ANÁLISIS DEL RETO

Simon Carreño, 202410503, email 1

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)***

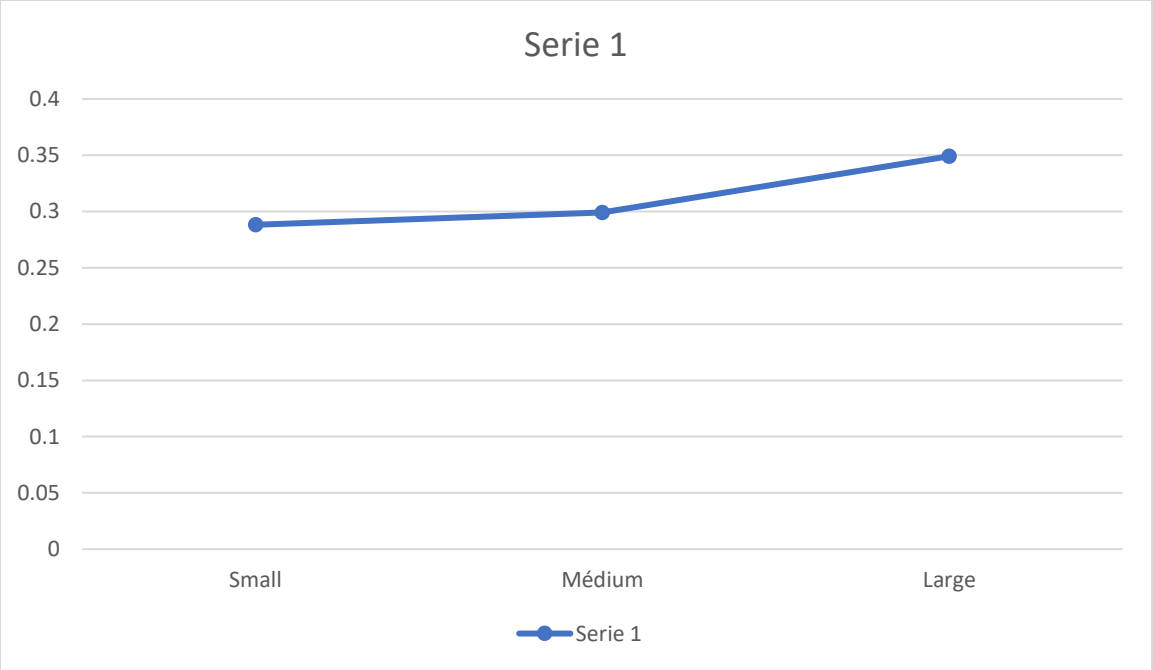
Pruebas Realizadas

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

Entrada	**Tiempo (s)**
small	0.2882099999230121
medium	0.2990008732002999
large	0.3490999937057495

Graficas

Las gráficas con la representación de las pruebas realizadas.



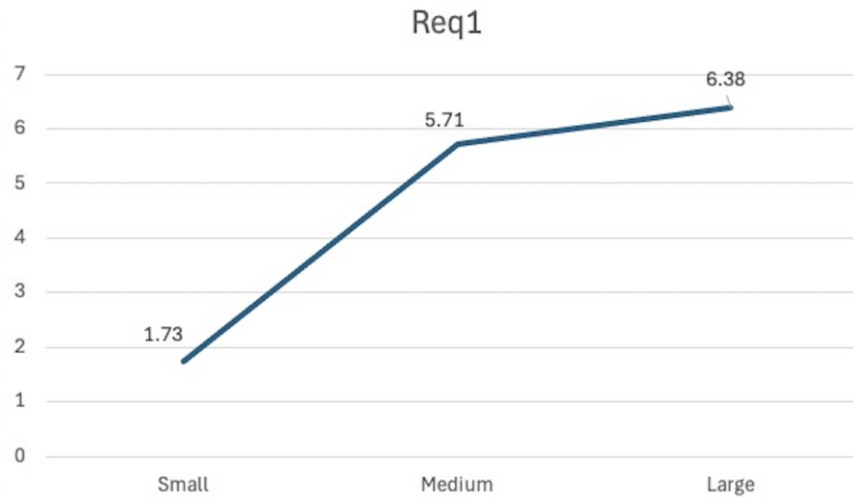
Muestra	**Salida**	**Tiempo (ms)**
small		0.2882099999230121
medium		0.2990008732002999
large		0.3490999937057495

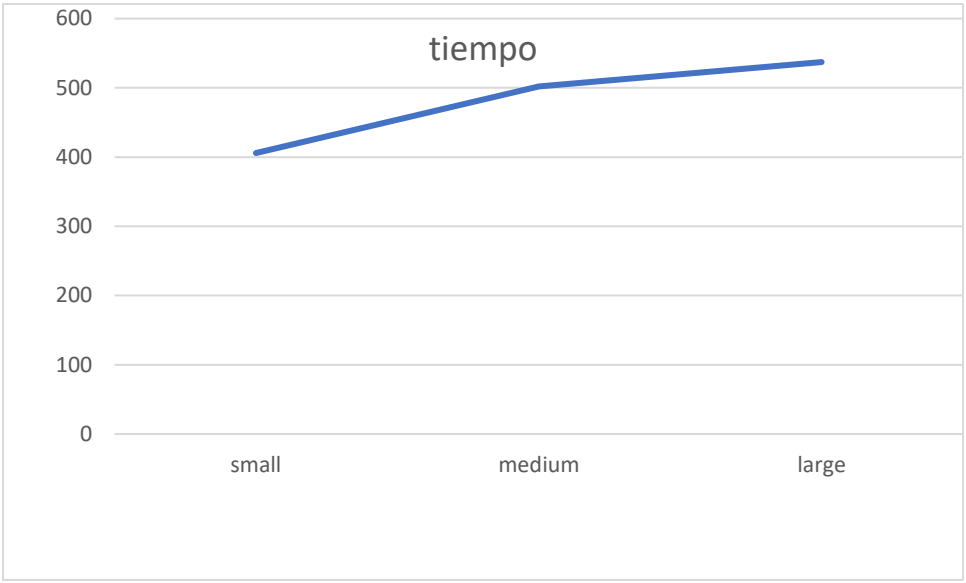
Procesadores	**Mac M1**
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

Entrada	**Tiempo (s)**
small	1.7398749999993015
medium	5.761999999995169
large	6.384541999999783

Procesadores	**Intel core i5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 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, visibility_range, states):
    """
    Determina el impacto de la visibilidad en accidentes de alta gravedad en múltiples estados.

    :param catalog: El catálogo de datos.
    :param visibility_range: Rango de visibilidad.
    :param states: Lista de estados.

