Análisis de Resultados-Reto3-EDA

David Burgos – 201818326-REQ2

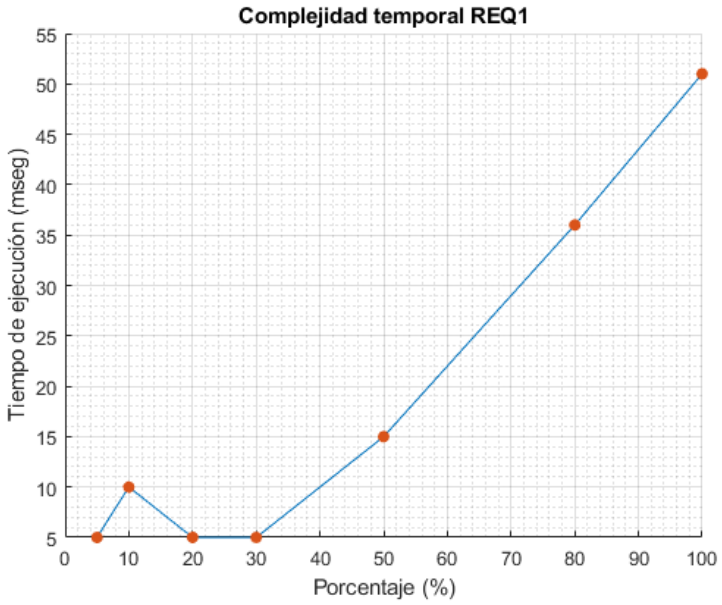
Andrés Mugnier – 201729994-REQ3

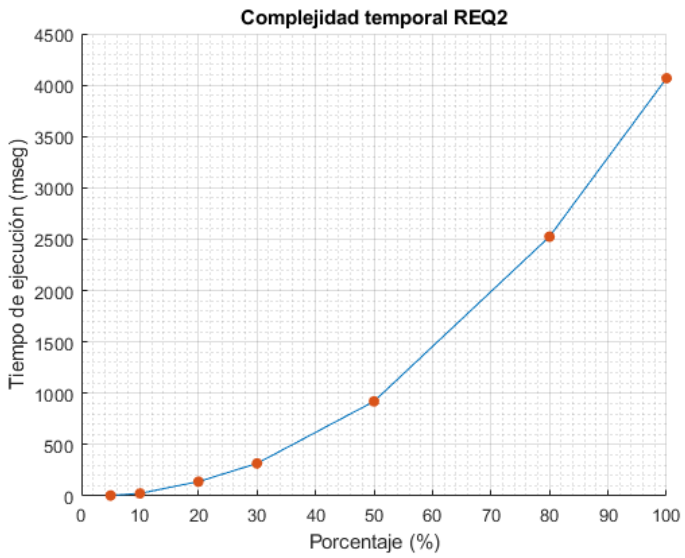
Análisis temporal experimental de los requerimientos:

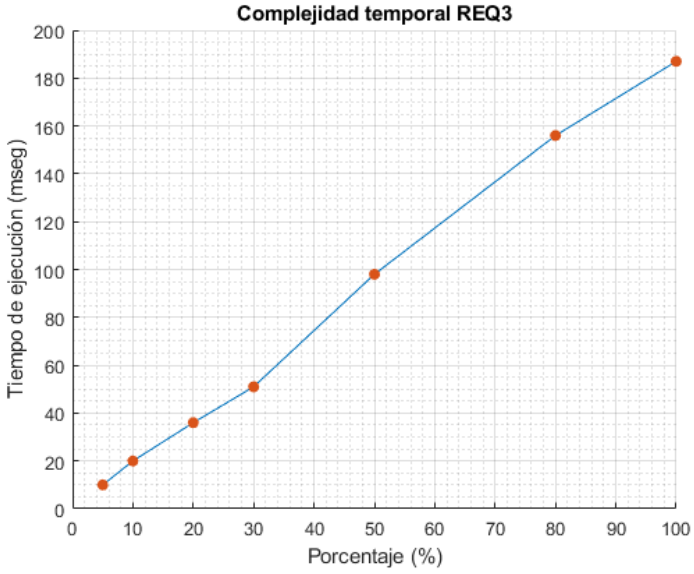
Se realizaron las pruebas de complejidad temporal para cada requerimiento utilizando los siguientes inputs:

