Documento de Análisis

Estudiante1: Santiago Sinisterra Arias 202022177 s.sinisterra@uniandes.edu.co – Realizo requerimiento 3

Estudiante2: Juan Felipe Serrano 201921654 j.serrano@uniandes.edu.co Realizo requerimiento 4

Complejidad:

Comparación de complejidad:

Req 1 reto 1: O(n^3/2)

```
def getagerange(catalog, date1, date2):
    artists = lt.iterator(catalog['Artists'])
    cronartists = lt.newList("ARRAY_LIST")
    for men in artists:
        if (int(men['BeginDate']) >= int(date1) and int(men['BeginDate']) <= int(date2)):
            lt.addLast(cronartists, men)
    sa.sort(cronartists, compareages)
    return cronartists
```

Req 2 reto 1: O(n^5/2)

```
def cronartwork(catalog, date1, date2):
    tamano = lt.size(catalog["Artworks"])
    sublist = lt.subList((catalog['Artworks']), 1, tamano)
    sublist = sublist.copy()
    c = lt.size(sublist) - 1
    while c > -1:
        element = lt.getElement(sublist, c)
        if element['DateAcquired'] < date1 or element['DateAcquired'] > date2:
            lt.deleteElement(sublist, c)
        sa.sort(sublist, cmpArtworkByDateAcquired)
        c = c-1
    return sublist
```

Req 3 reto 1: O(n^3/2)

```
def getartwoksandtech(catalog, artist):
    prod = lt.iterator(catalog['Productions'])
    for element in prod:
        if element['Artist'] == artist:
            bruh = element
    sa.sort(bruh['Artworks'], comparetechniques)
```

Req 4 reto 1: O(n^2)

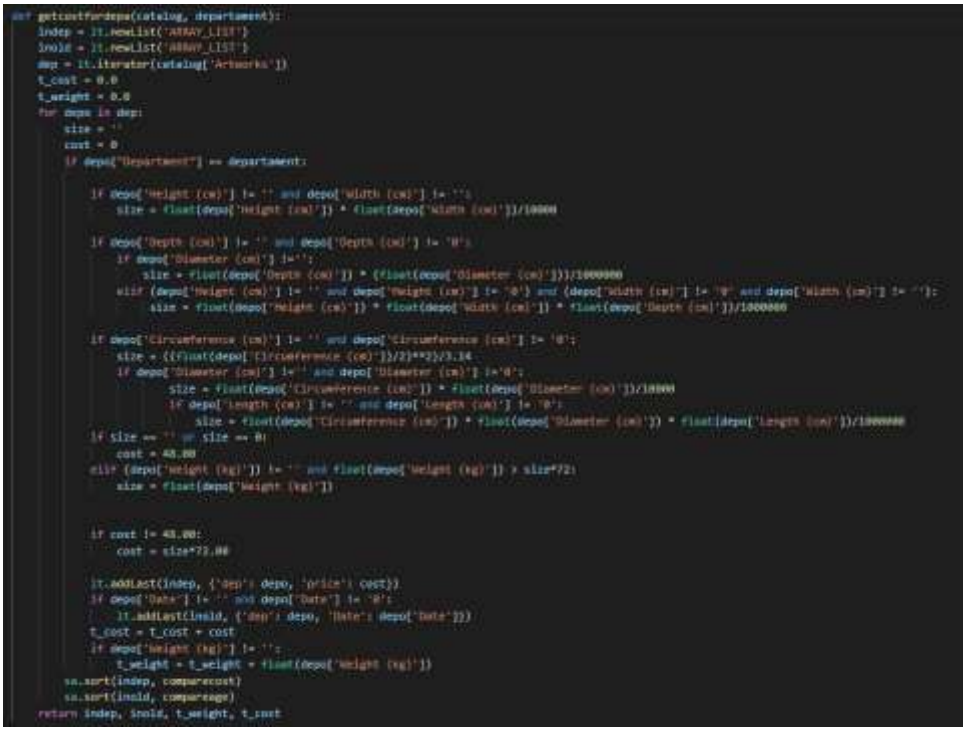
```
def organizeCountry(catalog):
    Artworks = catalog['Artworks']
    iterator = it.iterator(Artworks)
    ## Primero toca organizar las obras en listas por sus paises
    Paises = []
    ObrasPorPais = []
    for artwork in iterator:
        tamañoPaises = Paises.__len__()
        ## Caso inicial de los arrays
        if tamañoPaises == 0:
            Paises.append(artwork["Nationality"])
            ObrasPorPais.append([artwork])