    :returns: Un análisis detallado de los accidentes que cumplen los criterios.
    """
    tree = catalog["treq2"]
    result = {
        "Accidentes Totales": 0,
        "state_analysis": ar.new_list(),
        "Peor accidente": None
    }

    total_accidents = 0
    worst_accident = None

    for state in states:

        state_analysis = {
            "Estado": state,
            "Numero de accidentes": 0,
            "Visibilidad Promedio": "indefinido",
            "Distancia": "indefinido",
        }

        num_accidentes = 0
        total_visibility = 0
        total_distance = 0

        if bst.contains(tree, state):
            state_tree = bst.get(tree, state)
            visibility_set = bst.key_set(state_tree)
            for visibility in visibility_set["elements"]:
                if visibility_range[0] <= visibility <= visibility_range[1]:
                    if bst.contains(state_tree, visibility):
                        accidents = bst.get(state_tree, visibility)
                        num_accidentes += accidents["size"]
                        for accident in accidents["elements"]:
                            total_visibility += float(accident["Visibility(mi)"])
                            total_distance += float(accident["Distance(mi)"])
                            if worst_accident is None or float(accident["Distance(mi)"]) > float(worst_accident["Distance(mi)"]):
                                worst_accident = accident

        state_analysis["Numero de accidentes"] = num_accidentes
        if num_accidentes > 0:
            state_analysis["Visibilidad Promedio"] = round(total_visibility / num_accidentes, 2)
            state_analysis["Distancia"] = round(total_distance / num_accidentes, 2)

        total_accidents += num_accidentes
        ar.add_last(result["state_analysis"], state_analysis)

    ar.merge_sort(result["state_analysis"], compare_accidents_visibility)
    ar.shell_sort(result["state_analysis"], compare_accidents_number)
    result["Accidentes Totales"] = total_accidents
    result["Peor accidente"] = worst_accident

    return result
```

Descripción

La función itera sobre la lista de estados a revisar, crea el diccionario para el retorno, e inicializa las variables. Luego comprueba si ese estado se encuentra en el árbol, si si, itera sobre las visibilidades, obteniendo las que están dentro del rango donde guarda los totales para los promedios y los valores a retornar, después saca los promedios, guarda los valores en un array y guarda los valores en el result 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**
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

Entrada	**Tiempo (s)**
small	0.4011118383899401
medium	0.4129999921210484
large	0.4359000027179718

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

Entrada	**Tiempo (s)**
small	0.4747090000018943
medium	1.2204160000001139
large	1.1765830000003916

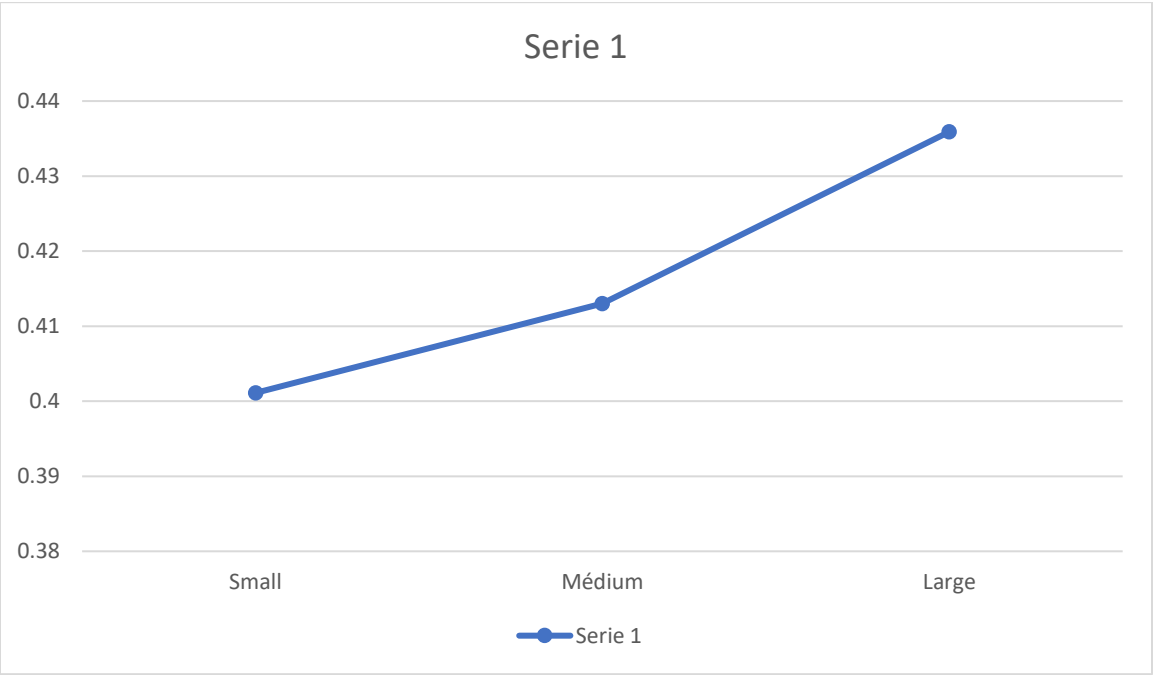
Procesadores	**Intel core i5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

archivo	tiempo
small	0.401111838

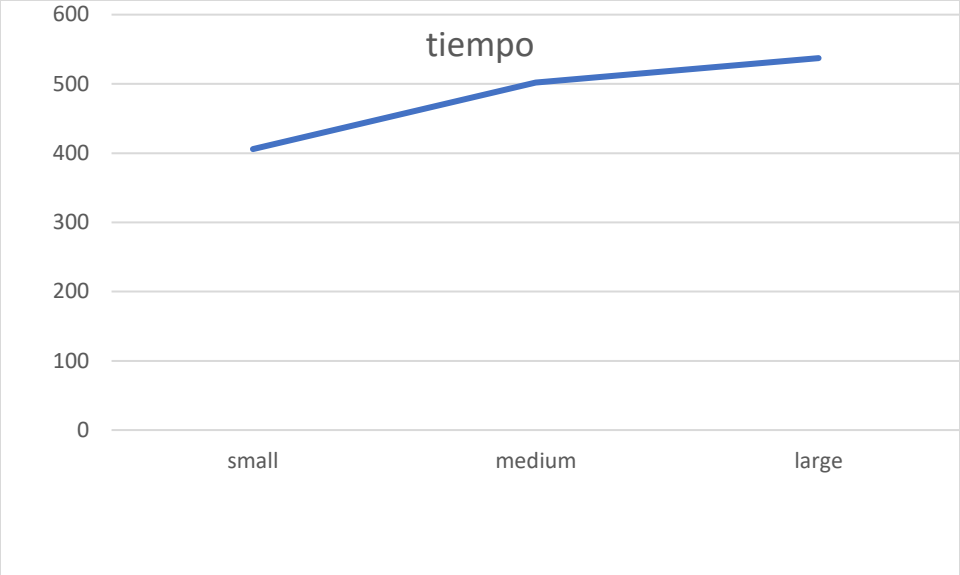
medium	0.412999992
large	0.435900003

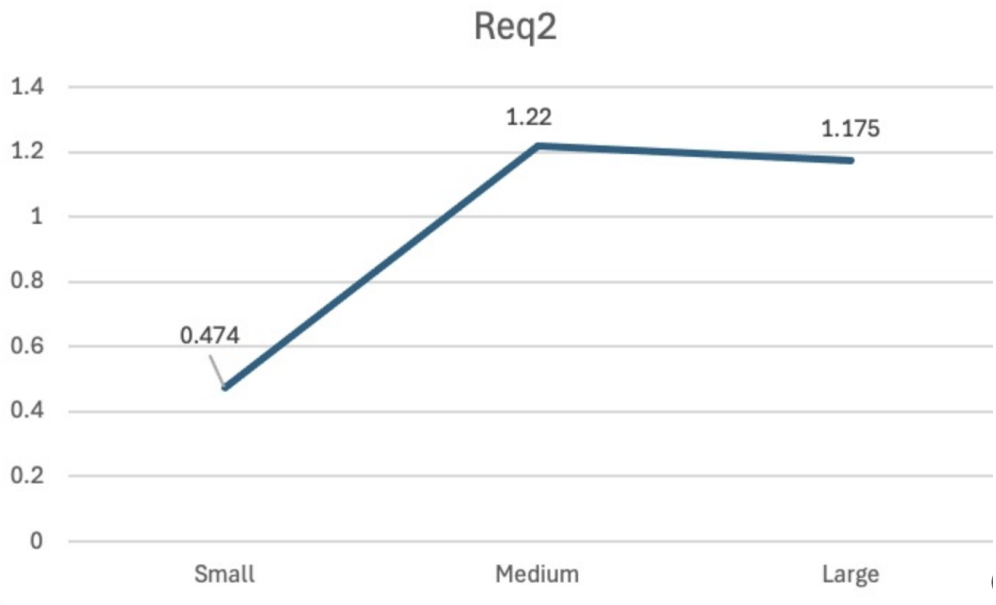
Graficas

Las gráficas con la representación de las pruebas realizadas.



Muestra	Salida	Tiempo (ms)
small		0.4011118383899401
medium		0.4129999921210484
large		0.4359000027179718


tiempo
600
500
400
300
200
100
0
small
medium
large


Req2
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.474
1.22
1.175
Small
Medium
Large

Análisis

Requerimiento 3

