Estructuras de datos

Documento Reto 2

Est1: David Burgos Mendez - d.burgos - 201818326 – Req 4

Est2: Andrés Mugnier Zuluaga- a.mugnier- 201729994 - Req 3

Análisis de complejidad teórica de los requerimientos:

Para todos los requerimientos se mostrará el código que implementa la solución y se marcará en amarillo las veces que se repite cada operación en el peor de los casos para así estimar la complejidad temporal en notación big O.

- Requerimiento 1:

```
def artistasCronologico(lista, inicio, final):
    """
    Retorna una lista con los artistas ordenados por epoca
    """

    # Se abre el mapa necesario y se crea la lista de retorno
    artistas = lista["artists"] -> t
    llaves = mp.keySet(artistas) -> t
    retorno = lt.newList() -> t


    for x in range(lt.size(llaves)): (lineal)

        #por cada artista se revisa si su fecha de nacimiento esta dentro de los parametros
        grupo = mp.get(artistas, lt.getElement(llaves, x))["value"] -> N.t
        edad = int(grupo["begindate"]) -> N.t


        if edad != 0 and edad != None and edad >= inicio and edad <= final: -> N.t


            #se saca la info del artista y se añade a la lista de retorno
            nombre = grupo["name"]
            muerte = int(grupo["enddate"])
            genero = grupo["gender"]
            nacionalidad = grupo["nationality"]
                                                                    } N.t

            agregar = {"nombre" : nombre, "edad" : edad, "muerte" : muerte, "genero" : genero, "nacionalidad" : nacionalidad} N.t
            lt.addLast(retorno, agregar)

    #se ordena la lista
    mrgsort.sort(retorno, compArtistasByBegindate) -> t log(t)

    return retorno -> t
```

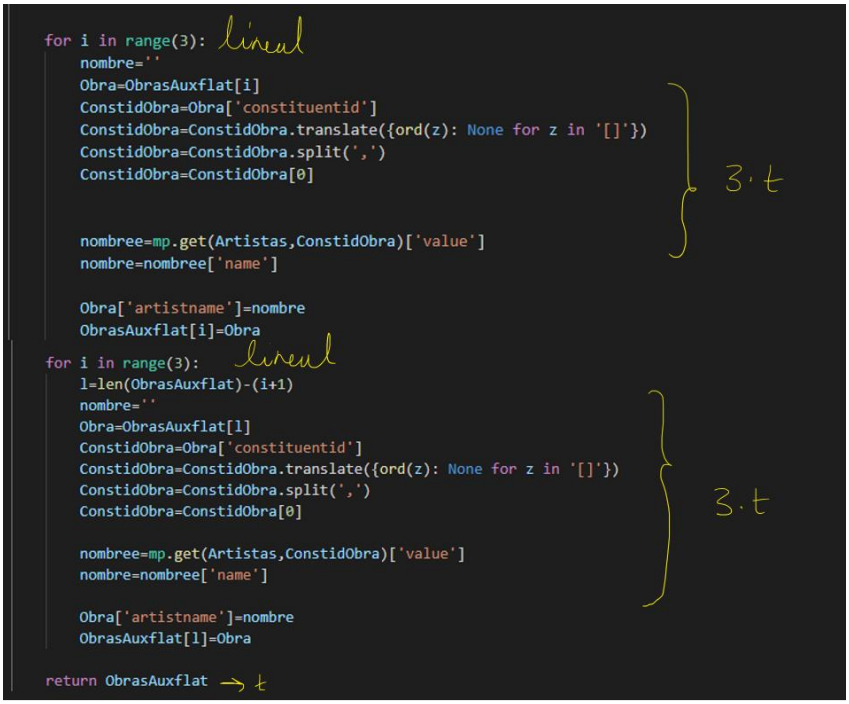
Como se puede ver en la figura anterior, si N es la cantidad total de artistas entonces tenemos un pedazo del código que es lineal pues se recorre todo este arreglo en un for. Luego, los artistas que están en el rango cronológico buscado son añadidos a una nueva lista y luego esta se ordena por fecha de nacimiento utilizando el algoritmo merge sort. Como sabemos, este algoritmo tiene una complejidad de n*log(n) en

todos los casos y como N*log(N) es peor que N, concluimos que la complejidad del primer requerimiento es O(n*log(n)).

