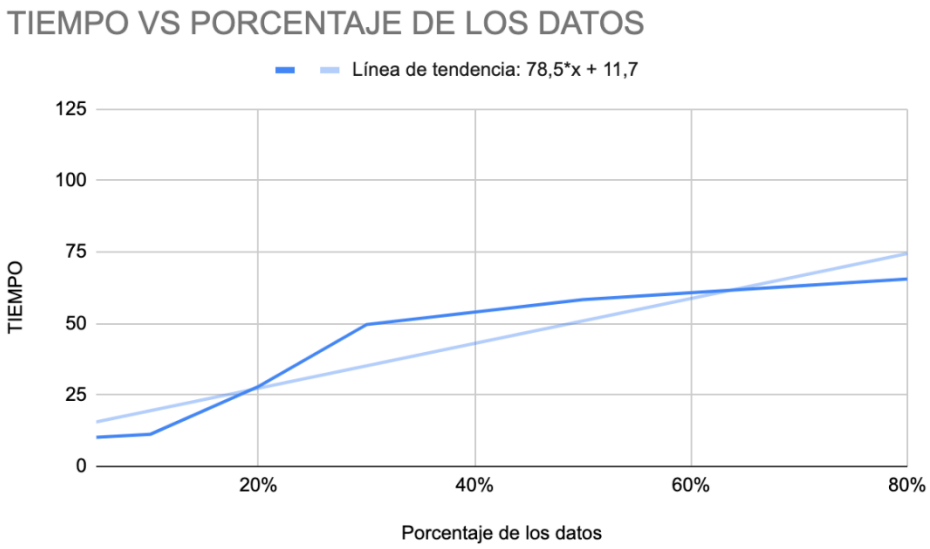
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	6,555533369
5pct	10,16466665
10pct	11,27149995
20pct	27,99036662
30pct	49,64439996
50pct	58,35643331
80pt	65,60903343
large	111,5029667

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

De acuerdo con la gráfica obtenida, vemos que se asemeja a la complejidad teórica de nlogn. Esto se debe a que la mayor complejidad temporal se encuentra en ordentar todos los países dado el criterio de comparación de los datos que almacenan. Sin embargo, a medida que aumenta al tamaño, pareciera que comienza a reducirse el crecimiento de la complejidad temporal, asemejándose a una función logarítmica. Esto puede deberse a que, realmente, hay un número limitado de equipos. Es decir, en el mundo hay aproximadamente 200 países, y debido a que el cálculo de las estadísticas de un país dado se lleva a cabo en la carga de datos, añadir más datos realmente no aumenta la complejidad temporal del **requerimiento** pues, el número de equipos no aumenta. Sin embargo, este aumento de datos sí se vería reflejado en la complejidad temporal de la carga de datos. En otras palabras, debido a que la carga de datos se encarga de realizar el cálculo de las estadísticas para un equipo dado, aumentar el tamaño de la muestra se vería reflejado en el tiempo que toma cargar los datos, mientras que en el requerimiento, simplemente encargado de obtener dichos datos previamente cargados, la complejidad no aumenta de forma tan significativa, pues el número de datos a retribuir (número de equipos) no crece después de cierto punto.

Requerimiento <<7>>

```
def req_7(data_structs,torneo,numero):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
```

```
mapa=mp.newMap(15,
maptype="PROBING",
loadfactor=0.5,
cmpfunction=compare_elements)
mapa_final= mp.newMap(200,
maptype="PROBING",
loadfactor=0.5,
cmpfunction=compare_elements)


lista_torneo= me.getValue(mp.get(data_structs["model"]["tournaments_7"],torneo))
mapa_scorers= data_structs["model"]["goal_scorers_by_year"]
goals=0
penalties= 0
autogoles=0
for key in lt.iterator(lista_torneo):
fecha=date.fromisoformat(key["date"])
goles= mp.get(mapa_scorers,fecha)
goals+= int(key["home_score"])
goals+= int(key["away_score"])
if goles:
goles=me.getValue(goles)
for cada in lt.iterator(goles):
if cada["away_team"]==key["away_team"] and cada["home_team"]==key["home_team"] :
if str(cada["penalty"])=="True":
penalties+=1
if str(cada["own_goal"])=="True":
autogoles+=1
nombre= cada["scorer"]
equipo= cada["team"]
```

```
entry= mp.get(mapa_final,nombre)
if entry:
valor=me.getValue(entry)
else:
valor=name(nombre)
mp.put(mapa_final,nombre,valor)
mapa=mapa_final_borrar(mapa,valor)
valor= valores_req_7(cada,valor,key,equipo)
mapa=mapa_final_añadir(mapa,valor)
total_tourn= mp.size(data_structs["model"]["tournaments_7"])
total_scorers= mp.size(mapa_final)
total_matches= lt.size(lista_torneo)

return total_tourn,total_scorers,total_matches,goals,penalties,autogoles,requerimiento_7(mapa,numero)
```