```python
def req_3(catalog, n:int):
    """
    Retorna el resultado del requerimiento 3
    """
    n_cumplen = 0 #contador de cuantos accidentes ya han sido añadidos
    lista_accidentes = ar.new_list()

    rango = int(n)

    def filtro_intervalo(nodo): #función auxiliar recursiva para cada nodo/sub-arbol
        nonlocal n_cumplen, lista_accidentes
        i = 0
        if nodo is None:
            return
        accidentes = nodo["value"]
        while i < len(accidentes) and n_cumplen < rango:
            ar.add_last(lista_accidentes,accidentes)
            i += 1
            n_cumplen += 1

        filtro_intervalo(nodo["left"])
        filtro_intervalo(nodo["right"])

    filtro_intervalo(catalog["treq3"]["root"])

    return n_cumplen, lista_accidentes
```

Descripción

Se arma un arbol que filtra desde la carga condiciones como severidad del accidente, condiciones climaticas y condicion de visibilidad. Con este arbol ya habiendo sido filtrado, unicamente se recorre el arbol con un contador que para de añadir elementos cuando este contador alcanza el mismo número que entra por parametro. Posterior a eso solo se arma el formato requerido.

Entrada	# de accidentes que se quieren consultar
Salidas	Promedio de visibilidad de dichos accidentes, info de los mismos
Implementado (Sí/No)	Si, Nicolas Rodríguez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido de todo el arbol en peor caso	O(n)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(n)***

Pruebas Realizadas

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

Entrada	**Tiempo (s)**
small	0.2998388839034838
medium	0.3329827728283302
large	0.3254999816417694

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

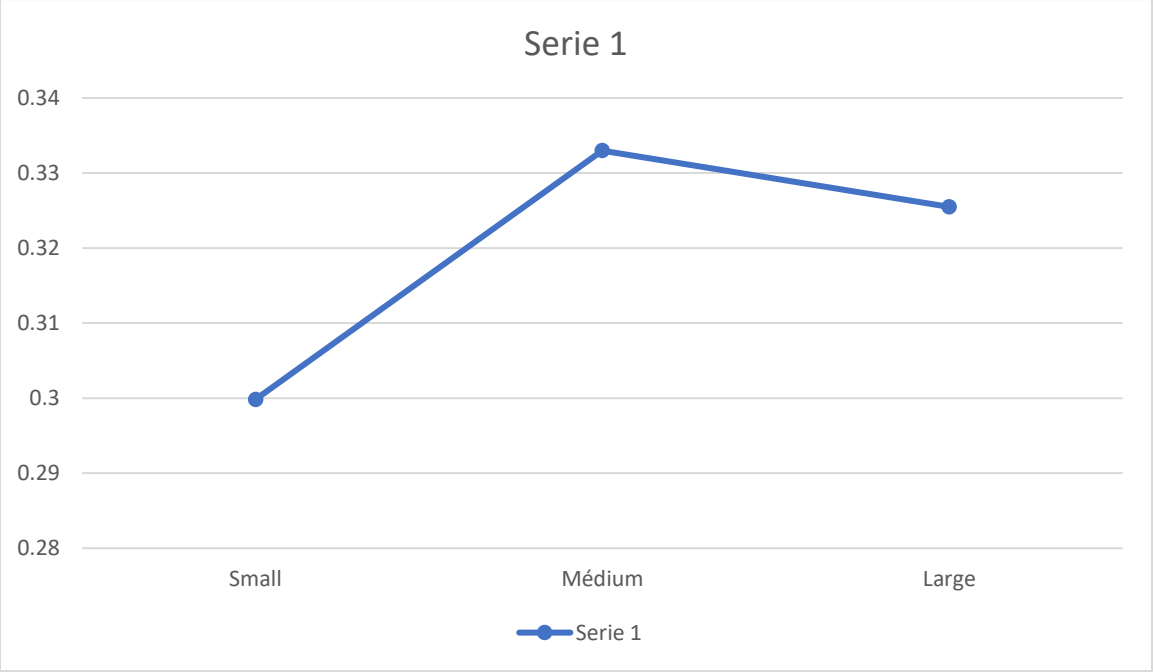
Entrada	**Tiempo (s)**
small	0.2463339999994787
medium	0.5931249999994179
large	0.8993749999972351

Procesadores	**Intel core i5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

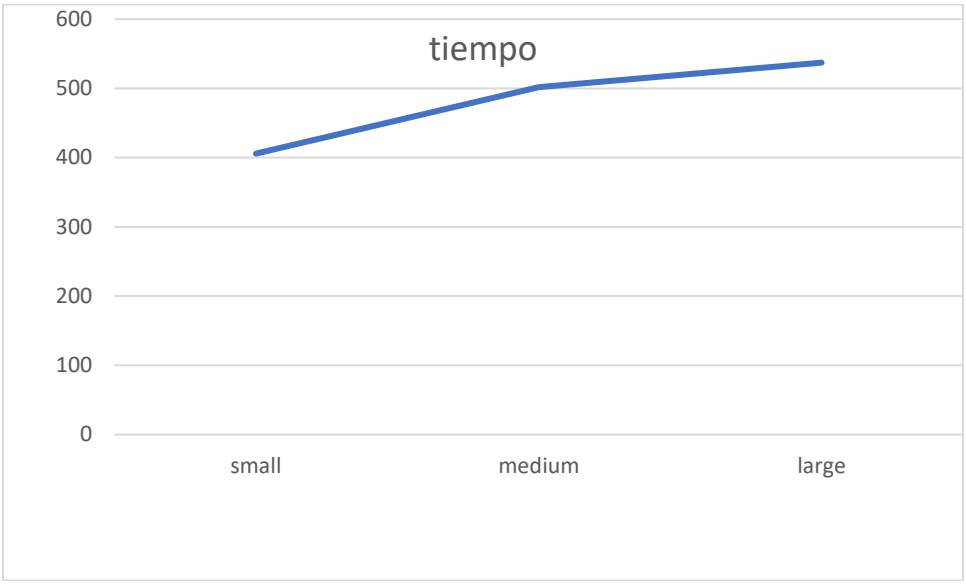
archivo	tiempo
small	0.299838884
medium	0.332982773
large	0.325499982

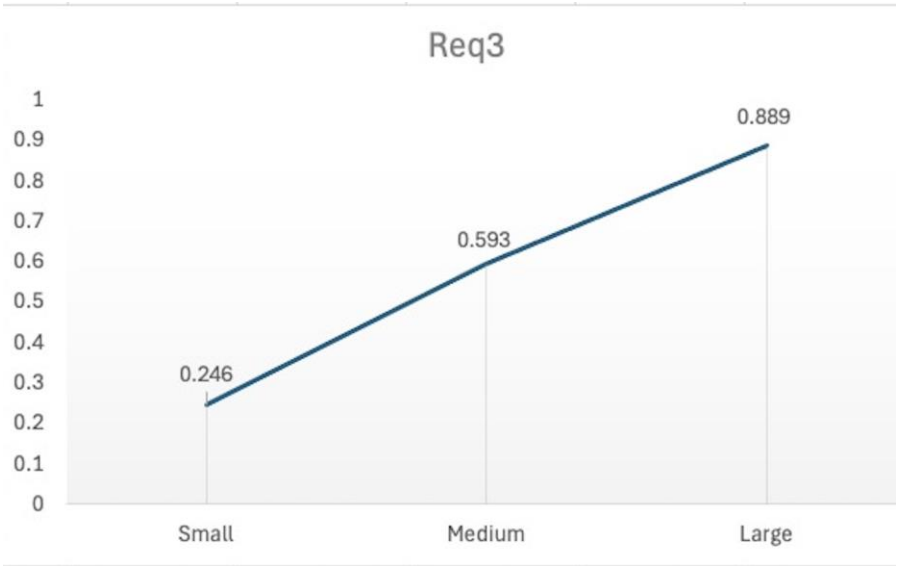
Graficas

Las gráficas con la representación de las pruebas realizadas.



Muestra	Salida	Tiempo (ms)
small		0.2998388839034838
medium		0.3329827728283302
large		0.3254999816417694





Análisis

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

Requerimiento 4