- Requerimiento 2:

```
def obrasCronologicoacq(inicio,final,catalog):
    '''
    Retorna las primeras tres y últimas 3 obras adquiridas en el rango de fechas inicio-final
    '''

    #Extrae el map con llaves artistID y values info del artista
    Artistas=catalog['artistID']
    #Pone las fechas iniciales en formate datetime
    YearInit=inicio.year
    MonthInit=inicio.month

    FechaInit=str(YearInit)+'-'+str(MonthInit)
    #Pone las fechas finales en formato datetime
    YearFinal=final.year
    MonthFinal=final.month

    FechaFinal=str(YearFinal)+'-'+str(MonthFinal)

    ObrasAux=[]
    for año in range(YearInit,YearFinal+1):
        for mes in range(MonthInit,MonthFinal+1):
            FechaAux=str(año)+'-'+str(mes)
            if mp.contains(catalog['artworksDateAcqYearMonth'],FechaAux):
                ObrasAuxRango=mp.get(catalog['artworksDateAcqYearMonth'],FechaAux)['value']
                ObrasAux.append(ObrasAuxRango)


    ObrasAuxflat = list(itertools.chain(*ObrasAux))

    ObrasAuxflat2=lt.newList()

    for Obra in ObrasAuxflat[::-1]:
        #print(Obra)
        lt.addLast(ObrasAuxflat2,Obra)
```

```
mrgsort.sort(ObrasAuxflat2, cmpArtworkByDateAcquired)

for i in range(lt.size(ObrasAuxflat2)):
    Obra=lt.getElement(ObrasAuxflat2,i)
    ObrasAuxflat[i]=Obra

for Obra in ObrasAuxflat:


    FechaReal=Obra['dateacquired']


    if  FechaReal < inicio:
        ObrasAuxflat.remove(Obra)
    else:
        break


for Obra in ObrasAuxflat[::-1]:
    FechaReal=Obra['dateacquired']


    if  FechaReal > final:
        ObrasAuxflat.remove(Obra)
    else:
        break
```



Como se puede ver en las figuras anteriores hay varias operaciones que se repiten N veces, donde N es el número de artistas que están entre la fecha inicial y la fecha final. También hay un pedazo de código que parece tener complejidad N^2. Sin embargo, las listas que recorren no son en verdad tan grandes, pues el primer for son todos los años que pueden existir entre el año de la fecha inicial y el año de la fecha final y el segundo for son todos los meses que pueden existir en un año. Es decir, Si M=(año final-año inicial), entonces este ciclo tendrá complejidad 12*M por lo que en verdad es lineal, pues el 12 es constante independientemente de que años se ingresen como input. A su vez, M siempre va a ser muchos ordenes de magnitud menor que N (el número de artistas en el rango de fechas) , pues usualmente hay muchos más artistas por año. Por lo tanto, no se va a considerar este doble for como complejidad cuadrática.

Luego, hay un sorting de la lista de todos los artistas por fecha de nacimiento y nuevamente, obtenemos complejidad N*log(N) en todos los casos pues se utilizó mergesort. Finalmente hay otros fors que parecen tener complejidad O(N) aunque en realidad son mucho más cortos, pues estos solo quitan de la lista algunos artistas que se pueden pasar por arriba o por debajo de las fechas inicial y final (pues se buscó por mes-año no por día-mes-año). Por lo tanto, estos fors recorrerán menos artistas de los que pudieron haber nacido en un mes (hay un break).

Finalmente obtenemos otros fors con complejidad constante 3.

Por lo tanto, aproximamos la complejidad de este requerimiento con O(N*log(N)).

- Requerimiento 3:

Para este requerimiento podemos ver en la figura anterior que la gran mayoría de operaciones solo se ejecutan una vez. Sin embargo, las operaciones que más se repetirían en el peor caso tienen orden lineal, pues se recorren las obras del artista buscado en varias ocasiones. Por lo tanto, la complejidad sería O(N) donde N es la cantidad de obras que tiene el artista.