Descripción

Se desea encontrar los jugadores de futbol con N puntos dentro de una competencia especifica. Esto puede entenderse como consultar los jugadores con puntaje especifico dentro de un torneo.

Entrada	• Nombre del torneo que se desea consultar (incluyendo amistosos). • El puntaje (N) de los jugadores dentro del torneo (ej.: 3, 5, 10 o 20).
Salidas	• El total de torneos disponibles para consultar. • El total de anotadores que participaron en el torneo. • El total de partidos dentro del torneo. • El total de anotaciones o goles obtenidos durante los partidos del torneo. • El total de goles por penal obtenidos en ese torneo • El total de autogoles en que incurrieron los anotadores en ese torneo • El listado de anotadores debe estar ordenado por el criterio compuesto de sus estadísticas. ○ El nombre del anotador.

	○ El puntaje que obtiene el jugador como anotador ○ El total de goles anotados. ○ El total de goles anotados por penales. ○ El total de autogoles anotados. ○ El tiempo promedio para anotar en minutos. ○ El total de torneos en que anotó el jugador. ○ El total de anotaciones obtenidos en una victoria. ○ El total de anotaciones obtenidos en un empate. ○ El total de anotaciones obtenidos en una derrota. ○ Ultimo gol anotado por el jugador con la siguiente información: ♦ Fecha del encuentro. ♦ Nombres de los equipos local y visitante. ♦ Puntaje de los equipos local y visitante. ♦ Minuto en que anotó el gol. ♦ Detalles técnicos del gol (si fue por falta desde el punto penal o autogol).
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Se itera con un for por la lista de partidos jugados en el torneo ingresado	O(n)
Paso 2: Se busca la fecha de cada uno de los partidos dados anteriormente	O(1)
Paso 3: se itera con un for anidado por cada uno de los goles de ese partido	O(m)
Paso 4: Se realizan algunos if para confirmar información	O(1)
Paso 5: Se crea un diccionario para agregar a la lista del mapa que tiene la llave del jugador	O(1)
Paso 6: si ya está el diccionario solo se edita los datos	O(1)
	O()
TOTAL	***O(n*m), comparado a O(n**2) del reto pasado.***

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 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	IOS 11.7.10

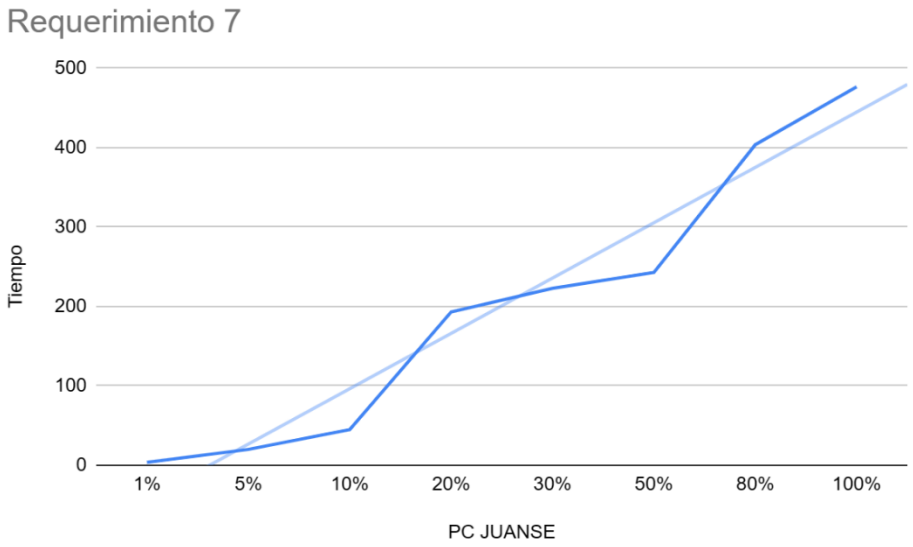
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	3,604666667
5pct	19,95333333
10pct	44,96666667
20pct	192,9966667
30pct	222,8
50pct	242,8
80pt	403,7333333
large	476,5666667