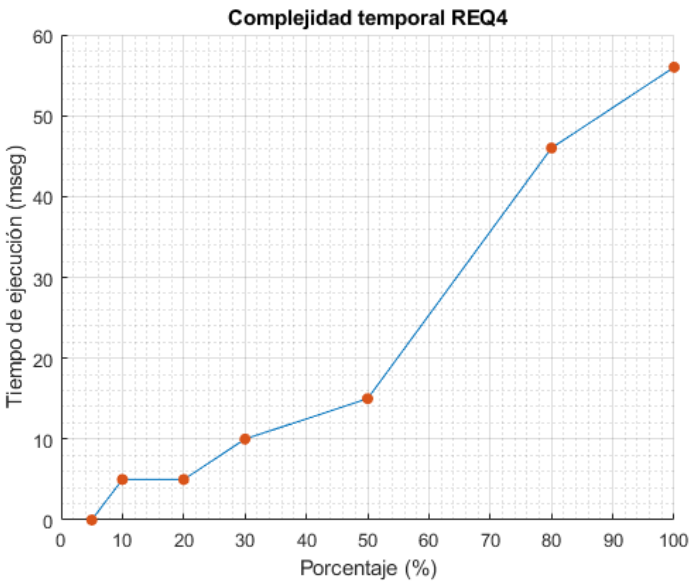
- REQ1: las vegas
- REQ2: 30-150 seg
- REQ3: 20:45-23:15
- REQ4: 1945-08-06, 1984-11-15
- REQ5: longitud -103.00 a -109.05 y una latitud 31.33 a 37.00.

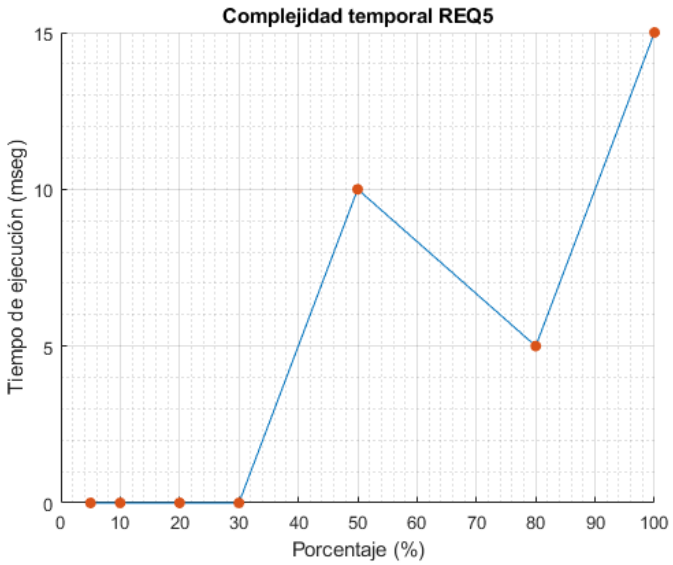
Cada prueba se realizó 3 veces para cada porcentaje de los datos propuestos. Los resultados de estas pruebas se pueden observar en las siguientes imágenes:

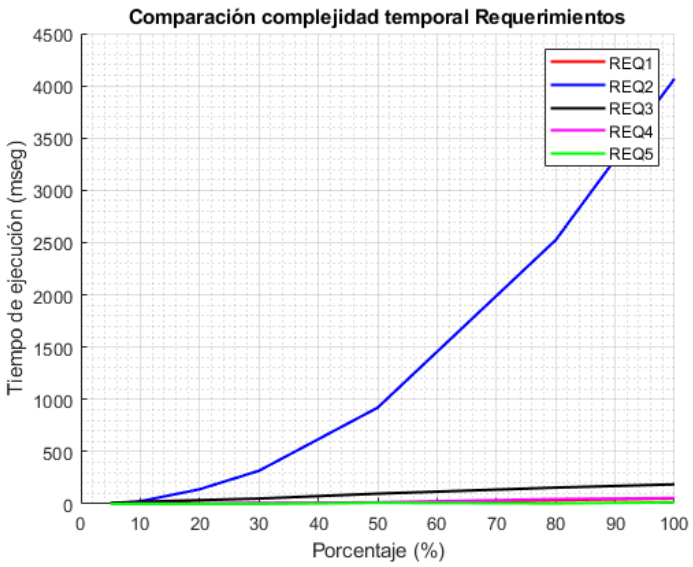


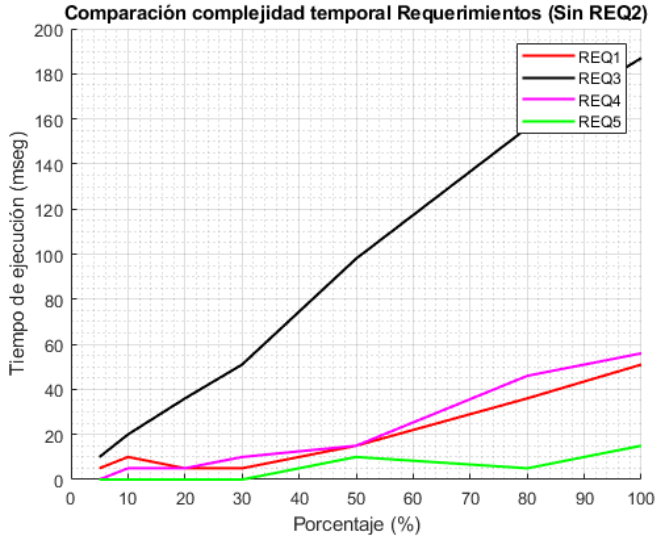

Complejidad temporal REQ2
Tiempo de ejecución (mseg)
Porcentaje (%)
0
500
1000
1500
2000
2500
3000
3500
4000
4500
0
10
20
30
40
50
60
70
80
90
100


Complejidad temporal REQ3
Tiempo de ejecución (mseg)
Porcentaje (%)
0
20
40
60
80
100
120
140
160
180
200
0
10
20
30
40
50
60
70
80
90
100


Complejidad temporal REQ4
Tiempo de ejecución (mseg)
Porcentaje (%)
0
10
20
30
40
50
60
70
80
90
100


Complejidad temporal REQ5
Tiempo de ejecución (mseg)
Porcentaje (%)
0
5
10
15
20
30
40
50
60
70
80
90
100