- Requerimiento 4:

```
def Nacionalidad_obras(catalog):
    """
    Lista con la nacionalidad
    """

    #se bisca el mapa con nacionalidad - obra
    artistas = catalog["nationalityartworks"]
    llaves = mp.keySet(artistas)
    retorno = lt.newList()


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

        #añade a una lista el la nacionalidad y el numero de obras que tiene
        mom = [lt.getElement(llaves, x),int(lt.size(grupo))]
        lt.addLast(retorno, mom)

    #Se ordenan los valores
    mrgsort.sort(retorno, compArtwrkByNatio)

    return retorno
```

t

N · t

t · log(t)

Como se puede ver en la figura anterior, este requerimiento es bastante simple. Se recorre una vez el diccionario que tiene como llaves las nacionalidades y como values las obras de estas nacionaliades, esto tiene complejidad lineal. Luego se cuenta cuántas obras hay de cada nacionalidad y finalmente se ordena esta lista comparando cuál nacionalidad tiene más obras. Como este ordenamiento se hace con un mergesort, la complejidad de este requerimiento se estima como O(N*log(N)). Donde N es la cantidad de nacionalidades en la base de datos.

- Requerimiento 5:

```
def Transporte(catalog,depa):

    '''
    Calcula el costo e inforación asociada a transportar un departamento del museo
    '''
    #Obtiene las obras del catalogo
    ObrasTotal=catalog['artworksbyDepartment']
    ObrasDepto=mp.get(ObrasTotal,depa)['value']
    #Crea una nueva lista para las obras del departamento a transportar
    ObrasDepto1=lt.newList()
    ObrasDepto2=lt.newList()
    #Acumulación del precio de transporte por obra
    TotalPrecio=0
    #Acumulación del peso total de las obras
    TotalPeso=0
    #Inicializa los precios máximos y sus correspondientes obras y las fechas mínimas con sus correspondientes obras
    maxC1=-1
    maxC2=-1
    maxC3=-1
    maxC4=-1
    maxC5=-1

    eltoC1={'name':'','dateacquired':'','constituentid':'','date':'','medium':'','dimensions':'','department':''
    ,'creditline':'','classification':'','circumference':'','depth':'','diameter':'','height':'','length':'','weight':'','width':''}

    eltoC2=eltoC3=eltoC4=eltoC5=eltoP1=eltoP2=eltoP3=eltoP4=eltoP5=eltoC1


    maxP1=2030
    maxP2=2030
    maxP3=2030
    maxP4=2030
    maxP5=2030


    #Recorre todas las obras del catálogo para ver calcular su precio
    for i in range(lt.size(ObrasDepto)-1):

        #Obtiene la obra
        Obra=lt.getElement(ObrasDepto,i+1)
        #Obtiene la fecha de la obra
        fecha=int(Obra['date'])

        if fecha:
            fecha=int(fecha)
        else:
            fecha=2021

        #Función que calcula el costo de transporte de una obra
        costo=CalcularCosto(Obra)
        #Función que calcula el peso de una obra
        peso=CalcularPeso(Obra)
        #Se le añade el atributo 'costo' a cada obra
        Obra['cost']=costo
        #Se acumulan el precio y el peso
        TotalPrecio+=costo
        TotalPeso+=peso
```

