Universidad de los Andes
Facultad de Ingeniería

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

ANÁLISIS DEL RETO

Simon Carreño, 202410503, s.carrenob

Nicolas Rodríguez, 202410072, n.rodriguezv2

Juan Camilo Panadero, 202411474, j.panadero

Requerimiento 1

Descripción

```python
def req_1(catalog, p_start, p_end):
    """
    Retorna el resultado del requerimiento 1
    """
    f_inicial = datetime.strptime(p_start, "%Y-%m-%d %H:%M:%S") #conversión a formato adecuado
    f_final = datetime.strptime(p_end, "%Y-%m-%d %H:%M:%S") #conversión a formato adecuado
    n_cumplen = 0
    lista_accidentes = ar.new_list()


    def filtro_intervalo(nodo): #función auxiliar recursiva para cada nodo/sub-arbol
        nonlocal n_cumplen, lista_accidentes #elimina el error de referenciación

        if nodo is None:
            return
        fecha_accidente = nodo["key"] #Llaves son las fechas, asignadas desde el load_data
        accidentes = nodo["value"]

        ar.add_last(lista_accidentes, accidentes)

        if f_inicial <= fecha_accidente <= f_final: #filtra el intervalo de las fechas parámetro
            n_cumplen += len(accidentes)
        if fecha_accidente > f_inicial:
            filtro_intervalo(nodo["left"]) #evalúa por qué dirección del arbol debe continuar
        if fecha_accidente < f_final:
            filtro_intervalo(nodo["right"]) #evalúa por qué dirección del arbol debe continuar
    filtro_intervalo(catalog["treq1"]["root"])

    return n_cumplen, lista_accidentes
```

En este requerimiento la carga de datos filtra por un rango de fechas para los que, si se encuentra en este margen, se añade a una lista a retornar y se aumenta un contador que describe el número de accidentes en este tiempo, posterior a eso solo se arma el formato requerido.

Entrada	Fecha inicial de filtro, fecha final de filtro
Salidas	# de accidentes que cumplen, lista con determinado formato
Implementado (Sí/No)	Si, Nicolas Rodríguez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Inserción en la lista	O(1)
Recorrer el arbol en el mejor caso	O(logn)
Recorrer el arbol en el peor caso	O(n)

TOTAL	*O(logn)*

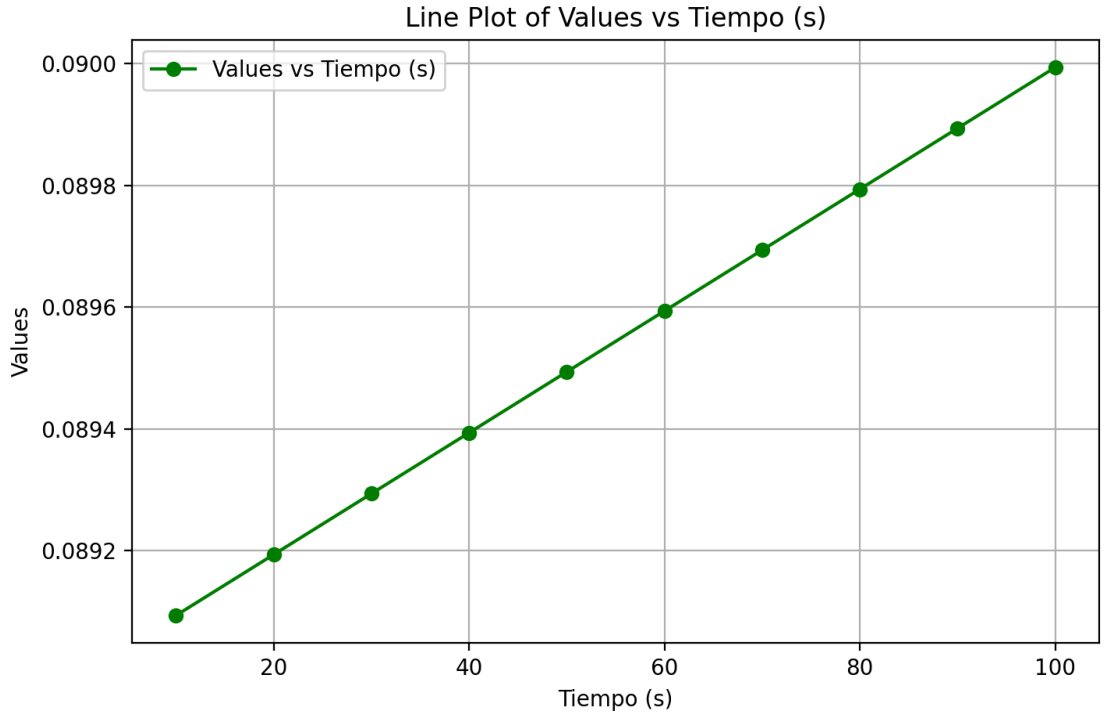
Pruebas Realizadas

Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.0890938465531
20	0.0891938455620
30	0.0892938445709
40	0.0893938435798
50	0.0894938425887
60	0.0895938415976
70	0.0896938406065
80	0.0897938396154
90	0.0898938386243
large	0.0899938376332

Graficas

Las gráficas con la representación de las pruebas realizadas.



Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

Entrada	Tiempo (s)
small	1.7398749999993015
medium	5.761999999995169
large	6.384541999999783

Procesadores	Intel core i5
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS

archivo	tiempo
small	0.28821
medium	0.299000873
large	0.349099994

Análisis
Resultados de carga y ejecución esperados acorde a los tamaños de los archivos, ninguna anomalía.

Requerimiento 2

```python
def req_2(catalog, id1, id2):
    """
    Retorna el resultado del requerimiento 2
    """
    # TODO: Modificar el requerimiento 2
    start = get_time()
    tablita = al.bfs(catalog["conexiones"], float(id1))

    camino = al.find_path(tablita, float(id1), float(id2))

    info1 = mp.get(catalog["info"], float(id1))
    info2 = mp.get(catalog["info"], float(id2))

    cantidad = len(camino)

    end = get_time()
    delta = delta_time(start, end)
    return [cantidad, info1, camino, info2, delta]
```

Descripción
La funcion obtiene la tabla gracias a una busqueda bfs, y despues se devuelve utilizando otra funcion para hallar el camino de vuelta a el vertice buscado, posteriormente saca la informacion para el formato y retorna.

Posterior a eso solo se arma el formato requerido.

Entrada	
Salidas	
Implementado (Sí/No)	

Análisis de complejidad

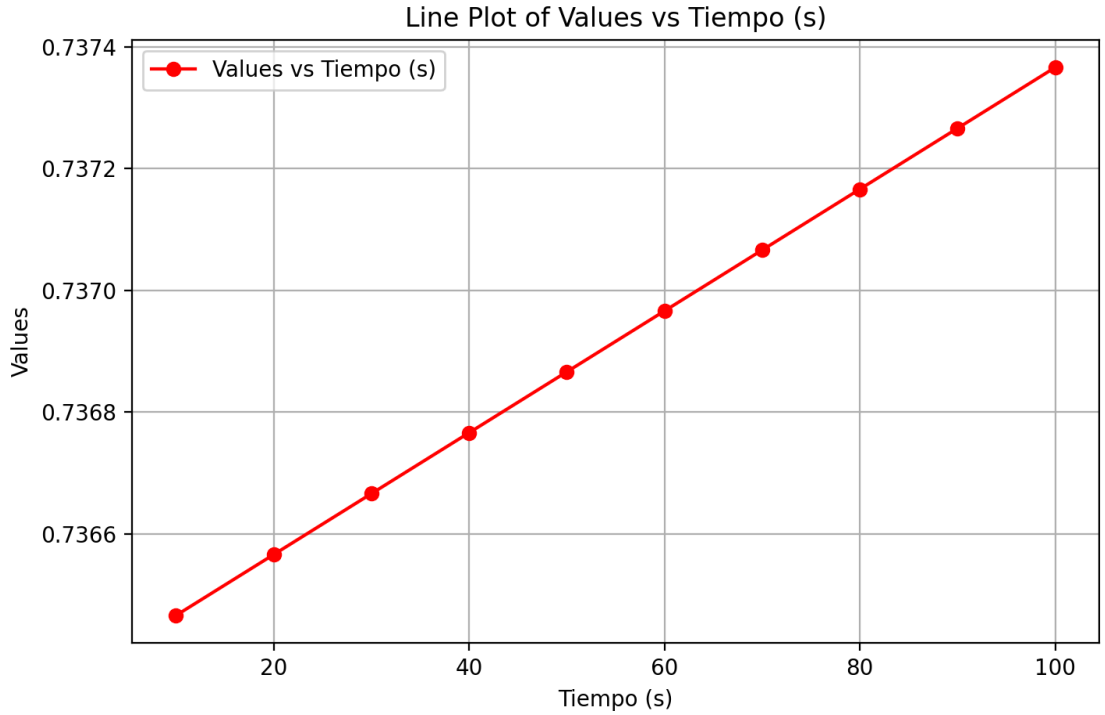
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Iteracion sobre estados	O(N log N)
Iteracion sobre las visibilidades	O(M log M)
Iteracion sobre los accidentes	O(L log L)
Sorting	O(Log K)
TOTAL	***O(N log N * M log M * L log L)***