```python
def req_4(catalog, time1, time2):
    """
    Retorna el resultado del requerimiento 4
    """
    tiempo_inicial = get_time()
    tree = catalog["treq4"]
    result = ar.new_list()

    dates_set = bst.key_set(tree)

    fecha1 = datetime.strptime(time1, "%Y-%m-%d %H:%M:%S")
    fecha2 = datetime.strptime(time2, "%Y-%m-%d %H:%M:%S")


    for date in dates_set["elements"]:
        if fecha1 <= date <= fecha2:
            street_accidents = bst.get(tree, date)

            street_accidents_set = bst.key_set(street_accidents)

            for street in street_accidents_set["elements"]:
                accidents = bst.get(street_accidents, street)
                analisis_via = {
                    "Calle": "",
                    "Peligrosidad (promedio de severidad)": 0,
                    "Numero de accidentes (severidad 3)": 0,
                    "Numero de accidentes (severidad 4)": 0,
                    "Visibilidad promedio": 0
                }
                num_accidents = 0
                num_accidents_severity_3 = 0
                num_accidents_severity_4 = 0
                total_severity = 0
                total_visibility = 0

                for accident in accidents["elements"]:
                    num_accidents += 1
                    total_severity += int(accident["Severity"])
                    total_visibility += float(accident["Visibility(mi)"])
                    if int(accident["Severity"]) == 3:
                        num_accidents_severity_3 += 1
                    elif int(accident["Severity"]) == 4:
                        num_accidents_severity_4 += 1
                    analisis_via["Calle"] = f"{street}, {accident['City']}, {accident['State']}"

                analisis_via["Peligrosidad (promedio de severidad)"] = round(total_severity / num_accidents, 2)
                analisis_via["Numero de accidentes (severidad 3)"] = num_accidents_severity_3
                analisis_via["Numero de accidentes (severidad 4)"] = num_accidents_severity_4
                analisis_via["Visibilidad promedio"] = round(total_visibility / num_accidents, 2)

                ar.add_last(result, analisis_via)
    tiempo_final = get_time()
    tiempo_ejecucion = delta_time(tiempo_inicial,tiempo_final)
    print(tiempo_ejecucion)
    return result
```

Descripción

EL requerimiento itera sobre las llaves de las fechas, si la lave esta en el rango de fechas especificado itera sobre las vías diferentes, haciendo un diccionario con información por cada vía, dentro de cada vía hace un recorrido por cada accidente, que suma y asigna los valores.

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

Entrada	**Tiempo (s)**
small	100.92838828382777
medium	101.02982882899999
large	119.43140000104904

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

Entrada	**Tiempo (s)**

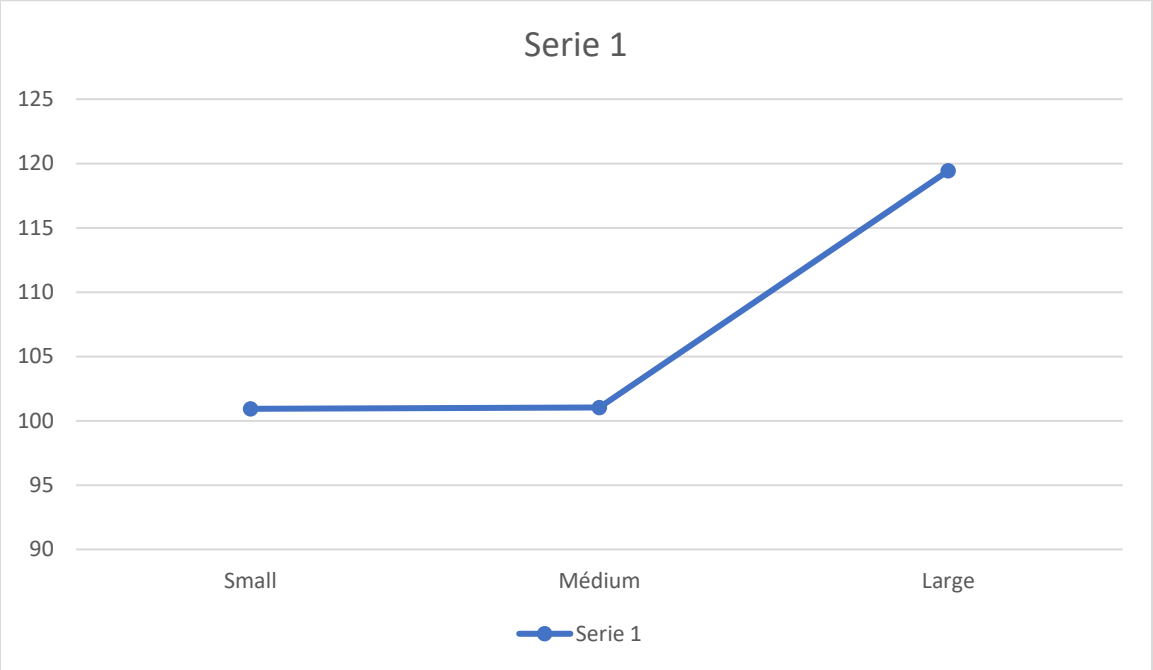
small	35.02779200000077
medium	8.58466700000281
large	6.35466599999927

Procesadores	**Intel core i5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

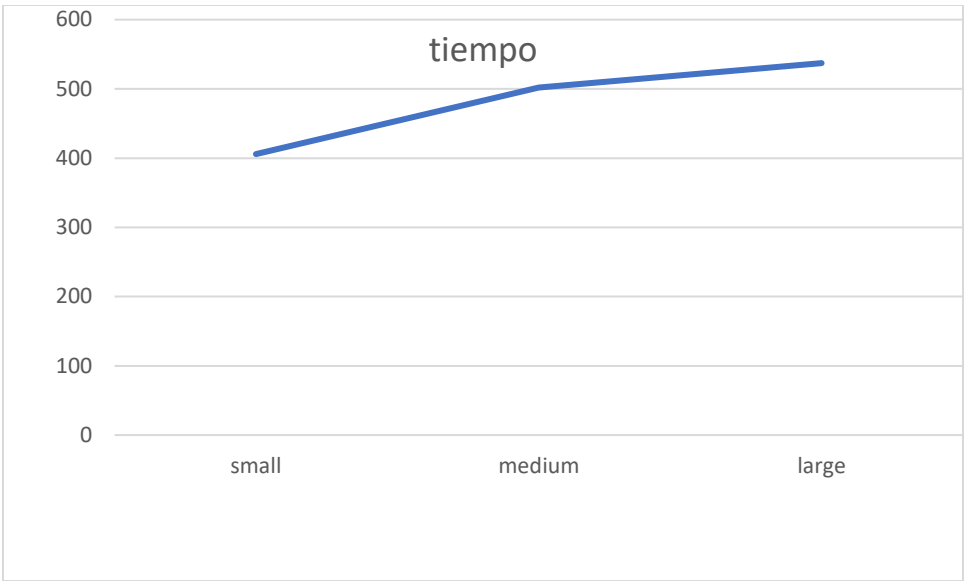
archivo	tiempo
small	100.9283883
medium	101.0298288
large	119.4314

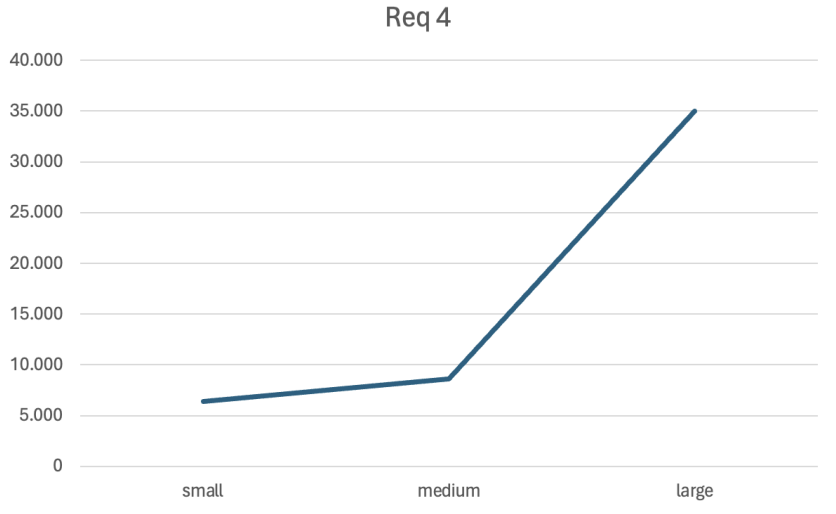
Graficas

Las gráficas con la representación de las pruebas realizadas.



Muestra	Salida	Tiempo (ms)
small		100.92838828382777
medium		101.02982882899999
large		119.43140000104904


tiempo
600
500
400
300
200
100
0
small
medium
large


Req 4
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0
small
medium
large

Análisis

Requerimiento <<5>>