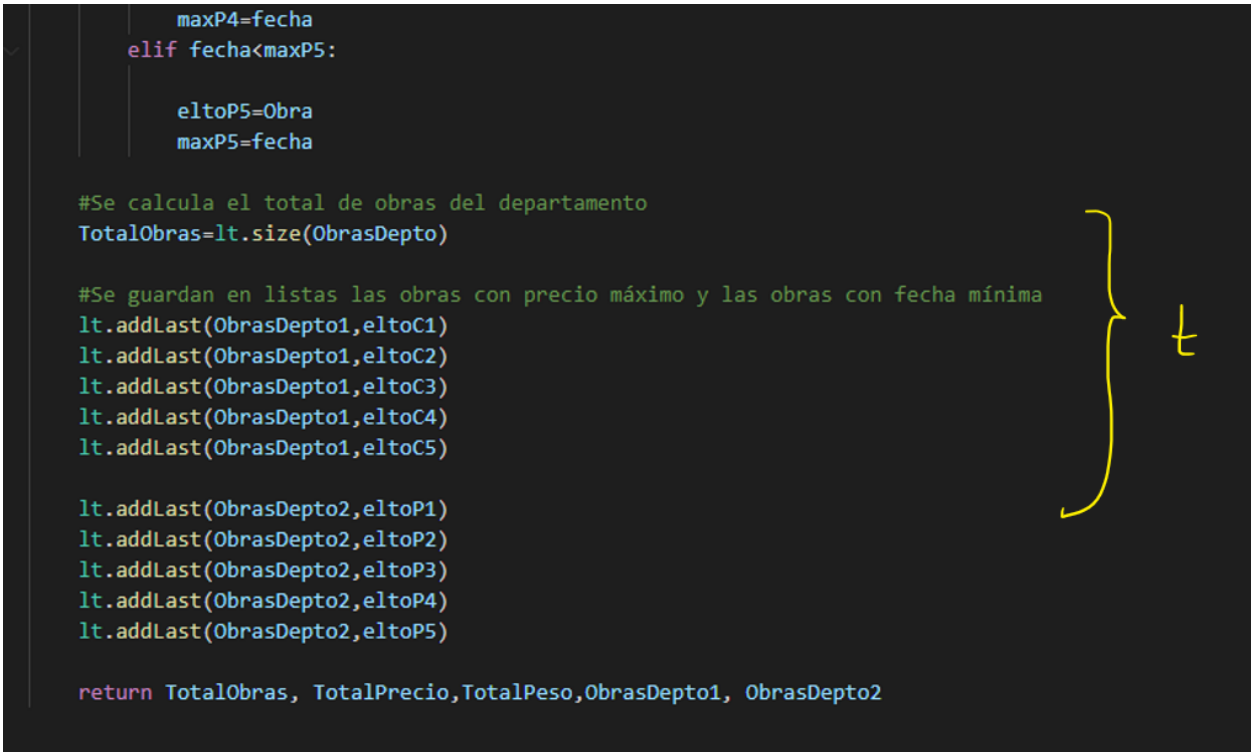
lineal

t

N·t

```
    #Se van actualizando las obras con precio máximo y las obras con fecha mínima
    if costo>maxC1:
        eltoC5=eltoC4
        maxC5=maxC4

        eltoC4=eltoC3
        maxC4=maxC3

        eltoC3=eltoC2
        maxC3=maxC2

        eltoC2=eltoC1
        maxC2=maxC1


        eltoC1=Obra
        maxC1=costo
    '''
    -
    -
    Muchos ifs
    -
    -

    '''
    elif costo>maxC2:
```

t



Como se puede ver en las figuras anteriores, la complejiad de este requerimiento es lineal, pues todas las operaciones se ejecutan una única vez, excepto dentro del único for que recorre todas las obras de dicho departamento del museo y calcula costos y peso de las obras a transportar. Si N es el número de obras que hay en el departamento, entonces este requerimiento tendría complejidad O(N). Sin embargo, aunque es lineal, este N suele ser muy grande. Por ejemplo, para el ejemplo del reto el departamento Drawings & Prints tiene más del 65% de todas las obras del museo, por lo que estaríamos recorriendo una inmensa cantidad de obras, aproximándonos a las complejidades que veíamos en el reto 1 pues hay que recorrer casi toda la base de datos.