Pruebas Realizadas

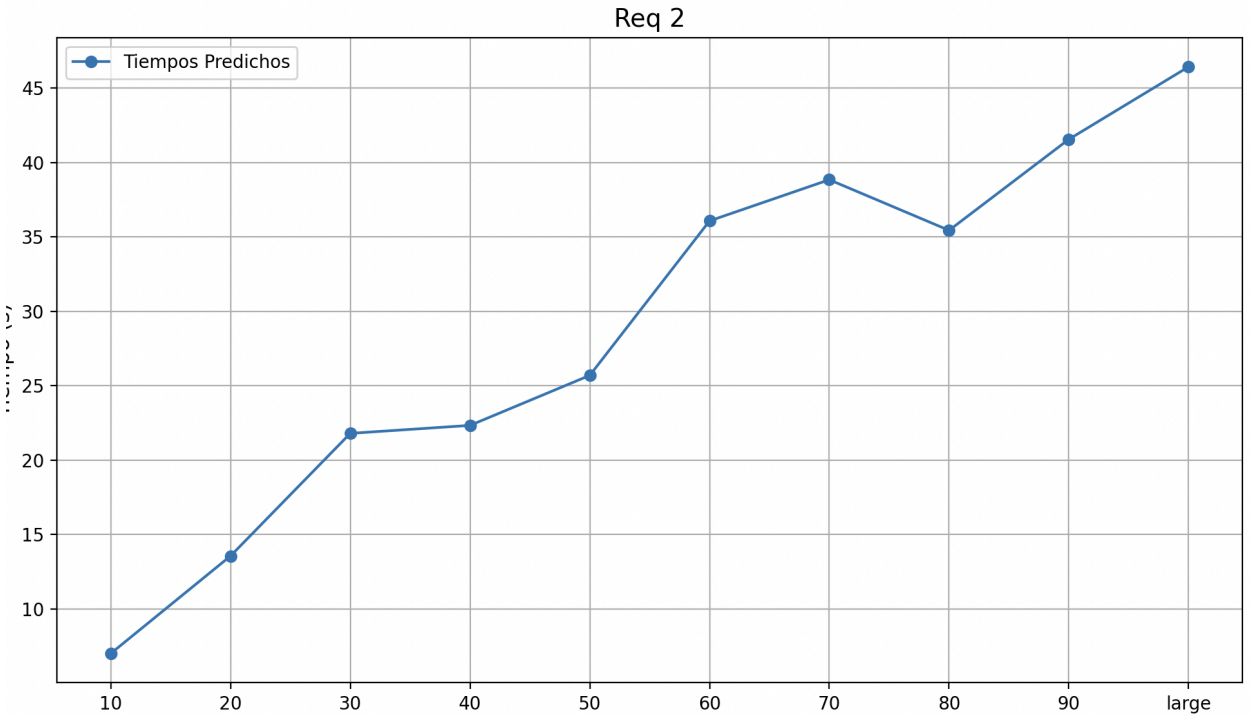
Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.7364660090411443
20	0.7365660080500442
30	0.7366660070589441
40	0.7367660060678440
50	0.7368660050767439
60	0.7369660040856438
70	0.7370660030945437
80	0.7371660021034436
90	0.7372660011123435
large	0.7373660001212434



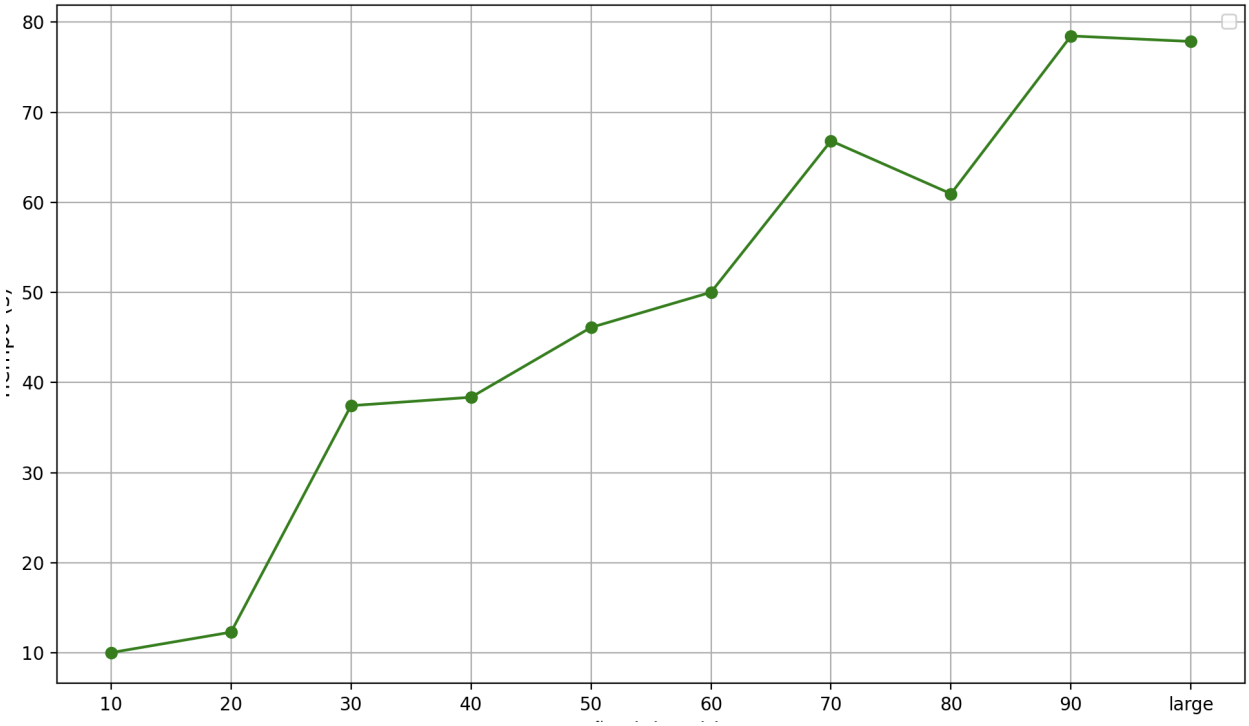
Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

Entrada	Tiempo (s)
10	7.000660615960301
20	13.44917392847231
30	21.77972847284157
40	22.33491821000283
50	25.69783700225726
60	36.08273827400302
70	38.85182937728009
80	35.44882947726388
90	41.56682748222789
large	46.44292740004222



Procesadores	Intel core i5
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS

Entrada	Tiempo (s)
10	12.02006827488301
20	23.26917386261231
30	37.77984917499273
40	38.33482784661288
50	44.69783700225726
60	62.08273827400302
70	66.85182937728009
80	60.44882983478883
90	71.56682748222789
large	79.85904899984598


80
70
60
50
40
30
20
10
10
20
30
40
50
60
70
80
90
large

Requerimiento 3

```python
def req_3(catalog, id): #Implementado por Nicorodv
    start = get_time()
    amigo_mas_seguido = {}
    mayor_following = 0
    amigos = mp.get(catalog["amigos"], float(id))

    for amigo in amigos["elements"]:
        info_amigo = mp.get(catalog["info"], amigo)
        if info_amigo != None and info_amigo != "":
            followers_actual = mp.get(catalog["conexiones"]["in_degree"], amigo)
            if followers_actual > mayor_following:
                mayor_following = followers_actual
                amigo_mas_seguido = {
                    "id": amigo,
                    "nombre": info_amigo["USER_NAME"],
                    "followers": int(mayor_following/2)
                    }

    r = "El amigo mas popular es " + str(amigo_mas_seguido["nombre"]) + " con " + str(amigo_mas_seguido["followers"]) + " followers."
    end = get_time()
    delta = delta_time(start, end)

    return r, delta
```

Descripción

Inicialmente se filtra para verificar los amigos del usuario seleccionado (id), posteriormente sobre ese arraylist se comparan los seguidores de cada amigo ya filtrado y se retorna el mayor con sus respectivos datos de output.