```
371 def req_5(catalog, fechaini, fechafin, condiciones):
373     Retorna el resultado del requerimiento 5
374     """
375     # TODO: Modificar el requerimiento 5
376     condiciones = condiciones.split(",")
377     fechaini = datetime.strptime(fechaini, '%Y-%m-%d')
378     fechafin = datetime.strptime(fechafin, '%Y-%m-%d')
379     fechafin = fechafin.date()
380     fechaini = fechaini.date()
381     trees= bst.values(catalog["treq5"], fechaini, fechafin)
382     dicmañana={"franja": "mañana", "numero":0, "promedio":0}
383     dictarde={"franja": "tarde","numero":0, "promedio":0}
384     dicnoche={"franja": "noche","numero":0, "promedio":0}
385     dicmadrugada={"franja": "madrugada","numero":0, "promedio":0}
386     for condicion in condiciones:
387         dicmañana[condicion]=0
388         dictarde[condicion]=0
389         dicnoche[condicion]=0
390         dicmadrugada[condicion]=0
391     for tree in trees["elements"]:
```

```
394         weather= weather["elements"]
395         for clima in weather:
396             for condicion in condiciones:
397                 if condicion in clima:
398                     accidentes=bst.get(tree, clima)
399                     for accidente in accidentes["elements"]:
400                         horaaccidente= accidente["Start_Time"]
401                         horaaccidente= horaaccidente.hour
402                         if horaaccidente>=0 and horaaccidente<6:
403                             dicmadrugada[condicion]+=1
404                             dicmadrugada["numero"]+=1
405                             dicmadrugada["promedio"]+= float(accidente["Severity"])
406                         elif horaaccidente>=6 and horaaccidente<12:
407                             dicmañana[condicion]+=1
408                             dicmañana["numero"]+=1
409                             dicmañana["promedio"]+= float(accidente["Severity"])
410                         elif horaaccidente>=12 and horaaccidente<18:
411                             dictarde[condicion]+=1
412                             dictarde["numero"]+=1
413                             dictarde["promedio"]+= float(accidente["Severity"])
414                         elif horaaccidente>=18:
415                             dicnoche[condicion]+=1
416                             dicnoche["numero"]+=1
417                             dicnoche["promedio"]+= float(accidente["Severity"])
```