        else:
            ## Mirar en todos los paises a ver si hay un lugar donde poner esta obra
            for index in range(0,tamañoPaises):
                ## Caso en el que encuentra el pais donde deberia poner la obra
                if str(artwork["Nationality"]) == str(Paises[index]):
                    ## La obra se coloca en el index actual de obras por pais en la lista de ese pais
                    ObrasPorPais[index].append(artwork)
                    break
                ## Si ya llego al final de la lista y no encontro donde poner la artwork creelo una
                # nueva categoria y guardela ahi
                elif index == tamañoPaises-1 and str(artwork["Nationality"]) != str(Paises[index]):
                    Paises.append(artwork["Nationality"])
                    ObrasPorPais.append([artwork])


    ## Ahora tenemos que organizar en terminos de que tan grande sea cada categoria
    numObrasPorPais = []
    ## Llenamos un nuevo array con los tamaños de los arrays
    for iteracion in ObrasPorPais:
        size = iteracion.__len__()
        numObrasPorPais.append(size)

    ## Organizamos el array teniendo en cuenta sus size mayor a menor
    nombresOrdenados = []
    ObrasOrdenadas = []
    numObrasOrdenadas = []
    iterador = 0
    sentinela = numObrasPorPais.__len__()
    while sentinela > 0:
        max = 0
        for i in range(0, numObrasPorPais.__len__()):
            pais = numObrasPorPais[i]
            if pais >= max:
                max = pais
                iterador = i
        nombresOrdenados.append(Paises[iterador])
        ObrasOrdenadas.append(ObrasPorPais[iterador])
        numObrasOrdenadas.append(max)
        numObrasPorPais.pop(iterador)
        Paises.pop(iterador)
        ObrasPorPais.pop(iterador)
        sentinela = numObrasPorPais.__len__()

    return nombresOrdenados,ObrasOrdenadas,numObrasOrdenadas
```