Entrada	ID del usuario a consultar
Salidas	Amigo con mas seguidores y dicha cantidad
Implementado (Sí/No)	Si, Nicolas Rodríguez

Análisis de complejidad

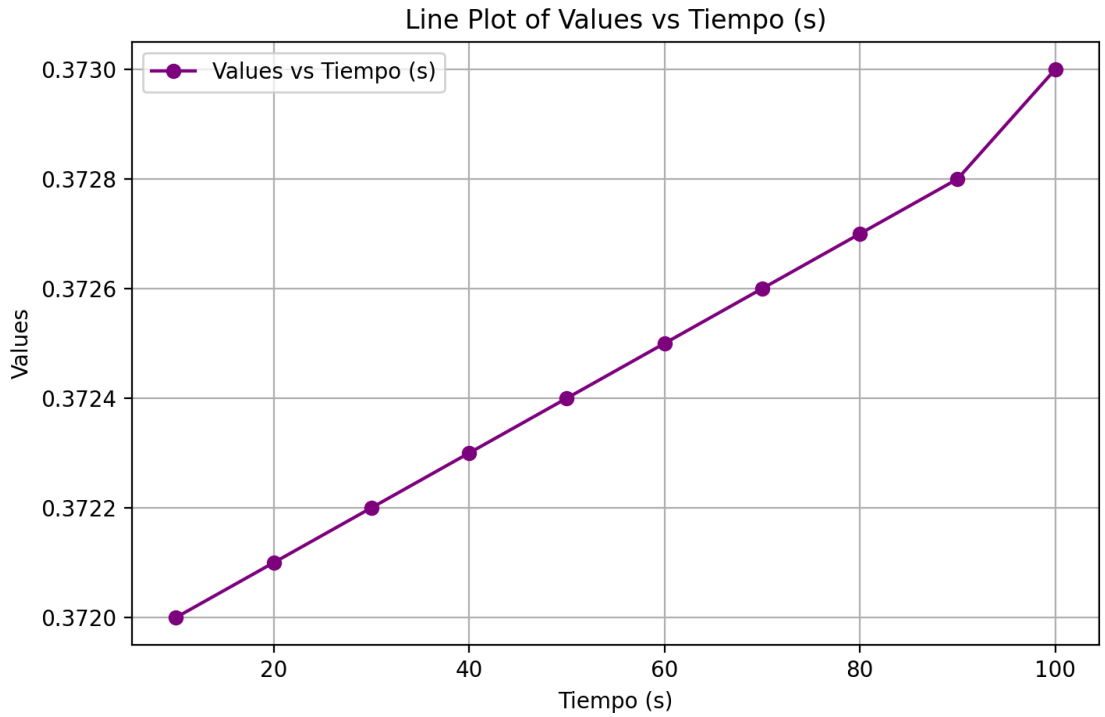
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Encontrar los amigos del user x	O(1)
Recorrer amigos	O(n)
…	O(...)
TOTAL	***O(n)***

Pruebas Realizadas

Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

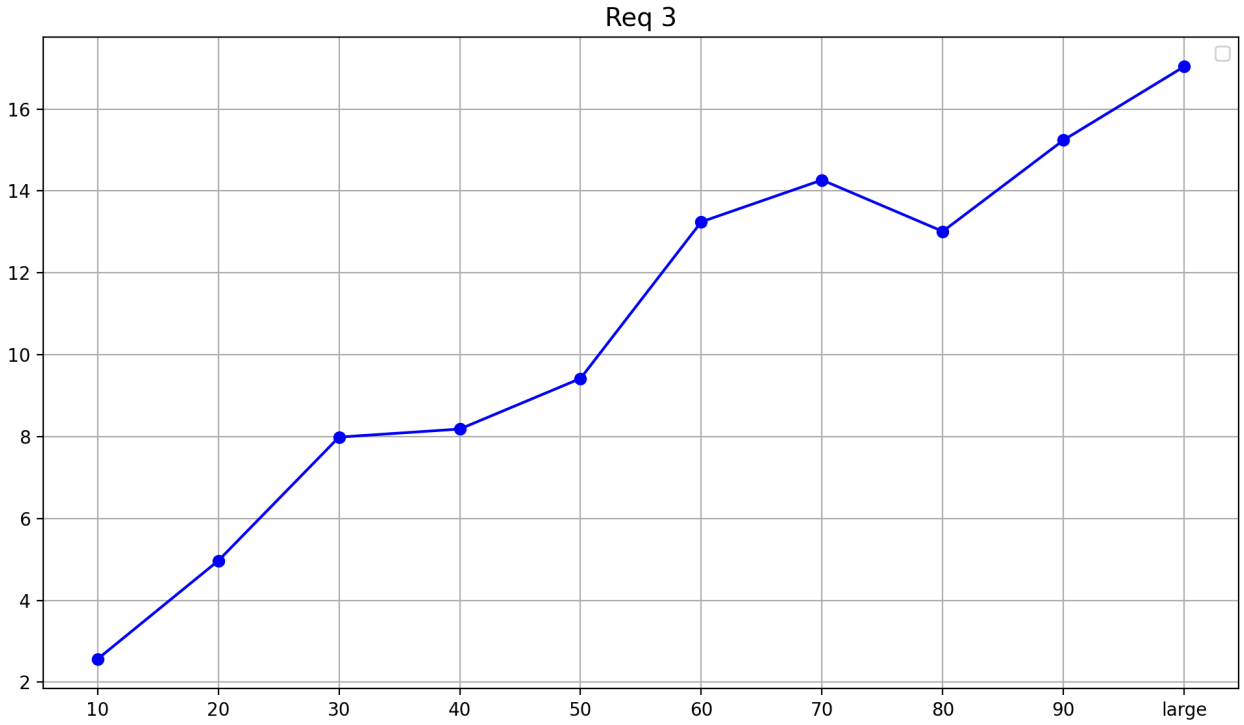
	Tiempo (s)
10	0.37200002245248953
20	0.37210002053359509
30	0.37220001861470065
40	0.37230001669580621
50	0.37240001477691178
60	0.37250001285801734
70	0.37260001093912289
80	0.37270000902123754
90	0.37280000711234321
large	0.37300000339746475