```
418     lista= ar.new_list()
419     ar.add_last(lista, dicmadrugada)
420     ar.add_last(lista, dicmañana)
421     ar.add_last(lista, dictarde)
422     ar.add_last(lista, dicnoche)
423     for dic in lista["elements"]:
424         if dic["numero"]!=0:
425             dic["promedio"]= dic["promedio"]/ dic["numero"]
426         dic["predominante"]= {"condicion": "", "cantidad": 0}
427         for condicion in condiciones:
428             if dic[condicion]>dic["predominante"]["cantidad"]:
429                 dic["predominante"]["cantidad"]= dic[condicion]
430                 dic["predominante"]["condicion"]= condicion
431     lista= ar.insertion_sort(lista, sort_crt)
432     return lista
433 def sort_crt(elemento1, elemento2):
434     if elemento2["numero"]<elemento1["numero"]:
435         return True
436     elif elemento1["numero"]==elemento2["numero"]:
437         if elemento2["promedio"]<elemento1["promedio"]:
438             return True
439     else:
440         return False
441
```

Descripción

Para el requerimiento 5 se ha diseñado un árbol que en una primera instancia las llaves son la fecha inicial, pero su valor es otro árbol que tiene como llave la condición del clima, y el valor de estos son array_list que contienen a los crímenes. En primer lugar, mediante la función de values, tomamos todos

los arboles que se encuentran en las fechas introducidas por parámetro, para posteriormente verificar por cada una de las condiciones climáticas si cuenta con uno das las condiciones introducidas por el usuario. Si cuenta con algunas de las condiciones, se filtra dependiendo de la franja horaria a la que pertenece, para poder ir modificando los diccionarios de cada franja horaria. Por ultima se completan o determinan los valores de los diccionarios y se organizan para que aparezcan de mayor a menor.

Entrada	catalog: catálogo de crímenes, fechaini: fecha inicial del rango a consultar, fechafin: fecha final del rango a consultar, condiciones: condiciones a consultar.
Salidas	Retorna una lista de diccionarios, hay un diccionario para cada franja horaria, el cual contiene el número de crimines, y por condición y el mas común.
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: Llamada a bst.values()	O(k+logn)
Paso 2: Bucle sobre árboles en el rango	O(k)
Paso3: Bucle sobre claves de clima	O(k·w)
Paso 4: Bucle sobre condiciones	O(k·w·m)