Req 5 reto 1: O(n^3/2)



Req 1 Reto 2: O(n^3/2) En la siguiente foto podemos ver como este requerimiento solo tiene dos operaciones importantes para la complejidad, un for loop y un shell sort. De los dos el de mayor complejidad es el shell sort por lo que la complejidad del requerimiento en total seria n^3/2

```
189    def filtrarArtistasPorAños(catalog, añoInicial , añoFinal):
190        listaRespuesta = lt.newList()
191        llaves = mp.keySet(catalog['artists'])
192        iterator = lt.iterator(llaves)
193        for llave in iterator:
194            dupla = mp.get(catalog['artists'],llave)
195            artista = me.getValue(dupla)
196            añoArtista = int (artista['BeginDate'])
197            if int(añoInicial) <= añoArtista and añoArtista <= int(añoFinal):
198                lt.addLast(listaRespuesta, artista)
199
200        ## Ahora que ya tenemos la lista filtrada podemos usar una funcion de ordenamiento y returnearla
201        sa.sort(listaRespuesta, compararArtistasPorAño)
202        return listaRespuesta
```

Req 2 Reto2: O(n^5/2)

Aclaración: Para calcular la complejidad en el peor de los casos toca considerar que el segundo for solo ocurre en los dos casos marginales del rango, por ende, al considerar que se active este for y el sort correspondiente como la peor situación, el primer for tendría una complejidad de O(2), para el segundo tener la complejidad de lo que sería la mitad de los datos presentes en uno de los años y la otra mitad en el otro, siendo esto O(N), y sus correspondientes sorts de $O(N^{3/2})$, con lo que se obtiene una complejidad $O(N^{5/2})$.

```
def filtrarObrasPorAños(catalog,fechaInicial,fechaFinal):
    listaRespuesta = lt.newList()
    llavesa = mp.keySet(catalog['DateAcquired'])
    llaves = lt.iterator(llavesa)

    contador = 0

    for llave in llaves:
        if(int(fechaInicial[:4]) < int(llave) and int(llave) < int(fechaFinal[:4])):
            dupla = mp.get(catalog['DateAcquired'],llave)
            obra = me.getValue(dupla)
            sa.sort(obra, compareByDate)
            res = {'anio': llave, 'obras': obra}
            lt.addLast(listaRespuesta, res)
            contador = contador + lt.size(obra)

        elif (int(fechaInicial[:4]) == int(llave) or int(llave) == int(fechaFinal[:4])):
            listasup = lt.newList()
            dupla = mp.get(catalog['DateAcquired'],llave)
            obra = me.getValue(dupla)
            obriterator = lt.iterator(obra)
            for o in obriterator:
                fechaArtista = o['DateAcquired']
                if fechaInicial <= fechaArtista and fechaArtista <= fechaFinal:
                    lt.addLast(listasup, o)
            sa.sort(listasup, compareByDate)
            res = {'anio': llave, 'obras': listasup}
            lt.addLast(listaRespuesta, res)
            contador = contador + lt.size(listasup)

    sa.sort(listaRespuesta, compareByDate2)

    return listaRespuesta, contador
```

Req 3 Reto2: O(n^3/2) El requerimiento 3 usa principalmente un for y dos sort por lo que la complejidad final del requerimiento debería ser en el peor de los casos la complejidad del shell sort, se concluye así que tiene una complejidad de O(n^3/2)

```
def clasificarObrasDeArtistaPorTecnica(catalog, nombre):
    mapartists = mp.valueSet(catalog['artists'])
    iteration = lt.iterator(mapartists)
    elegido = []
    for artista in iteration:
        if artista['DisplayName'] == nombre:
            elegido = artista
            break
    sa.sort(elegido['Obras'], comparetech)

    popularity = lt.newList("ARRAY_LIST")
    art1 = None
    art2 = None
    n = 0
    tech_num = 0
    artlist = lt.newList("ARRAY_LIST")
    iteration = lt.iterator(elegido['Obras'])

    for artwork in iteration:
        art1 = artwork['Medium']
        if art1 != art2:
            tech_num = tech_num + 1
            if art2 != None:
                lt.addLast(popularity, dic)
            n = 1
            artlist = lt.newList("ARRAY_LIST")
            lt.addLast(artlist, artwork)
            dic = {"Medium": artwork['Medium'], 'Number': n, 'Obras': artlist}
            art2 = art1
        else:
            n = n + 1
            art2 = art1
            lt.addLast(artlist, artwork)
            dic = {"Medium": artwork['Medium'], 'Number': n, 'Obras': artlist}

    if art1 == None:
        tech_num = 0
    elif art1 == art2:
        lt.addLast(popularity, dic)
    sa.sort(popularity, comparetechniques)
    return elegido, popularity, tech_num
```

Req 4 Reto2: O(n) El requerimiento 4 se beneficia en gran manera por la falta de recorridos para identificar elementos específicos que originalmente se guardaban en un arreglo. Al implementar mapas estos recorridos son evitados y se logra una complejidad de O(1) por lo que se disminuyen en gran medida la cantidad de for que se tiene que incluir en la solución. La solución solo tiene un for para buscar en todo el mapa de nacionalidades cual es la que tiene mayor número de obras asociadas, por lo que su complejidad final sería O(n)

```
301  def obrasPorNacionalidad(catalog):
302      listaRespuesta = lt.newList()
303      maxNumObras = 0
304      maxNacionalidad = None
305      listkeyset = mp.keySet(catalog['Nationalities'])
306      iterador = lt.iterator(listkeyset)
307      mapaNacionalidades = mp.newMap(100,
308                                     maptype='PROBING',
309                                     loadfactor=0.5)
310      for nacionalidad in iterador:
311          dupla = mp.get(catalog['Nationalities'],nacionalidad)
312          mp.put(mapaNacionalidades, nacionalidad, lt.size(me.getValue(dupla)))
313          listanacionalidad = me.getValue(dupla)
314          if lt.size(listanacionalidad) > maxNumObras:
315              maxNumObras = lt.size(listanacionalidad)
316              maxNacionalidad = nacionalidad
317
318      dupla = mp.get(catalog['Nationalities'], maxNacionalidad)
319      listaRespuesta = me.getValue(dupla)
320
321      return listaRespuesta, maxNacionalidad, maxNumObras, mapaNacionalidades
```