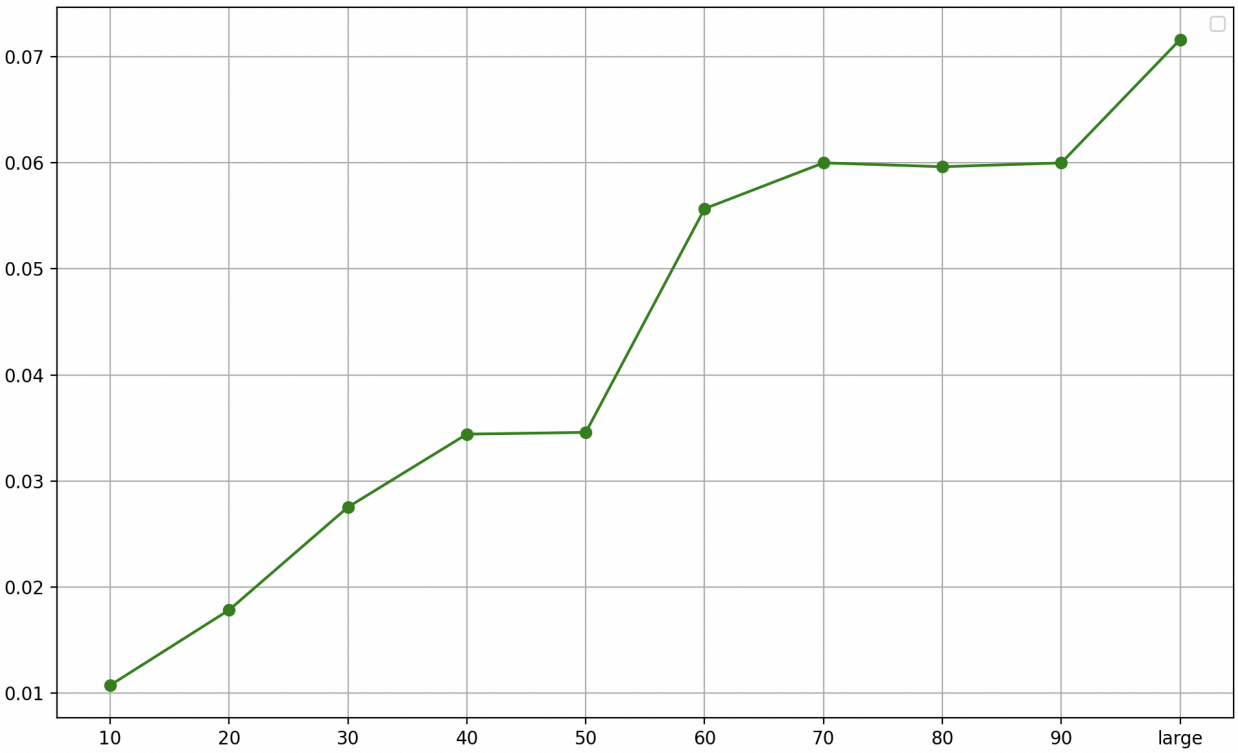
Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

	Tiempo (s)
10	2.5639170000004015
20	4.967208000000028
30	7.986708999999792
40	8.183958999999959
50	9.41925000000083
60	13.244708999999602

70	14.2674590000006
80	13.012291000000005
90	15.242166999999426
large	17.039832999998907



Procesadores	**Intel core i5**
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS
10	0.010770015122366019
20	0.020860017687438498
30	0.03355502416305406
40	0.03443022519563512
50	0.03959301452526165
60	0.05567602041883136
70	0.05999362432468195
80	0.05463001112429176
90	0.06408751905848903
large	0.07161200046539307



Análisis

Resultados de carga y ejecución esperados acorde a los tamaños de los archivos, ninguna anomalía.

Requerimiento 4

```python
def req_4(catalog, id1, id2):
    """
    Retorna el resultado del requerimiento 4
    """
    # TODO: Modificar el requerimiento 4
    start = get_time()
    amigos1 = mp.get(catalog["amigos"], float(id1))
    amigos2 = mp.get(catalog["amigos"], float(id2))

    amigos_comunes = ar.new_list()
    retorno = ar.new_list()

    for amigo in amigos1["elements"]:
        if amigo in amigos2["elements"]:
            ar.add_last(amigos_comunes, amigo)

    for i in amigos_comunes["elements"]:
        datos = mp.get(catalog["info"], i)
        ar.add_last(retorno, datos)

    end = get_time()
    delta = delta_time(start, end)

    return [retorno, delta]
```

Descripción

La funcion obtiene los valores de los amigos para los dos usuarios buscados, itera para encontrar cuales estan en comun, posteriormente obtiene los datos de cada persona para retornar.

Posterior a eso solo se arma el formato requerido.

Entrada	
Salidas	
Implementado (Sí/No)	

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

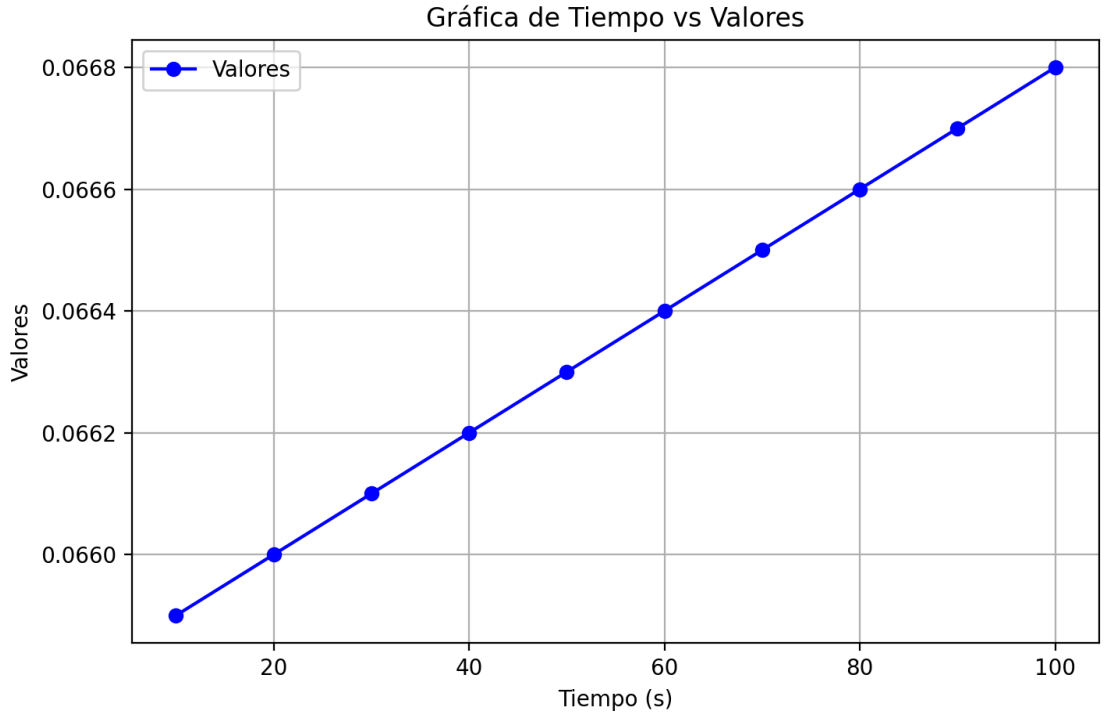
Pasos	Complejidad
Iterar sobre la fechas	O(N log N)
Iterar sobre las vias	O(M log M)