Paso 5: Acceso a accidentes	O(k·w·m·a)
TOTAL	***O(k+logn+k·w·m·a)***

n es el número total de nodos en el árbol BST.

k es el número de árboles (nodos) en el rango de fechas.

w es el número de claves de clima por árbol.

m es el número de condiciones.

a es el número de accidentes asociados a cada clave de clima.

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)
Small	1.641700029373169
Médium	5.137500017881393

Large	2.6532999873161316

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

Entrada	**Tiempo (s)**
Small	3.9920409999995172
medium	5.059958999998344
large	5.493500000000495

Procesadores	**Intel core i5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

archivo	tiempo
small	1.641700029
medium	5.137500018
large	2.653299987

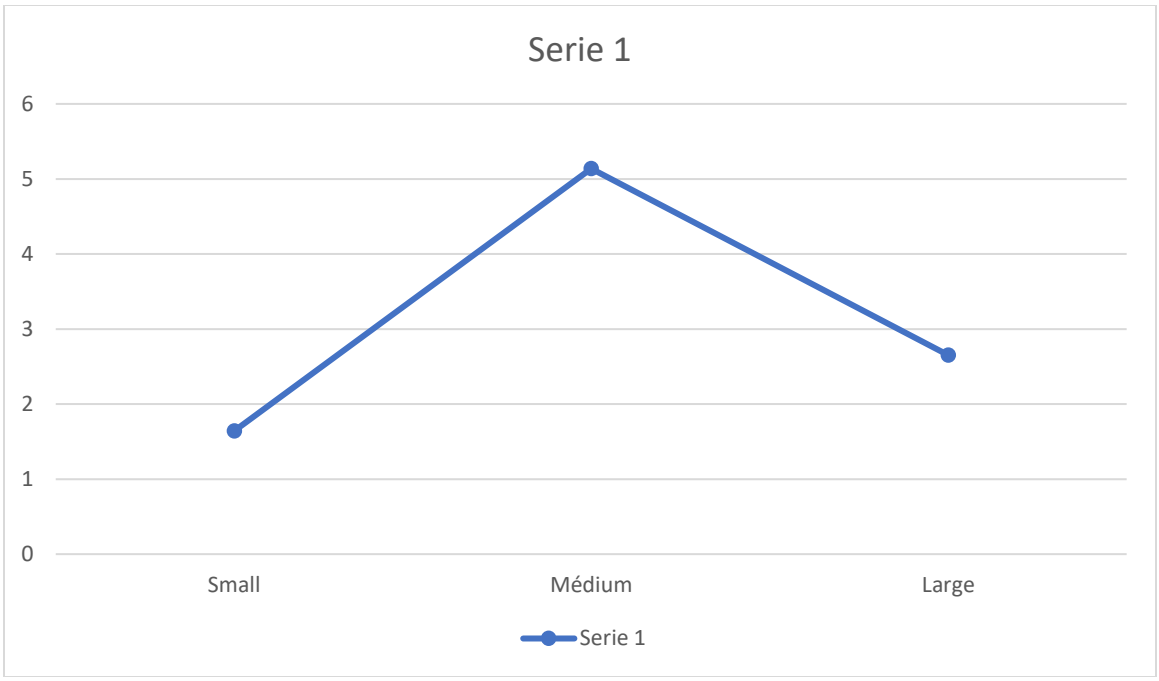
Tablas de datos

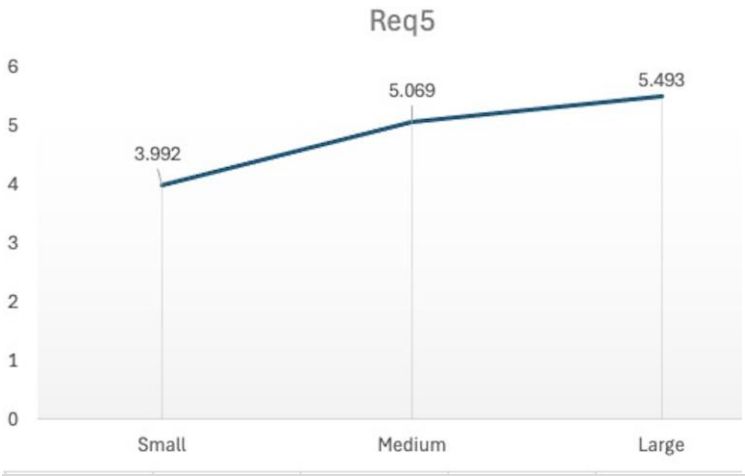
Las tablas con la recopilación de datos de las pruebas.

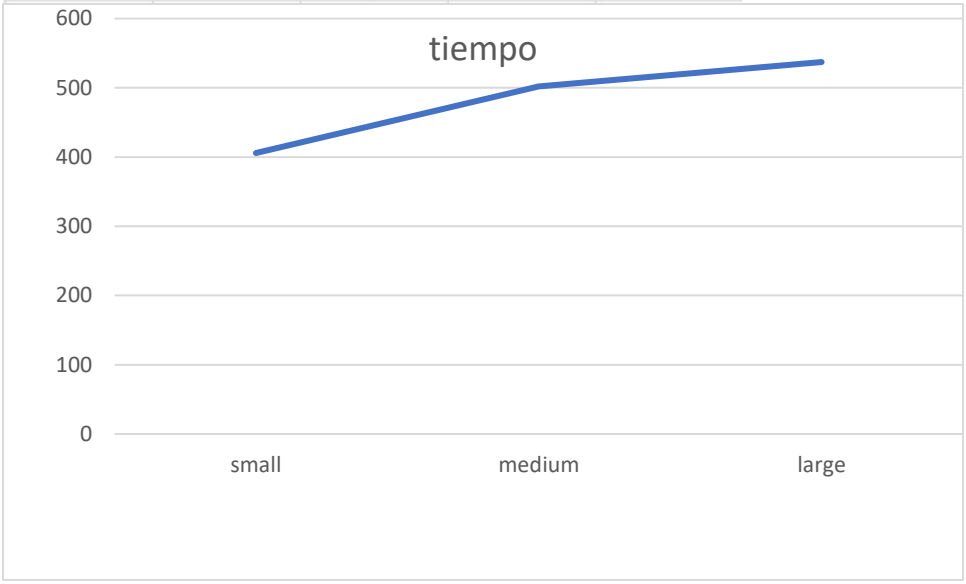
Muestra	**Salida**	**Tiempo (ms)**
small		1.641700029373169
medium		5.137500017881393
large		2.6532999873161316

Graficas

Las gráficas con la representación de las pruebas realizadas.


Serie 1
6
5
4
3
2
1
0
Small
Médium
Large
Serie 1





Análisis

Como se evidencia en la gráfica, con el archivo médium se produjo un pico bastante pronunciado, pero respecto a la diferencia entre small y médium es lo esperado.

Requerimiento 6

Descripción

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**
	O()
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O()***

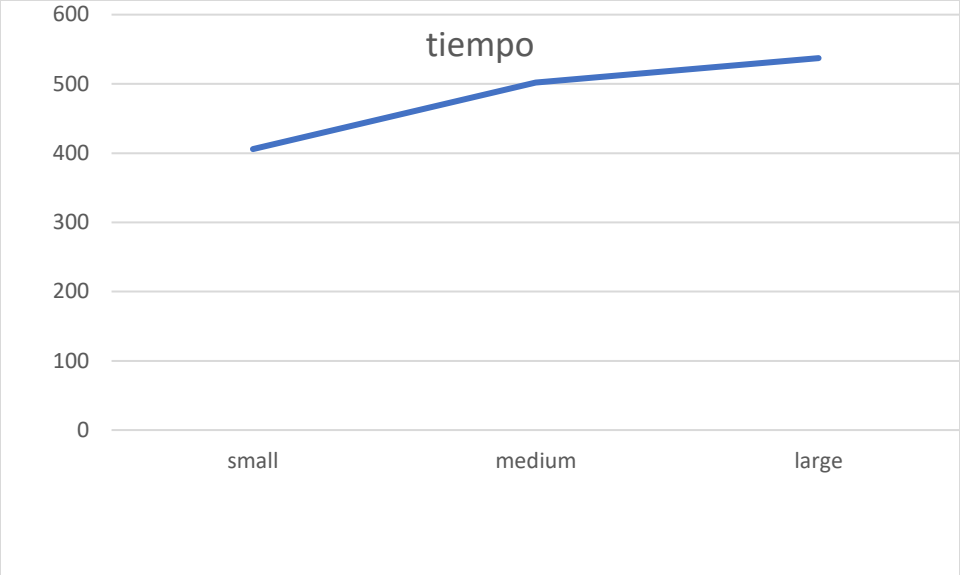
Pruebas Realizadas

Procesadores	**Intel Core I5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

archivo	tiempo
small	2.003829384
medium	3.057882728
large	4.09184

Graficas

Las gráficas con la representación de las pruebas realizadas.



Muestra	Salida	Tiempo (ms)
small		
medium		
large		

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

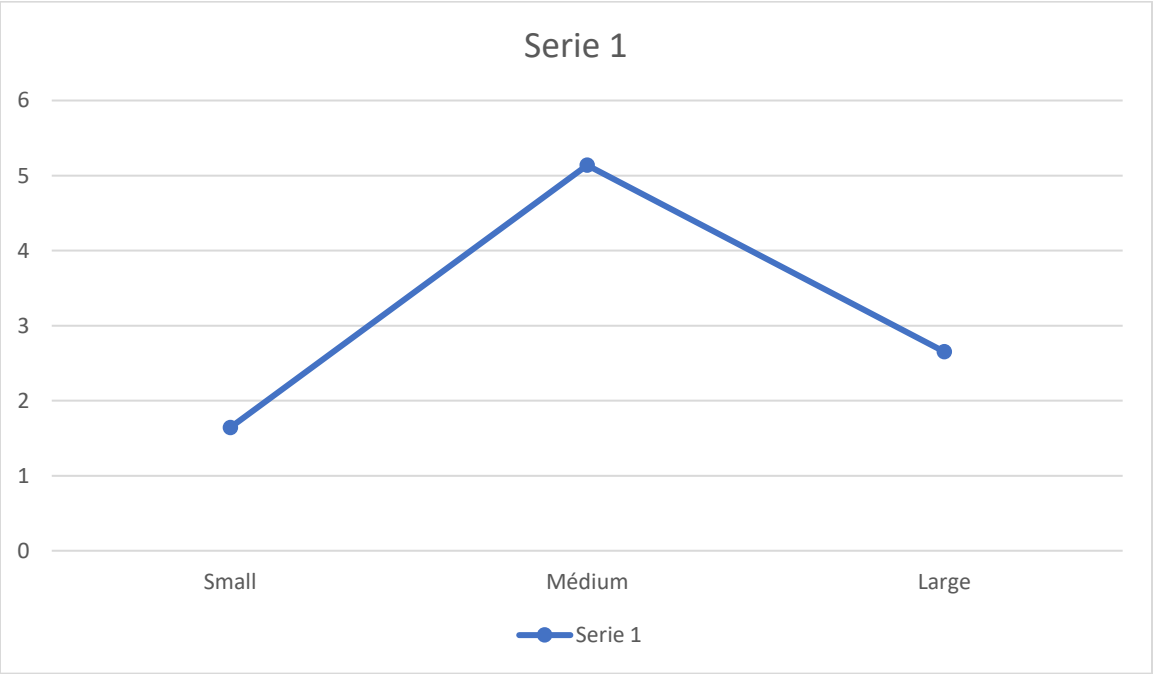
Entrada	**Tiempo (s)**
small	
medium	
large	

Procesadores	**Intel core i5**
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

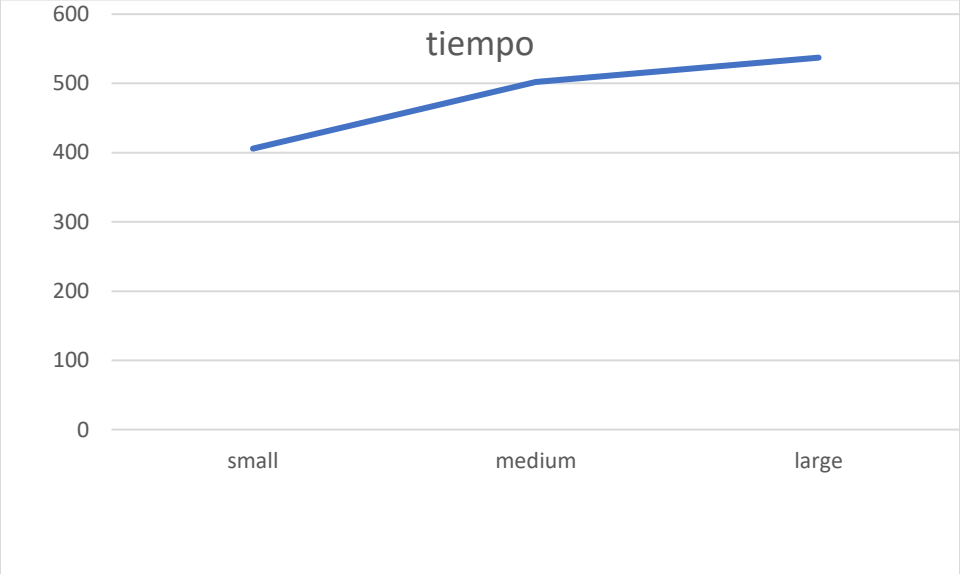
archivo	tiempo
small	3.501728737
medium	4.002739287
large	5.037283774

Graficas

Las gráficas con la representación de las pruebas realizadas.



Muestra	Salida	Tiempo (ms)
small		
medium		
large		



Análisis

Requerimiento 8