Req 5 Reto2: O(n^3/2) Este requerimiento nuevamente solo usa un for y el shell sort al final por lo que se conlcuye que la complejidad máxima seria n^3/2

Análisis de datos:

Complejidades: Revisando la comparación de las complejidades del anterior reto se puede ver una mejora considerable de complejidad de todos los requerimientos. Mas específicamente se puede ver una mejora en el requerimiento 2 y el 4, ya que el req 2 tenía una complejidad bastante alta por un mal posicionamiento de un sort y el req 4 tenía n^2, esto mejoro en gran medida en el reto 2 con la implementación de mapas ya que su tiempo de búsqueda se reduce en gran medida y sus iteraciones sobre los artworks se reduce el número de artworks a revisar drásticamente. En adición, debido a que utilizamos muchos mapas para las búsquedas directas de todos los demás requerimientos esto reduce la complejidad todos los recorridos necesarios para buscar términos específicos, sin embargo, esto se dificulta de igual manera cuando se quiere conseguir datos organizados y ahí toca recurrir nuevamente a las listas.

Req1: La complejidad del algoritmo se mantuvo igual (n^3/2), sin embargo debido a que la lista que ser recorre en la versión del reto 1 es todos los artoworks y la lista que se recorre en el reto 2 es un alista más pequeña, la solución del reto dos es más rápida en el mundo real aunque tengan la misma complejidad ideal.

Req2: La complejidad del req 2 cambia de n^5/2 a n^3/2 por lo que su mejora es clara, no solo esto, pero gracias a un cambio que se hizo en el manejo de las listas de los rangos el tiempo de carga del reto 2 tiende a ser mucho menor del esperado gracias a las cortas listas debido a los mapas.

Req3: La complejidad del req 3 se mantiene igual, pero por la misma razón anterior, el recorrido de listas es mucho más corto en el caso real por lo que es mucho más veloz como se puede ver después en las pruebas de tiempo.

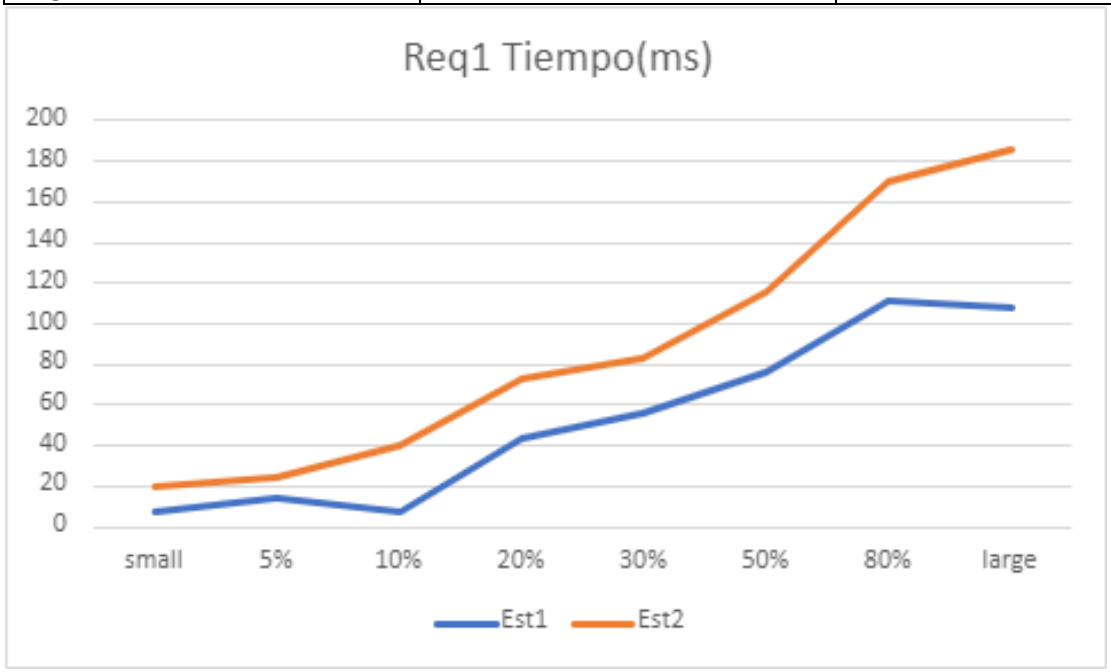
Req4: La complejidad del req 4 se cambia de n^2 a complejidad de n gracias a que la demora de acceder a los datos de los mapas reducen un ciclo de for completo del algoritmo haciendo que se demore mucho menos.