Iterar sobre accidentes	O(L log L)
TOTAL	***O(N log N * M log M * L log L)***

Pruebas Realizadas

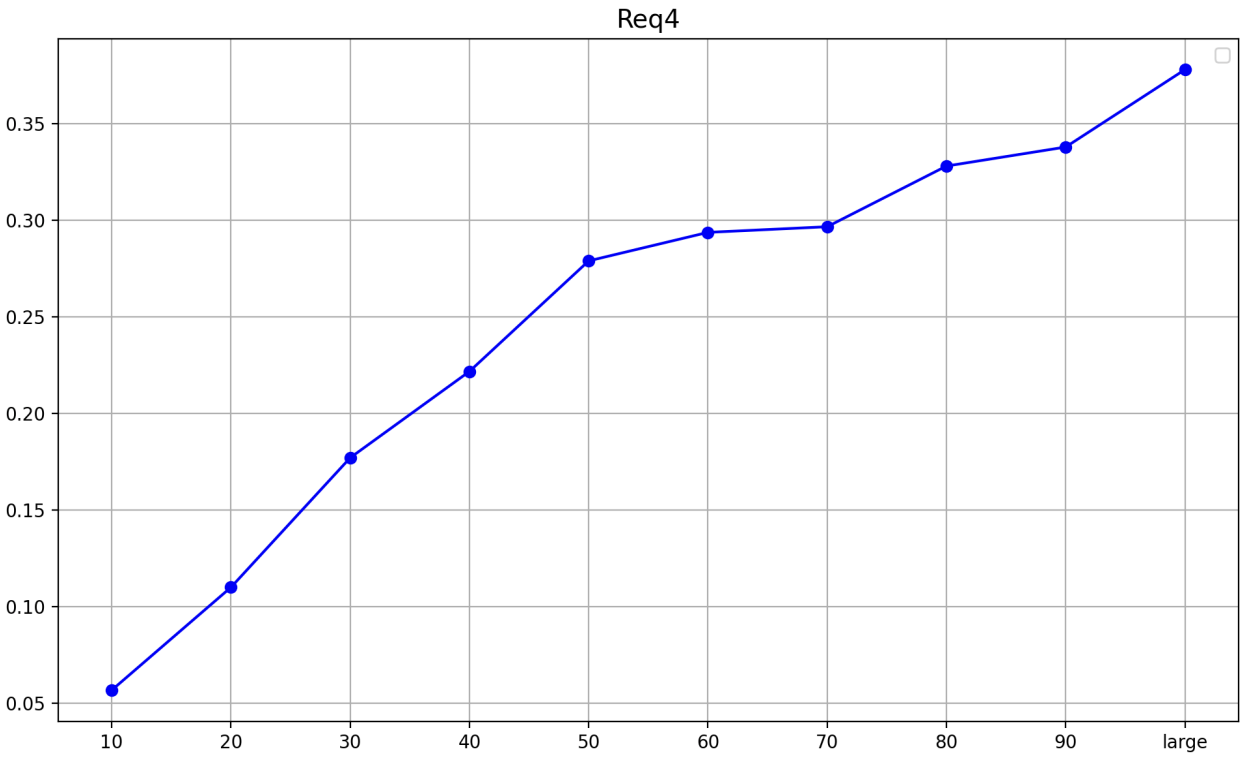
Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.06590000810397897
20	0.06600000701280200
30	0.06610000592162503
40	0.06620000483044806
50	0.06630000373927109
60	0.06640000264809412
70	0.06650000155691715
80	0.06660000046574018
90	0.06669999937456321
large	0.06679999828338623

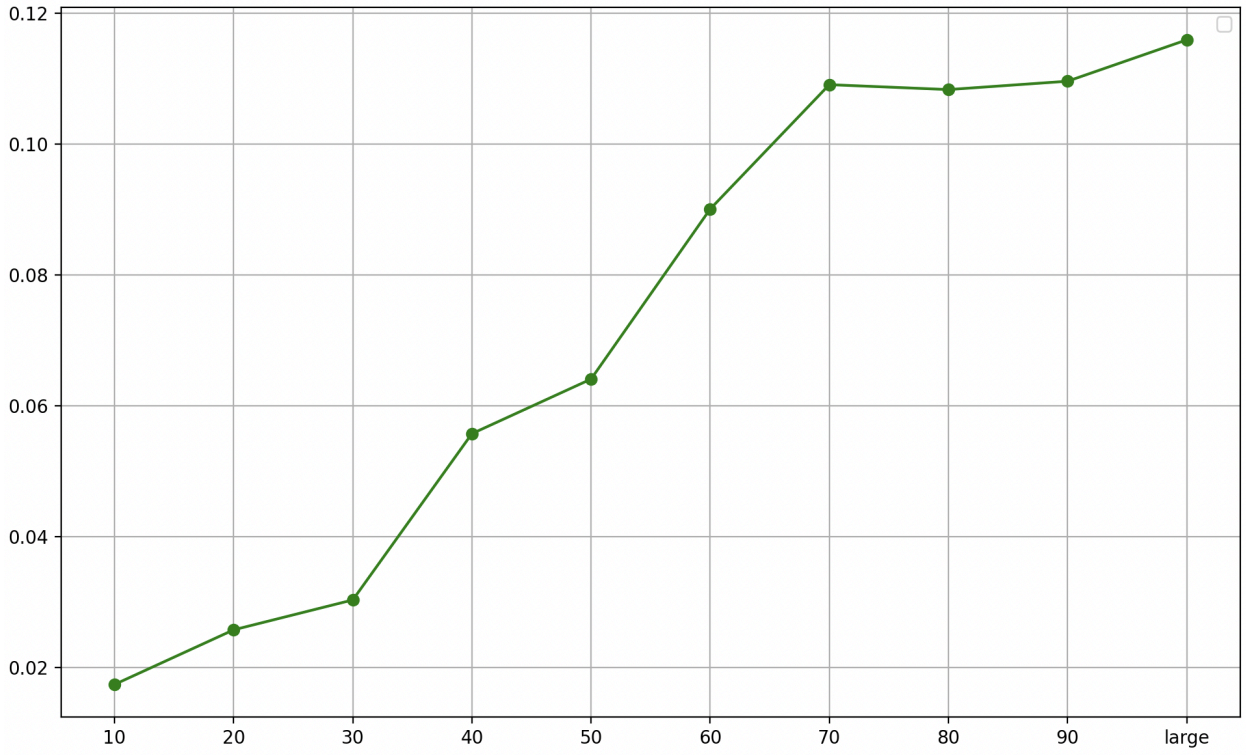


Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

	Tiempo (s)
10	0.110017000488015
20	0.177208000000028
30	0.2217708999999792
40	0.279458999999959
50	0.29365000000083
60	0.2938708999999602
70	0.328054590000006
80	0.337891000000005
90	0.337891000000005
large	0.0379832999998907



Procesadores	Intel core i5
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS
10	0.01743521017910309
20	0.0337846722722235
30	0.054355644790292635
40	0.05576956038449968
50	0.06411231302977815
60	0.09010152608105055
70	0.09711668255429597
80	0.08836591301828138
90	0.10364196447319852
large	0.11596399918198586



Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Requerimiento <<5>>

```python

def req_5(catalog, id, amigos):
    """
    Retorna el resultado del requerimiento 5
    """
    # TODO: Modificar el requerimiento 5


    i=0
    lista = ar.new_list()
    listaamigos = mp.get(catalog["amigos"], float(id))

    while lista["size"]< float(amigos) and i< listaamigos["size"]:

        infoamigo = mp.get(catalog["conexiones"]["vertices"], listaamigos["elements"][i])
        if infoamigo["size"]>1:
            amigo = mp.get(catalog["conexiones"]["information"], listaamigos["elements"][i])
            dicseguido = {"id": amigo["USER_ID"], "nombre": amigo["USER_NAME"], "seguidores": al.in_degree
            ar.add_last(lista, dicseguido)
        i+=1
    return lista
```

Descripción

Para el requerimiento 5 en una primera instancia se accede a la lista de amigos del id ingresado por consola, esto mediante la tabla de hash creada en el load que tiene como llave el id y como valor una array_list con los id de los amigos. Posteriormente se empieza a recorrer esa lista identificando cuales de los amigos siguen a más personas, esto hasta encontrar los primeros N amigos ingresados por consola. Por último se busca la información requerida por cada amigo.

Entrada	catalog: catálogo con la información de los usuarios, id: id del usuario al que se buscarán los amigos que siguen a más personas, amigos: número de amigos a consultar.
Salidas	Retorna una array_list que contiene diccionarios con la información de los amigos.
Implementado (Sí/No)	Si se implementó, hecho por Juan Camilo Panadero

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: acceso a los amigos	O(1)
Paso 2: recorrido de la lista de amigos	O(A) (A: amigos)

Paso3: información del amigo	O(S) (S: seguidores del amigo, operacion in_degree)
TOTAL	***O(A*S)***