Resumen y comparación con el reto 1:

Complejidad de los Requerimientos		
	Reto 1	Reto 2
REQ1	O(N*log(N))	O(N*log(N))
REQ2	O(N*log(N))	O(N*log(N))
REQ3	O(N)	O(N)
REQ4	O(N^2)	O(N*log(N))
REQ5	O(N)	O(N)

Como podemos ver en la figura anterior, todos los requerimientos tuvieron la misma complejidad temporal teórica salvo el REQ4 que mejoró de tener una complejidad cuadrática a ser Nlog(N). Sin embargo, esta notación no refleja las mejoras que ocurrieron en los tiempos realmente, pues, aunque todos los requisitos parecen tener la misma complejidad, los ordenes

de la N son muy distintos. Es decir, en el reto 1 en muchos de los requisitos teníamos que recorrer la base de datos completa (recorrer toda la lista de artistas o toda la lista de obras). En el reto dos las listas que tenemos que recorrer son MUCHO más pequeñas, pues gracias a los maps podemos escoger las llaves de las listas que verdaderamente nos interesa recorrer.

Por ejemplo, en el requerimiento 4 antes teníamos que recorrer todos y cada uno de los artistas para ir contando las nacionalidades de ellos. En el archivo large, hay 15224 artistas. Ahora, en este reto podemos utilizar el map que creamos anteriormente que tiene como llaves las nacionalidades y como values las listas con todos los artistas de cada nacionalidad. Por lo tanto, únicamente tenemos que recorrer las nacionalidades posibles (en el archivo large no hay más de una centena de ellas). De esta forma, aunque en la tabla parece que no mejoran los tiempos, en este caso estamos hablando de que N en el requerimiento 1 era 15224 mientras que en el reto dos N no es más de 100.

Otro ejemplo está en el requisito 5. Antes, teníamos que recorrer todas las obras para saber si estaban en el departamento buscado y luego hacer los cálculos. Ahora, podemos únicamente recorrer las obras del departamento que nos interesan usando el map que tiene como llaves los departamentos del museo y como values las listas de todas las obras por departamento.

Por lo tanto, en todos los requerimientos estamos disminuyendo el tiempo de ejecución aunque en la tabla no lo parezca. Esto se hará más claro viendo las siguientes gráficas que muestran los tiempos de ejecución de todos los requerimientos en este reto.

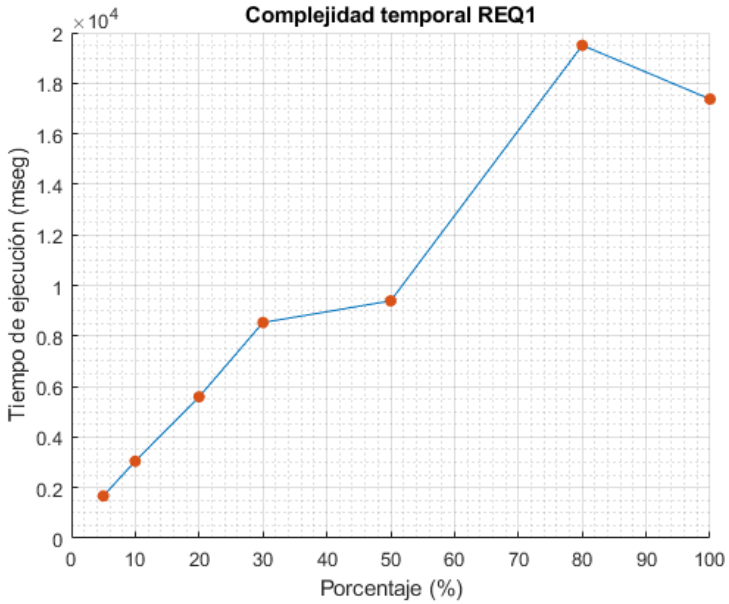
Pruebas de tiempos de ejecución y memoria:

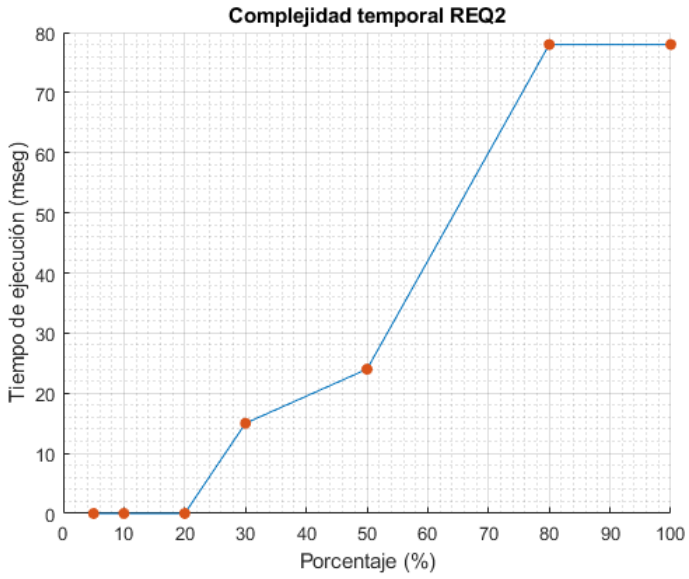
- Para el REQ1 se utilizó el input: desde- 1920 hasta- 1985
- Para el REQ2 se utilizó el input: desde- 1980-3-21 hasta- 1990-3-21
- Para el REQ3 se utilizó el input: Artista- Louise Bourgeois
- En el REQ4 no se necesita un input
- Para el REQ5 se utilizó el input: Departamento- Drawings & Prints
-

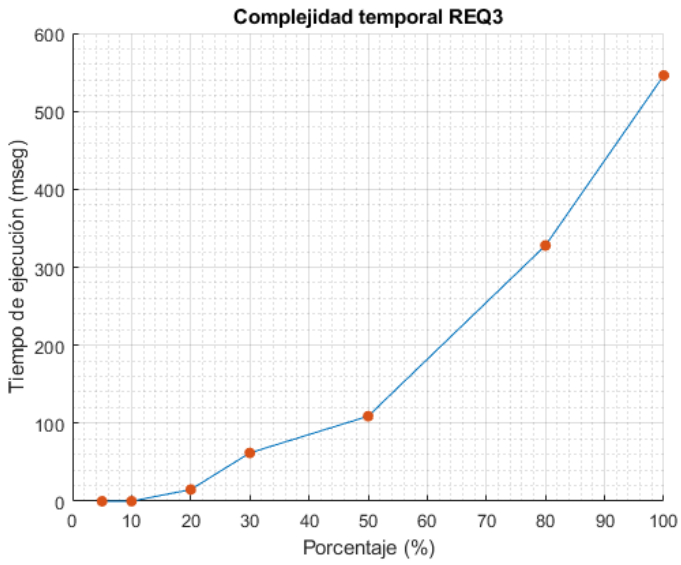
Memoria Utilizada por requerimiento:

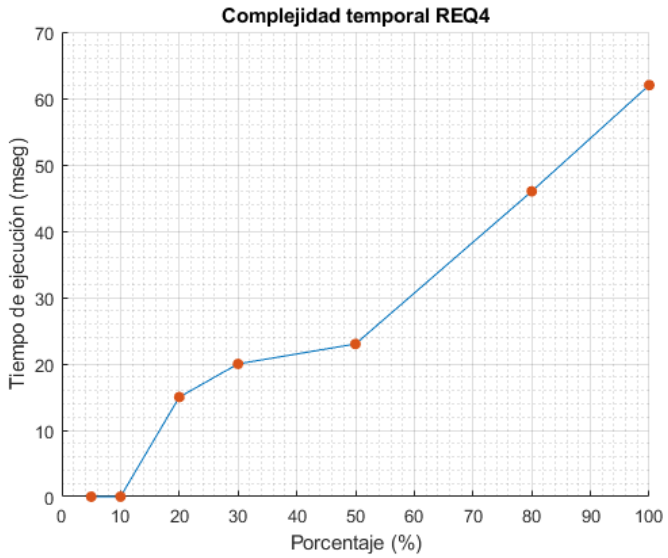
Memoria Utilizada por Requerimiento (GB)	
REQ1	3.72 - 3.74
REQ2	3.63-3.63
REQ3	4.11 -4.21
REQ4	3.81 -4
REQ5	3.92 -4.6