Req5: La complejidad nuevamente se mantiene igual, pero en la realidad el recorrido es mucho menor ya que no se tiene que hacer recorridos de toda la lista de obras sino que solo se hace la del medio especifico.

Tiempos de ejecución:

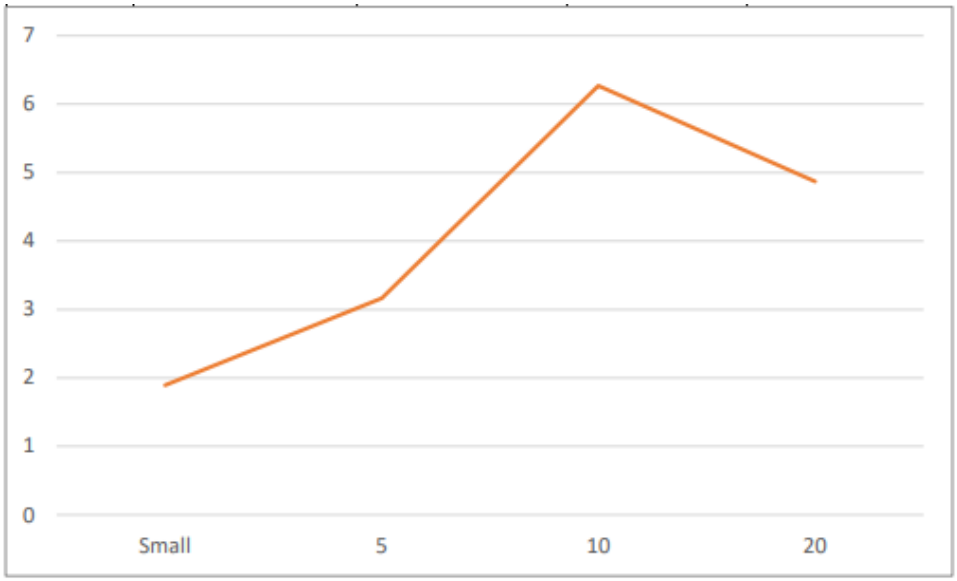
Req 1 Reto 2:

Req1 Reto2		
Tamaño	Tiempo (ms)	Est2
small	6,943319559	19,9911320
5%	14,37384129	24,55174541
10%	7,653674841	39,75131965
20%	43,81189179	72,61718440
30%	55,49042153	83,07569814
50%	76,37338996	115,2669935
80%	111,001575	169,7024219
large	107,493772	186,0682833