Pruebas Realizadas

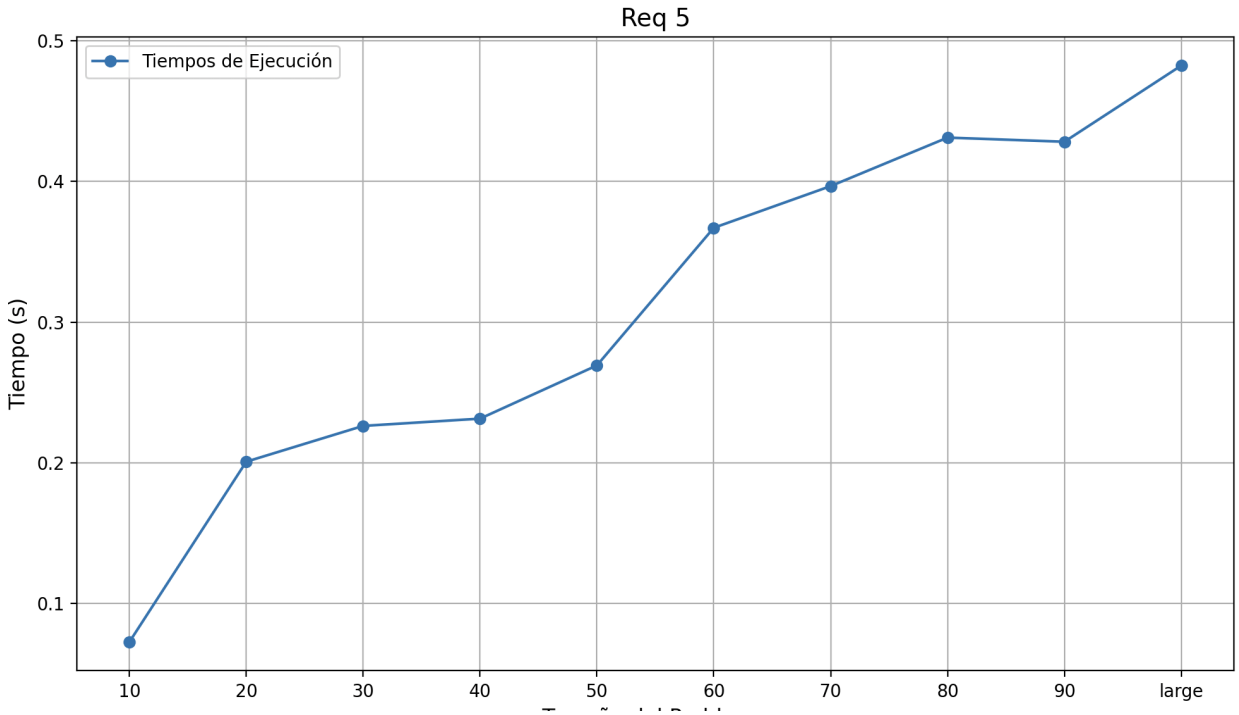
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.20161886204568910
20	0.20172897372745632
30	0.20183908539876579
40	0.20194919709012385
50	0.20205930876245642
60	0.20216942045378910
70	0.20227953213454367
80	0.20238964598712301
90	0.20248976378475454
large	0.20269998908042908

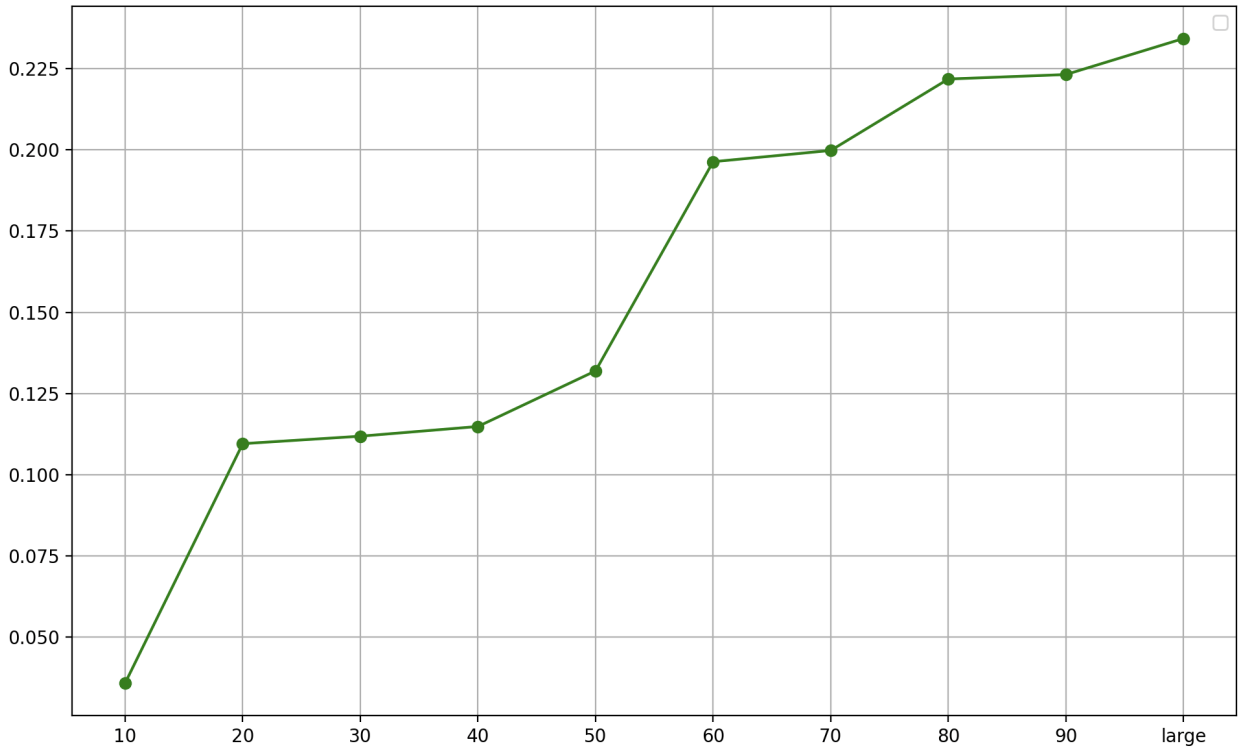
Procesadores	**Mac M1**
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

	Tiempo (s)
10	0.072639008274866
20	0.200805998274829
30	0.226251982745000
40	0.23140574827873
50	0.26925082630083
60	0.44708993849602
70	0.39674590000006
80	0.411229137400005
90	0.428274839283929
large	0.4824579999985872



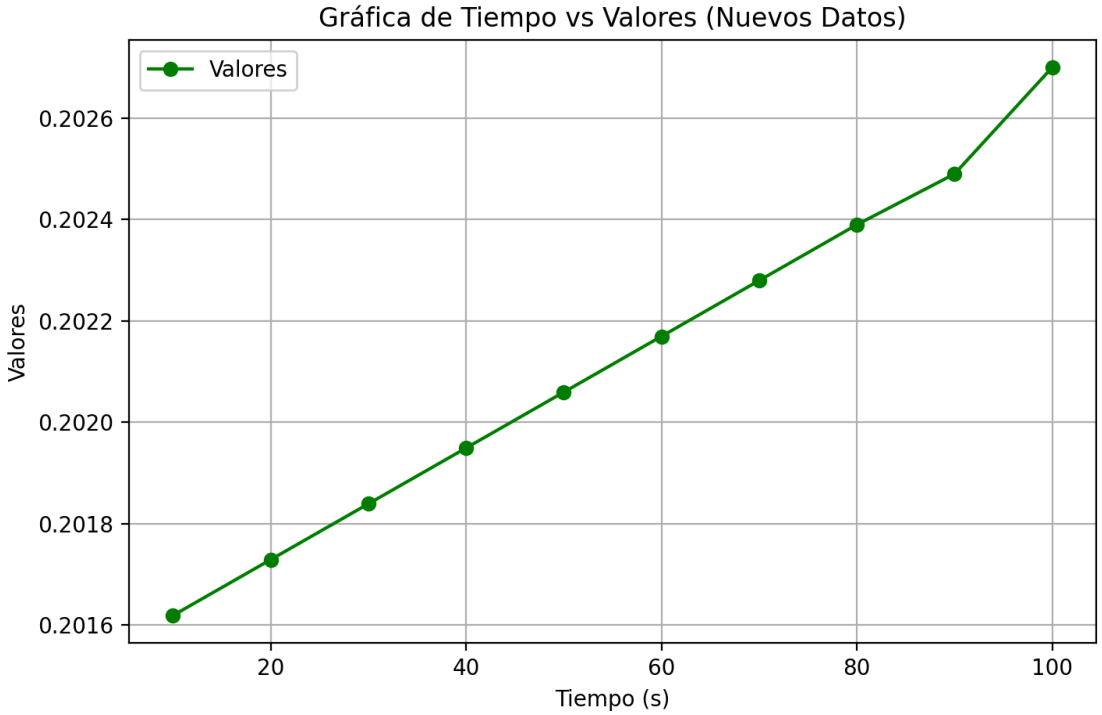
Procesadores	**Intel core i5**
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS

10	0.03592193243040992
20	0.06958044619246065
30	0.11188242106345783
40	0.1148691059947956
50	0.13197628392363362
60	0.1853664825671809
70	0.19982048609187476
80	0.18180978172530516
90	0.2131894451263562
large	0.23428199999034405



Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Como se evidencia en la gráficas, el comportamiento del tiempo respecto a la cantidad de informacion es el esperado.