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

La complejidad del requerimiento implementada con mapas solo hace un recorrido ya previamente filtrado por el team, así que lo que recorre es muy pequeño. Además, solo se busca la llave de la fecha para los goles y se itera bajo cada uno de los goles por cada partido. El requerimiento no es muy lento sin importar que se use gran cantidad de datos. Esto supone una mejora con respecto al reto pasado, donde se tenía que recorrer la totalidad de los datos y comparar si una entrada coincidía con los datos que se pedían. Sin embargo, debido a que en el mapa se cuenta con unas condiciones iniciales de acceso instantáneo a una lista filtrada, las iteraciones que se hacen se ejecutan a través de un tamaño de datos reducido.

Requerimiento <<8>>

```
ef req_8(data_structs, team, start_y, end_y):

"""

Función que soluciona el requerimiento 8

"""

# TODO: Realizar el requerimiento 8

map1 = data_structs['team_year_info']

y_i = start_y

n_years = 0

team_map = me.getValue(mp.get(map1, team))

home_matches = 0

away_matches = 0



years_list = lt.newList("ARRAY_LIST")

while int(y_i)<=int(end_y):

if mp.contains(team_map,int(y_i)):

n_years+=1

year_dic = me.getValue(mp.get(team_map, int(y_i)))

home_matches+=year_dic['home_matches']
```

```
away_matches+=year_dic['away_matches']
lt.addLast(years_list, year_dic)
y_i = str(int(y_i)+1)
sort_years_first(years_list)
results = data_structs['results_date']


first_year_list= me.getValue(mp.get(team_map, lt.firstElement(years_list)['year']))['dates']
last_year_list = me.getValue(mp.get(team_map, lt.lastElement(years_list)['year']))['dates']
merg.sort(first_year_list,dates_new_first_criteria)
merg.sort(last_year_list, dates_new_first_criteria)
oldest_date = lt.lastElement(last_year_list)
newest_match_d =lt.firstElement(first_year_list)


merg.sort(years_list, req8_sort_criteria)
first_year_list= me.getValue(mp.get(team_map, lt.firstElement(years_list)['year']))['dates']
last_year_list = me.getValue(mp.get(team_map, lt.lastElement(years_list)['year']))['dates']


results_list = me.getValue(mp.get(results, newest_match_d))
for element in lt.iterator(results_list):
if element['home_team']==team or element['away_team']==team:
newest_match = element
total_matches = home_matches+away_matches


return years_list, n_years, total_matches, home_matches, away_matches, oldest_date, newest_match
```

Descripción

Como analista de futbol Deseo comparar el desempeño histórico anual (año por año) de dos selecciones nacionales en torneos oficiales (excluyendo los juegos amistosos) hasta una fecha específica.

Entrada	• Nombre del equipo. • El año inicial del periodo a consultar • El año final del periodo a consultar
Salidas	• Los años que comprende el historial entre las fechas especificadas. • El total de partidos disputados por el equipo. • El total de partidos disputados por el equipo como local. • El total de partidos disputados por el equipo como visitante. • La fecha del último partido más antiguo. • Los detalles del partido más reciente con la siguiente información: ○ Fecha del partido. ○ Equipo local o Equipo visitante. ○ Marcador del equipo local (goles del local). ○ Marcador del equipo visitante (goles del visitante). ○ País del encuentro. o Ciudad ○ Nombre del torneo asociado. • El listado de las estadísticas anuales del equipo ordenados cronológicamente del más reciente al más antiguo dentro del periodo a consultar. Las estadísticas que se deben listar son: ○ El total de partidos disputados en el año. ○ El total de puntos obtenidos en el año. 11 ○ La diferencia de goles. 12 ○ El total de puntos obtenidos desde la línea penal. ○ El total de puntos recibidos por autogol. ○ El total de victorias en el año. ○ El total de empates en el año. ○ El total de derrotas en el año. ○ El total de goles obtenidos por sus jugadores. ○ El total de goles recibidos por el equipo. ○ El jugador con más anotaciones en el año con la siguiente información: ♦ Nombre del jugador. ♦ El total de goles anotados.
Implementado (Sí/No)	SI