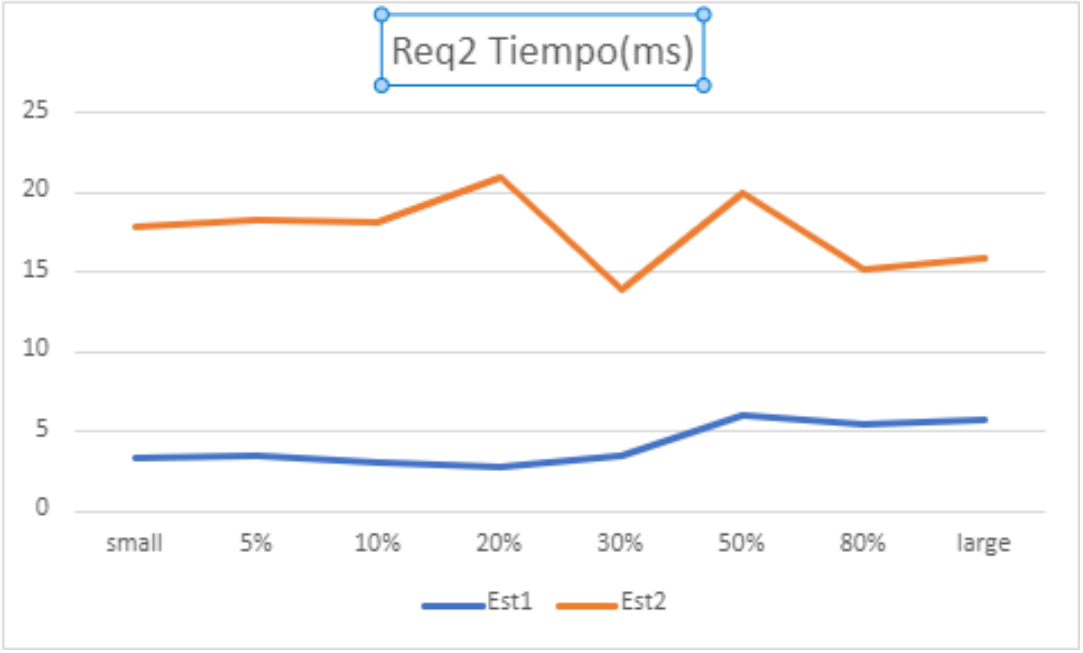
Req1 Reto1:

Req1	Tamaño_artistas	Tamaño_obras	Tiempo (ms)
Small	1948	768	1.893
5	4996	7572	3.165
10	6656	15008	6.268
20	8724	29489	4.869



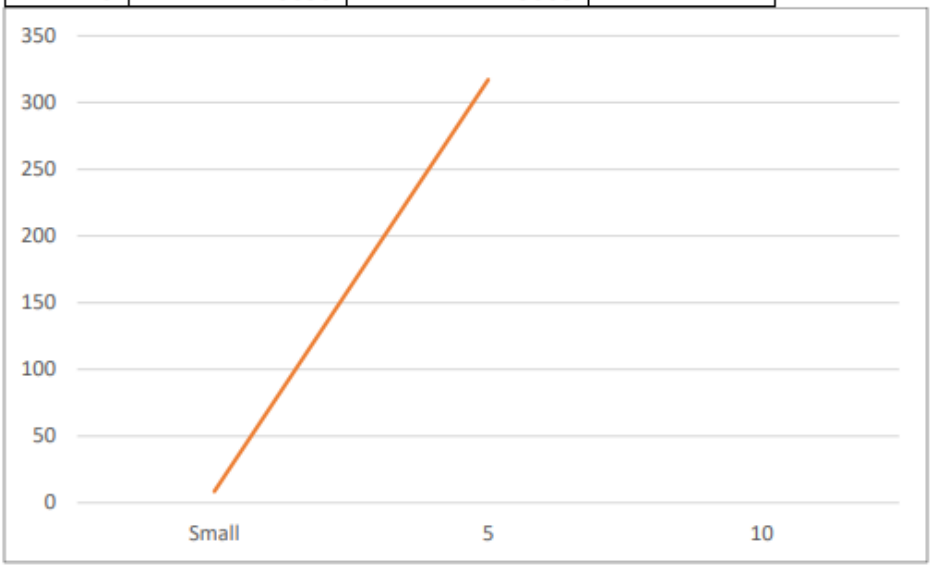
Req 2 Reto2:

Req2 Reto 2 Tiempo (ms)		
Tamaño	Est1	Est2
small	3,315648079	17,8430886
5%	3,44699955	18,2319753
10%	3,105351925	18,2020009
20%	2,822018385	21,0351033
30%	3,449903965	13,8982668
50%	6,024789095	19,9861839
80%	5,397404432	15,2205000
large	5,794452667	15,9042487