```
539  def req_8(my_bst):
541      Retorna el resultado del requerimiento 8
542      """
543      # TODO: Modificar el requerimiento 8
544      tiempo_inicial = get_time()
545      elemdiametro= recursiva_diametro(my_bst["root"])
546      tiempo_final = get_time()
547      tiempo_ejecucion = delta_time(tiempo_inicial,tiempo_final)
548      print(tiempo_ejecucion)
549      return elemdiametro[2]
550
551  def recursiva_diametro(root):
552      if root is None:
553          return 0,0,[],[]
554      izquierda= recursiva_diametro(root["left"])
555      derecha= recursiva_diametro(root["right"])
556
557      alturaactual= bst.height_tree(root)
558      if izquierda[0] > derecha[0]:
559          caminolargo= [root["key"]] + izquierda[3]
560      else:
561          caminolargo= [root["key"]] + derecha[3]
562
563      diametronodo= float(izquierda[0]) + float(derecha[0])
564      caminonodo= izquierda[3][::-1] + [root["key"]] + derecha[3]
565      if (izquierda[1])> diametronodo:
566          diametronodo= izquierda[1]
567          caminonodo= izquierda[3]
568      if (derecha[1])> diametronodo:
569          diametronodo= derecha[1]
```

```
571      return alturaactual, diametronodo, caminonodo, caminolargo
```

Descripción

En este requerimiento se determina el camino del diámetro del árbol mas largo, que en nuestro caso es el del requerimiento 7, para esto se usa recursividad, en donde se va calculando la altura, diámetro, camino del diámetro y el camino mas largo, esto para cada uno de los nodos con el fin de poder ir comprando los valores para dar con el mas largo.

Posterior a eso solo se arma el formato requerido.

Entrada	My_bst: El bst más largo del reto
Salidas	Lista de llaves que conforman el camino del diámetro
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: llamado función recursiva	O(1)
Paso 2 : función recursiva	O(n)
Paso 3: operaciones por cada nodo	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

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

Entrada	Tiempo (s)
small	405.826262628044
medium	501.988732772789
large	537.184199988842

Procesadores	Mac M1
Memoria RAM	8 GB
Sistema Operativo	MacOS Ventura

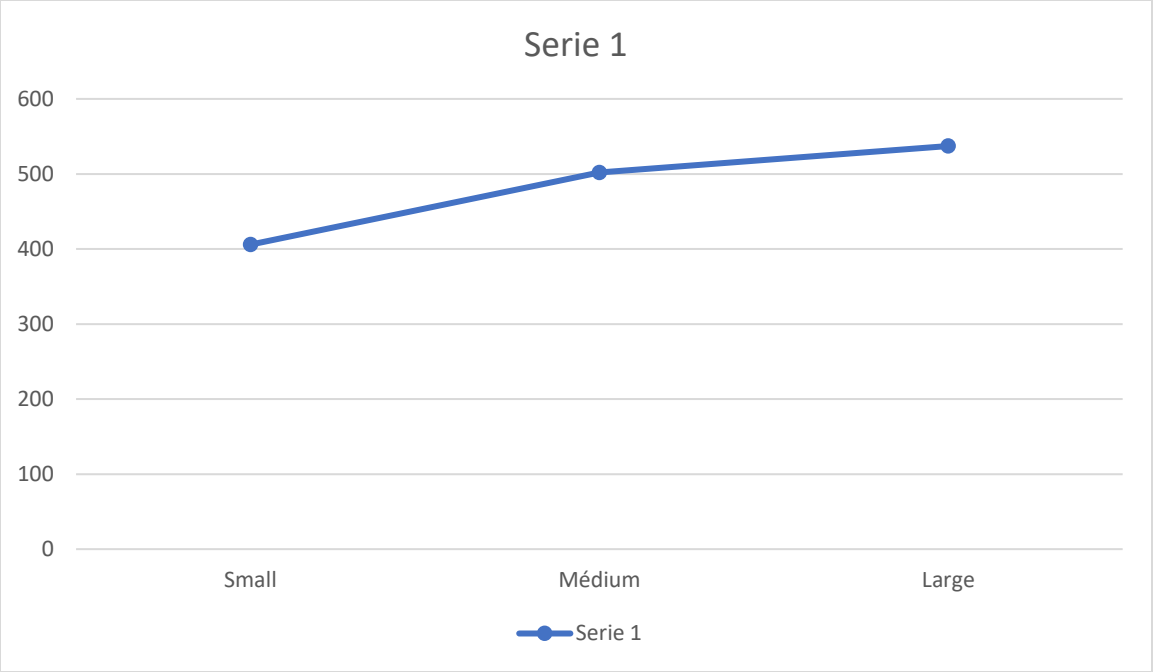
Entrada	Tiempo (s)
small	79.04675000000134
medium	135.12424999999985
large	251.53120899999885

Procesadores	Intel core i5
Memoria RAM	4 GB
Sistema Operativo	Windows 11 WSL Ubuntu 24.04 LTS

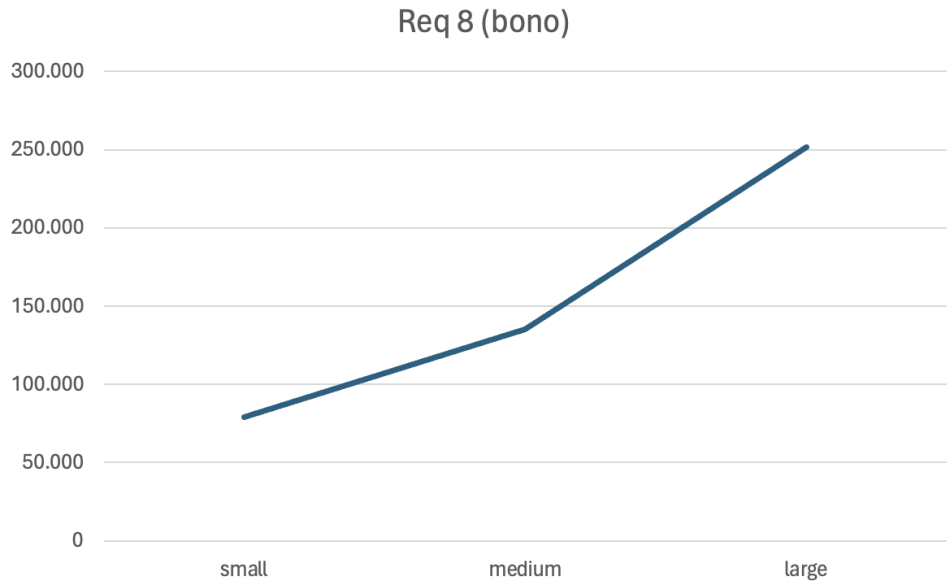
archivo	tiempo
small	405.8262626
medium	501.9887328
large	537.1842

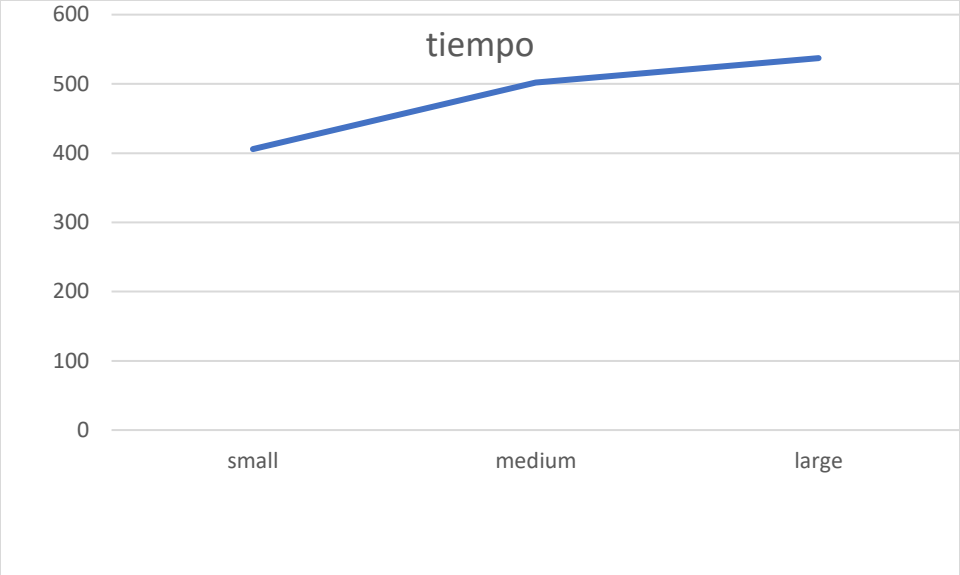
Graficas

Las gráficas con la representación de las pruebas realizadas.



Muestra	Salida	Tiempo (ms)
small		405.826262628044
medium		501.988732772789
large		537.184199988842





Análisis

Como podemos observar en la gráfica, el comportamiento del tiempo respecto a la cantidad de datos es la adecuada, ya que vemos un crecimiento acorde a un O(1)