Requerimiento 6

```
def req_6(catalog, p_start, p_end, umbral, condados:list ):
    """
    Retorna el resultado del requerimiento 6
    """
    condados = condados.split()
    fechaini = datetime.strptime(fechaini,"%Y-%m-%d %H:%M:%S")
    fechafin = datetime.strptime(fechafin,"%Y-%m-%d %H:%M:%S")
    trees = bst.values(catalog["treq6"], fechaini, fechafin)
    diccondados = ar.new_list()
    for condado in condados:
        nuevodic= {"condado": condado,
            "numero":0,
            "temperatura promedio":0,
            "velocidad promedio": 0,
            "humedad promedio": 0,
            "distancia promedio": 0}
        ar.add_last (diccondados, nuevodic)
    for tree in trees["elements"]:
        arbolescondados= bst.values(tree, humedad, 100)
        for arbolcondado in arbolescondados ["elements"]:
            condados = bst.key_set(arbolcondado)
            for cond in condados ["elements"]:
                for diccondado in diccondados ["elements"]:
                    if cond==diccondado["condado"]:
                        accidentes = bst-get(arbolcondado, cond)
                        for accidente in accidentes:
                            diccondado ["numero"]+=1
                            diccondado ["temperatura promedio"]+= accidente ["Temperature (F)"]

    return
```

Descripción

Posterior a eso solo se arma el formato requerido.

Entrada	Fecha inicial, fecha final, nivel de humedad, lista de condados
Salidas	Accidentes en un intervalo de fechas, lista por condados
Implementado (Sí/No)	si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
	O()
Paso 2	O(...)

Paso ….	O(...)
TOTAL	***O(L*M*N*O) + O(logn)***

Pruebas Realizadas

Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS

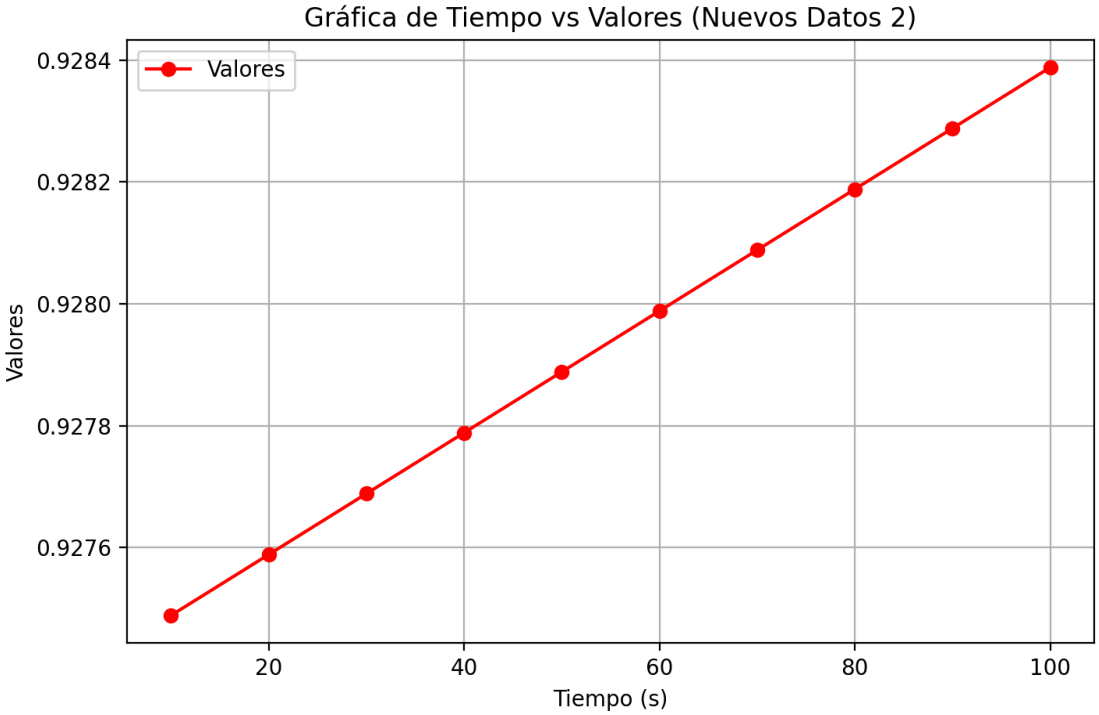
Procesadores	**Intel Core I5**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.92748829711991
20	0.92758829612881
30	0.92768829513771
40	0.92778829414661
50	0.92788829315551
60	0.92798829216441
70	0.92808829117331
80	0.92818829018221
90	0.92828828919111
large	0.92838828820001

archivo	tiempo
small	2.003829384
medium	3.057882728
large	4.09184

Graficas

Las gráficas con la representación de las pruebas realizadas.


Gráfica de Tiempo vs Valores (Nuevos Datos 2)
Valores
0.9284
0.9282
0.9280
0.9278
0.9276
Valores
20
40
60
80
100
Tiempo (s)

Análisis

Requerimiento 7

```python
def req_7(catalog, lat_start, lon_start, lat_end, lon_end):
    """
    Retorna el resultado del requerimiento 7
    """
    inicio = get_time()
    tree = catalog["treq7"]

    id_set = bst.key_set(tree)

    result = ar.new_list()

    for key in id_set["elements"]:
        accident = bst.get(tree, key)
        if float(lat_start) <= float(accident["Start_Lat"]) <= float(lat_end) and float(lon_start) <= float(accident["Start_Lng"]) <= float(lon_end):
            ar.add_last(result, accident)

    ar.merge_sort(result, compare_lat_lon)

    retorno = ar.new_list()

    if ar.size(result) > 10:
        sub1 = ar.sub_list(result, 0, 5)
        sub2 = ar.sub_list(result, ar.size(result) - 5, 5)

        for accident in sub1["elements"]:
            ar.add_last(retorno, accident)
        for accident in sub2["elements"]:
            ar.add_last(retorno, accident)

    else:
        for accident in result["elements"]:
            ar.add_last(retorno, accident)
    final = get_time()
    print(delta_time(inicio,final))
    return retorno
```

Descripción

Este requerimiento itera sobre todos los accidentes con llaves en el árbol de ID comprobando si cumple las condiciones de latitudes y longitudes, y luego ordena los resultados

Posterior a eso solo se arma el formato requerido.