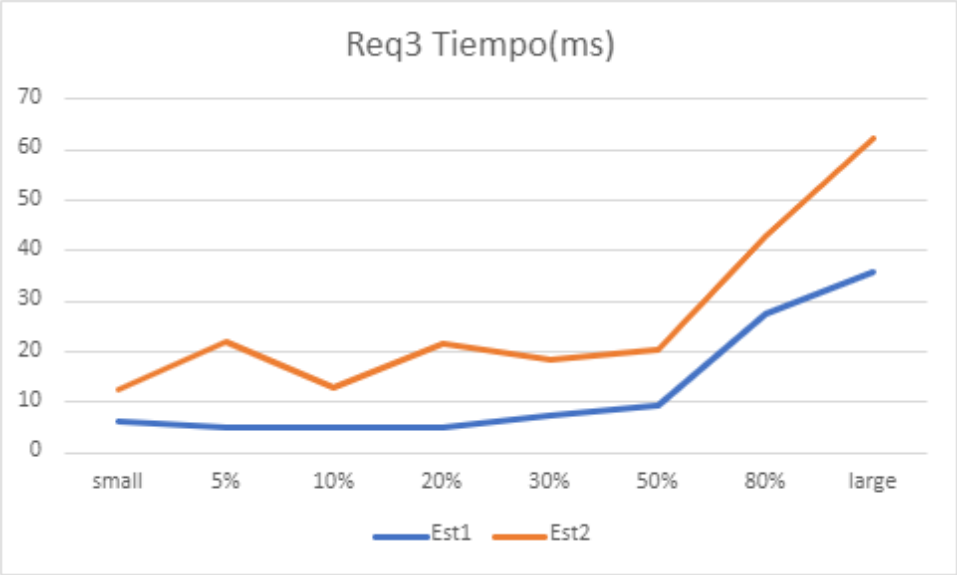
Req2 Reto1:

Req2	Tamaño_artistas	Tamaño_obras	Tiempo (ms)
Small	1948	768	8.444
5	4996	7572	317.143
10	6656	15008	



Req 3 Reto1:

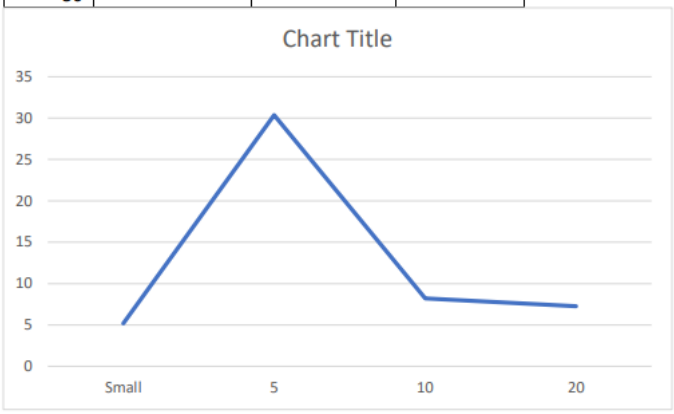
Req3 Reto2Tiempo (ms)		
Tamaño	Est1	Est2
small	6,006117821	12,574293
5%	4,976615429	22,152379
10%	5,031712294	13,059099
20%	4,801939249	21,453245
30%	7,357511759	18,469853
50%	9,261232853	20,341278
80%	27,64831519	42,796186
large	35,6582098	62,245119



Req 3 Reto 1:

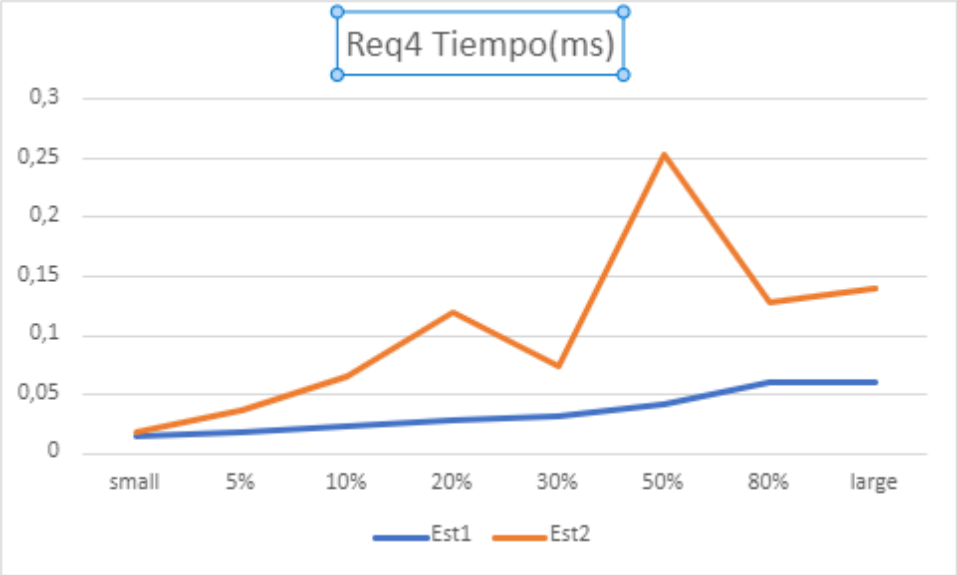
Req3	Tamaño_artistas	Tamaño_obras	Tiempo (ms)
Small	1948	768	5.189
5	4996	7572	30.36585951
10	6656	15008	8.215

20			7.257
30			
50			



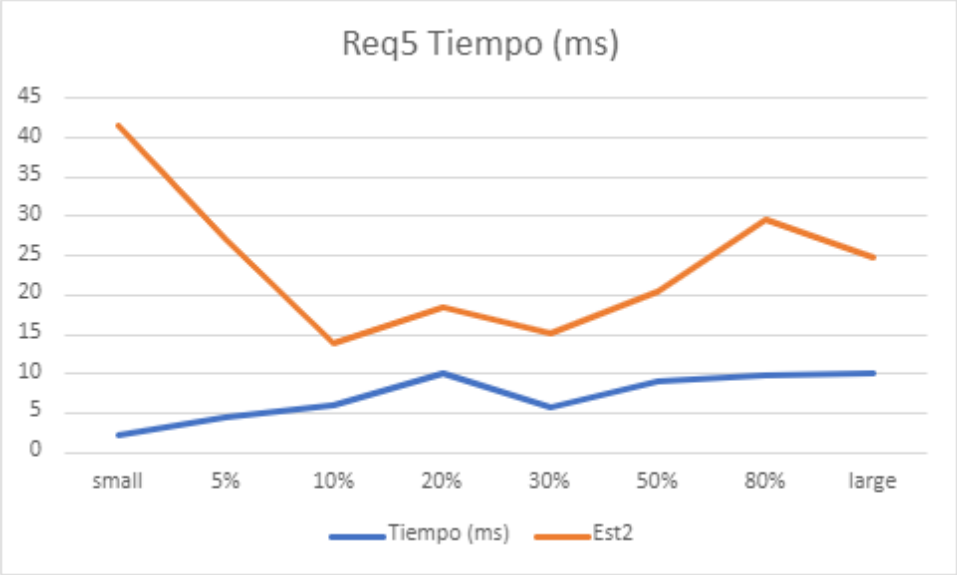
Req 4:

Req4 Reto2 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,015141964	0,017364502
5%	0,018622398	0,036902905
10%	0,02279377	0,065824747
20%	0,0280056	0,118683577
30%	0,031978846	0,073800
50%	0,042134285	0,253322601
80%	0,060000658	0,128656387
large	0,059602261	0,139520



Req 5 Reto 2:

Req5 Reto2		
Tamaño	Tiempo (ms)	Est2
small	2,234867334	41,45515
5%	4,374613047	27,12984
10%	5,893184185	13,74966
20%	9,935406208	18,37101
30%	5,759776831	15,20328
50%	8,974297047	20,53516
80%	9,918561935	29,71318
large	9,961227417	24,73095



Req5 Reto1:

Req5	Tamaño_artistas	Tamaño_obras	Tiempo (ms)
Small	1948	768	33.785
5	4996	7572	242
10	6656	15008	
20			
30			
50			

--