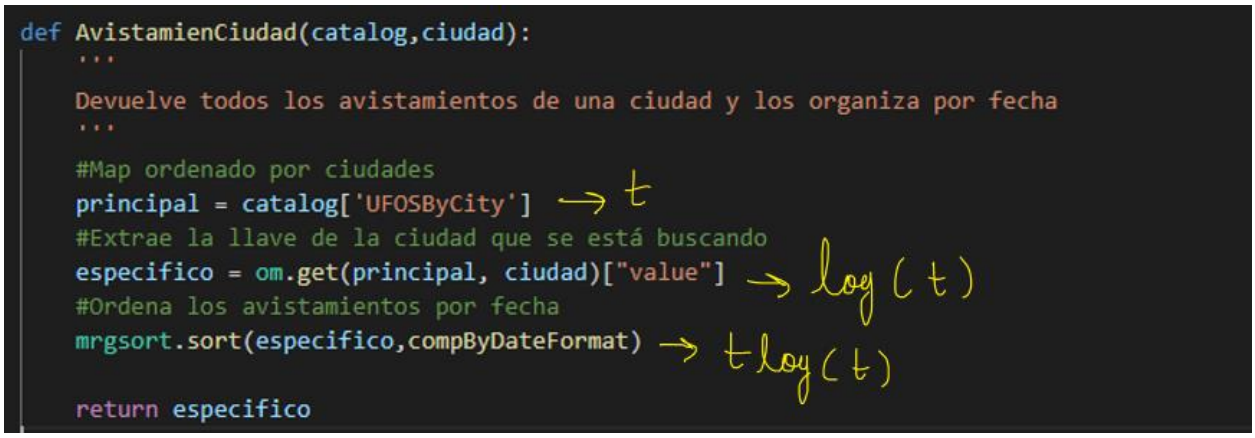


Como se puede ver en las figuras anteriores, en todos los requerimientos se obtuvieron tiempos muy buenos, siendo el tiempo de respuesta más lento (REQ2, 100%) de 0.67 minutos. Esto se debe, en su mayoría a las estructuras de datos utilizadas durante la carga de datos, pues todos los requerimientos tenían prácticamente la misma estructura: busque los elementos en un rango [liminf,limsup] para algún criterio y luego organice estos elementos por otro criterio. Por lo que se creó un RBT para cada requerimiento con las llaves del primer criterio y luego se organizaban con respecto al segundo criterio.

Como todos los arboles utilizados fueron RBT's, el tiempo más largo fue de hecho en la carga de datos, pues los RBT's hicieron muy eficientes las búsquedas (O(log(N)) en todos los requerimientos, pues como ya sabemos los RBT's siempre están balanceados.

Análisis temporal teórico de los requerimientos:

- REQ1:



Para este REQ tenemos un om.get() de un RBT que tiene complejidad log(N) y luego un mergesort que tiene complejidad Nlog(N). Por lo tanto, la complejidad del REQ1 es O(Nlog(N)).

- REQ2:

```python
def AvistamienDireccion(catalog, limInf, limSup):
    '''
    Retorna los avistamientos por duración [liminf,limsup] y los ordena por esa duración.
    '''
    #Toma el map ordenado por duración
    principal = catalog['UFOSBySeconds']
    retorno = lt.newList()

    listaVal = [limInf]
    val1 = limInf
    val2 = limSup
    #Toma la llave máxima del map y su value y cuantos avistamientos tiene
    maximoLL = om.maxKey(principal)
    maximoComp = om.get(principal, maximoLL)["value"]
    maximoCant = lt.size(maximoComp)

    while val1 < val2:
        val1 += 1
        listaVal.append(val1)
    #Recorre las duraciones entre limInf y limSup  y toma sus }
    # avistamientos para añadirlos a retorno (Ya quedan ordenados)
    for x in listaVal:

        if om.contains(principal, float(x)):

            espesifico = om.get(principal, x)["value"]

            for x in range(lt.size(espesifico)):

                elemento = lt.getElement(espesifico, x+1)
                lt.addLast(retorno, elemento)


    return [retorno, [maximoLL, maximoCant]]
```

Para este REQ tenemos que si M=limsup-liminf entonces al recorrer todas las posibles duraciones entre liminf y limsup tendríamos un ciclo de M*N.
Si N=avistamientos en el rango limsup-liminf entonces el segundo for no es cuadrático pues recorre N elementos (el árbol es un árbol de listas entonces para cada nodo tenemos que recorrer una lista, esto parece cuadrático pero no lo es).
Sin embargo, para cada uno de estos ciclos hacemos un om.get() y un om.contains(), por lo que la complejidad de este REQ termina siendo O(N*log(N))

- REQ3:

En este requerimiento tenemos un om.values() para sacar los nodos en el rango requerido, por lo tanto tenemos una complejidad de log(N). Nuevamente, el doble for no es cuadrático pues M=número de avistamientos y N=número de nodos.

- REQ4: Este requerimiento es exactamente el mismo REQ3, por lo tanto tiene complejidad O(log(N)) también.
- REQ5:

Para este requerimiento nuevamente tenemos un om.keys() y un doble for que por las mismas razones anteriores no es cuadrático. Sin embargo, dentro de este for siempre se hace un om.get(), por lo que la complejidad de este req sería de O(Nlog(N)).

- Resumen:

Complejidades temporales	
Requisito	Complejidad
1	O(Nlog(N))
2	O(Nlog(N))
3	O(log(N))
4	O(log(N))
5	O(Nlog(N))