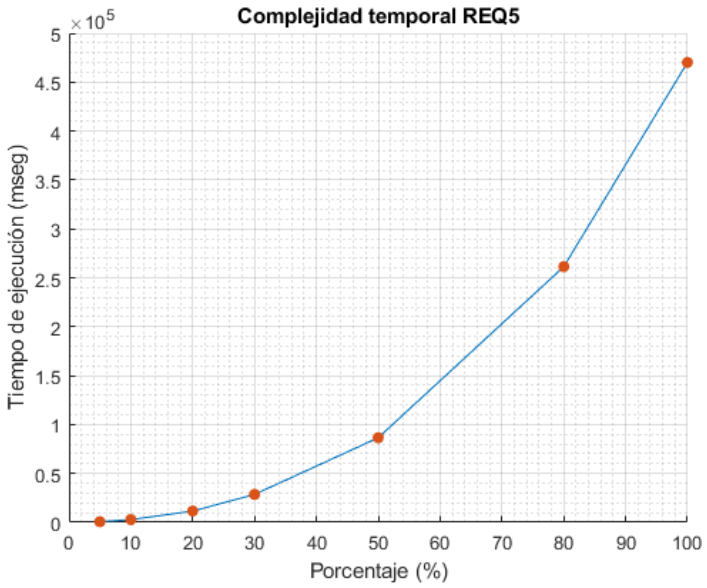
Gráficas de tiempos de ejecución:

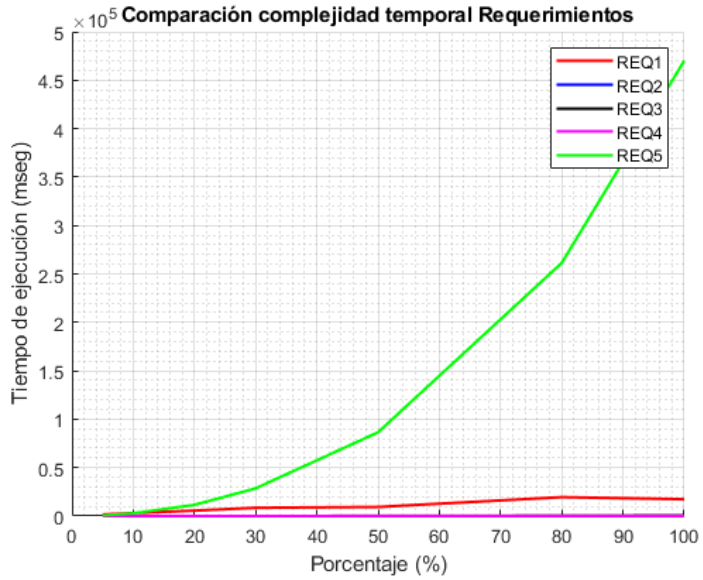





Complejidad temporal REQ3
Tiempo de ejecución (mseg)
Porcentaje (%)
0
100
200
300
400
500
600
0
10
20
30
40
50
60
70
80
90
100


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





Analizando las gráficas encontramos que los requerimientos 3 y 5 Muestran un aparente comportamiento cuadrático con diferentes proporciones, mientras que en los requerimientos 1, 2 y 4 es más difícil especificar su comportamiento gracias a que no es consistente como aumenta el crecimiento temporal, en especial en el primero y segundo ya que el dato final casi no cambia o disminuye el tiempo.

Cabe resaltar que los tiempos obtenidos fueron mucho más bajos en este reto, pues el que más tardó (requerimiento 5) con una enorme diferencia con los otros requerimientos se demoró

aproximadamente 8.3 minutos, mientras que en el reto dos estábamos obteniendo tiempos de ejecución superiores a los 10 minutos en casi todos los requerimientos.

Al igual que en el reto 1, es difícil saber solo con las gráficas si el comportamiento sí es verdaderamente lineal o nlog(n) o lo que sea, pues el porcentaje de la base de datos que ingresamos en el código no es directamente proporcional al 'N' que le va a entrar a cada requerimiento. Por lo tanto, no se puede analizar de manera apropiada los resultados obtenidos.