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Obtener mapa del equipo de acuerdo a su nombre.	O(1)
Paso 2 : While que recorre los años ingresados por el usuario.	O(y), donde y es el número de años.
Paso 3: Mirar si el mapa del equipo contiene la llave del años dado.	O(1)
Paso 4: Obtener el diccionario con los datos asociados a ese año, dando el año como llave.	O(1)
Paso 5: Añadir de último el año a la lista que guarda registro de todos los años (Array list).	O(1)
Paso 6: Usar la función auxiliar sort_years_first para ordenar los años del más nuevo al más antiguo	O(y*logy) Donde y es el número de años.
Paso 7: Obtener la lista, para el primer y el último año, que contiene todos los partidos asociados a ese año.	O(1)
Paso 8: Iterar a través de los partidos asociados a una fecha específica.	O(n)
TOTAL	***O(n), comparado a O(n**2) del reto pasado.***

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	**11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz**
Memoria RAM	12,0 GB
Sistema Operativo	Windows 11 Home Single Language

Comparación de tiempo con Reto #1

TAMAÑO ENTRADA	**RETO 1**	**RETO2**
small	0,99	1
5pct	5,58	2
10pct	10,86	2,711599906

20pct	28,97	2,421866655
30pct	38,83	3
50pct	85,49	3,025233269
80pt	123,97	2,957133293
large	160,90	3

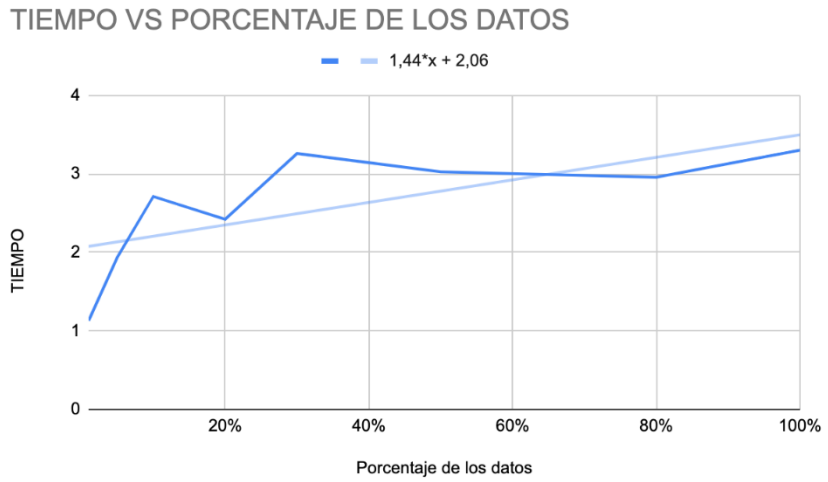
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (s)**
small	1
5pct	2
10pct	2,711599906
20pct	2,421866655
30pct	3
50pct	3,025233269
80pt	2,957133293
large	3
small	

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.

En la gráfica se puede ver que, inicialmente, presente un crecimiento en el tiempo de ejecución a medida que los datos incrementan. No obstante, con los mayores volúmenes de datos, el tiempo de ejecución pareciera permanecer constante. Esto puede deberse a que, inicialmente, es posible que

aumentara de forma significativa los datos asociados a los parámetros de búsqueda introducidos, mientras que en los otros volúmenes de datos no hubieran tantos datos asociados. De la misma forma, la complejidad O(n) está dada por los partidos asociados a una fecha específica. Debido a que inicialmente se hace un filtro por fecha, dada la llave dentro del mapa, el tamaño de la lista que se recorre realmente no es muy grande, es decir, no hay un aumento significativo. De la misma forma, se puede concluir que el merge sort que se hace con el rango de años realmente no afecta el crecimiento de complejidad a medida que crecen los años. Esto se debe a que, debido a que se usarion los mismos parámetros de búsqueda, el número de años permanece constante a lo largo de las pruebas. En resumen, el crecimiento de ejecución temporal en este requerimiento se debe principalmente a que, dada una fecha, es probable que hayan más datos asociados a ella. El número de años permanece constante, por lo que el while que itera a través de ellos, y el ordenamiento que se hace de ellos, no cambian de complejidad y permanececen constantes.