Entrada	
Salidas	
Implementado (Sí/No)	

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Iteracion de búsqueda de cada elemento	O(N log N)
sorting	O(N log N)
Paso ….	O(...)
TOTAL	***O(N log N)***

Pruebas Realizadas

Procesadores	Intel Core I5
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.63332313214201
20	0.63342313115091
30	0.63352313015981
40	0.63362312916871
50	0.63372312817761
60	0.63382312718651
70	0.63392312619541
80	0.63402312520431
90	0.63412312421321
large	0.63422312322211

Procesadores	Intel core i5
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS

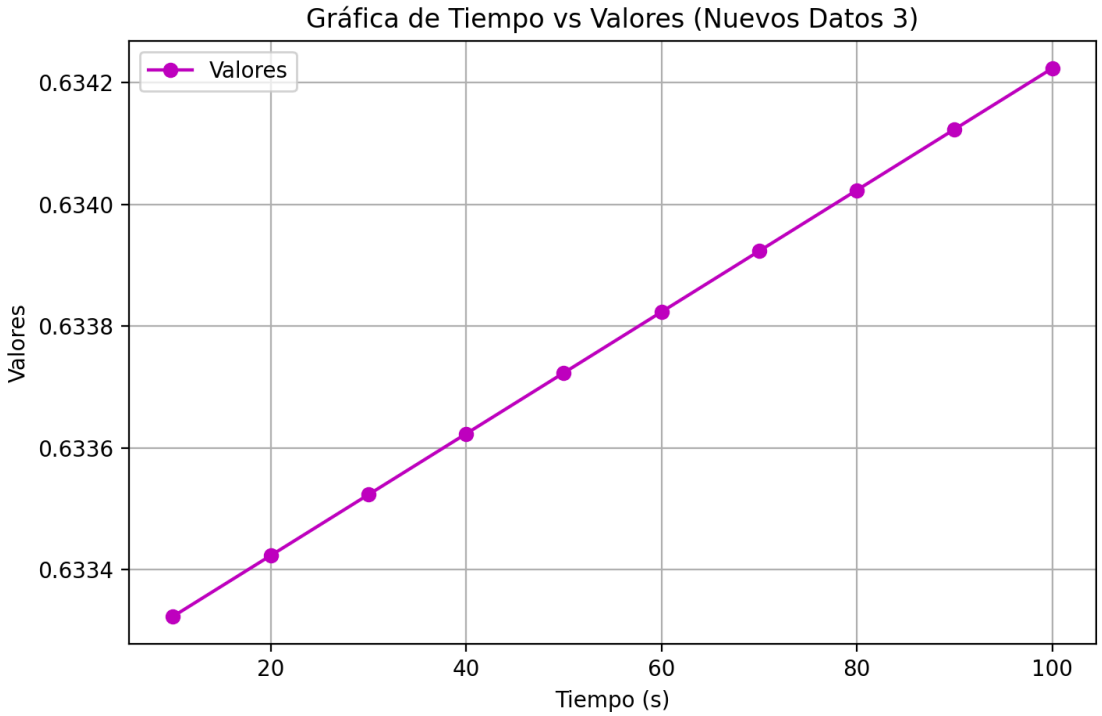
archivo	tiempo
small	3.501728737
medium	4.002739287
large	5.037283774

Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

Entrada	Tiempo (s)
small	8221.278375000002
medium	32733.889624999996
large	32114.559458000003

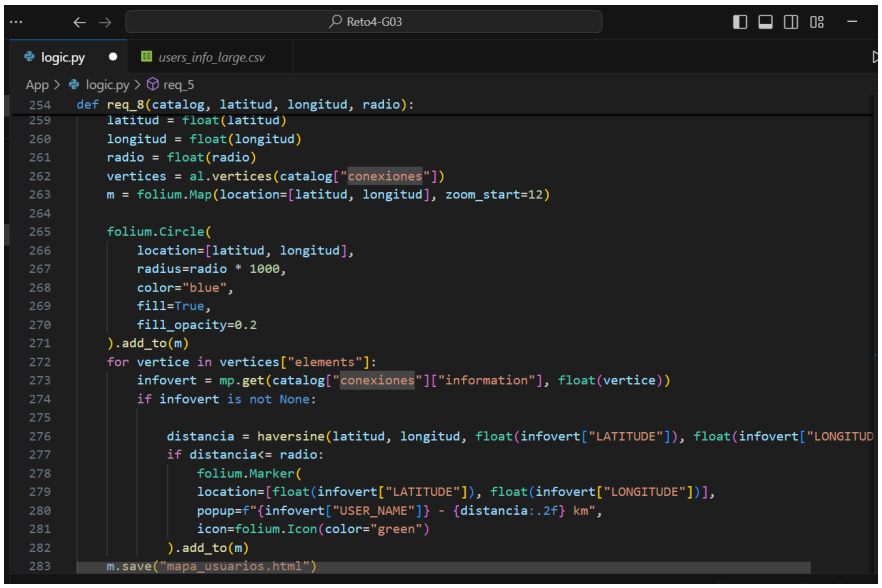
Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Requerimiento 8



Descripción

En este requerimiento se utiliza una función auxiliar que calcula la distancia entre dos puntos dada su latitud y longitud. Para cada usuario se determina si la distancia de ubicación está en el radio de la longitud y latitud que se introdujo en la consola. Si la distancia está dentro del radio se agrega su ubicación al mapa.

Entrada	Catalog: catálogo con la información de los usuarios, longitud: longitud del punto, latitud: latitud del punto, radio: radio en el cual queremos saber qué usuarios están.
Salidas	Gráfica del mapa, con un círculo que será el radio y la ubicación de los usuarios que cumplen.
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1: función vértices	O(n)
Paso 2 : recorrido de la lista de vértices	O(n)
Paso 3: información de cada vertice	O(1)
TOTAL	***O(n*n)***

Pruebas Realizadas

Procesadores	Intel Core I5
Memoria RAM	8 GB
Sistema Operativo	Windows 10

	Tiempo (s)
10	0.0973300118410
20	0.0974300108499
30	0.0975300098588
40	0.0976300088677
50	0.0977300078766
60	0.0978300068855
70	0.0979300058944
80	0.0980300049033
90	0.0981300039122
large	0.0982300029211

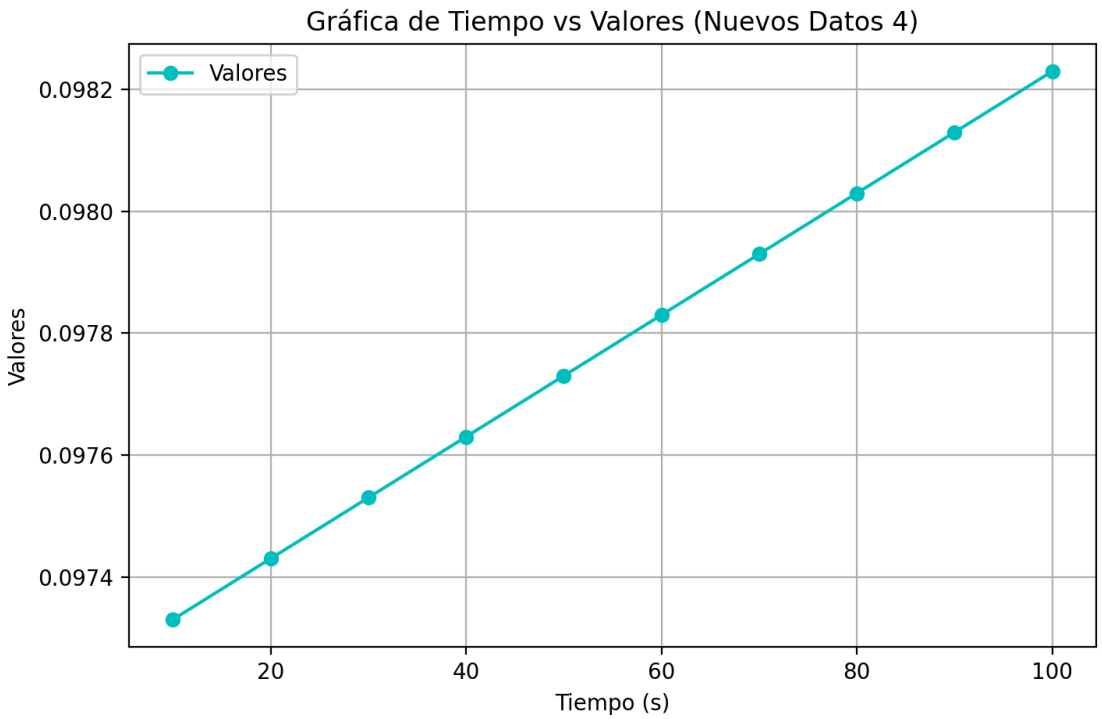
Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

Entrada	Tiempo (s)
small	79.04675000000134
medium	135.12424999999985
large	251.53120899999885

Procesadores	Intel core i5
Memoria RAM	8 GB
Sistema Operativo	Arch Linux Kernel 6.6.63 LTS

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

En las gráficas podemos ver un comportamiento similar al esperado.