Nicolás Díaz Montaña – 202021006 – n.diaz9@uniandes.edu.co
Daniel R Alonso A - 201419873 - dr.alonso10@uniandes.edu.co

Análisis de complejidad

Carga de datos:

```
if int(inputs[0]) == 1:
        print("Cargando información de los archivos ....")
        catalog = initCatalog()
        start_time = time.process_time()
        loadData(catalog)
        stop_time = time.process_time()
        elapsed_time_mseg = (stop_time - start_time)*1000
        print("-" * 74)
        print('Obras cargadas: ' + str(lt.size(catalog['artworks']))+ "\n")
        print('Artistas cargados: ' + str(lt.size(catalog['artists'])) + "\n")
        print ("Tiempo de carga: " + str(elapsed_time_mseg))
```

Figura 1 – Si el usuario ingresa 1

Una vez que el usuario oprima se llama a la función loadData() el cual va a cargar los elementos del csv al catalogo por medio de las funciones loadArtist(), loadArtworks() y load Nationality().

```
def loadArtworks(catalog):
    """
    Carga las obras del archivo.
    """
    booksfile = cf.data_dir + 'MoMA/Artworks-utf8-large.csv'
    input_file = csv.DictReader(open(booksfile, encoding='utf-8'))
    for artwork in input_file:
        model.addArtwork(catalog, artwork)
```

Figura 2 - controller.loadArtworks()

```
def addArtwork(catalog,artwork):
    if artwork["DateAcquired"] == "" or artwork["DateAcquired"] == "Unknown":
        hoy = date.today()
        artwork["DateAcquired"] = hoy.strftime("%Y-%m-%d")
    if artwork["Date"] == "" or artwork["Date"] == "Unknown":
        hoy = date.today()
        artwork["DateAcquired"] = hoy.strftime("%Y-%m-%d")
    lt.addLast(catalog["artworks"], artwork)
    addArtworkMedium(catalog,artwork)
    addDate_Artworks(catalog, artwork)
```

Figura 3 - model.addArtwork()

addArtwork() se encarga de añadir las fechas de adquisición y creación si la obra no las tiene. Finalmente agrega a la lista de artista, al mapa de medios y al mapa de fechas la obra en la que esta el ciclo en ese momento por medio de addlast(), addArworkMedium() y addDate_Artworks() respectivamente. Debido a que estas

funciones tienen un algoritmo de O(1), pero se encuentran en un ciclo como se evidencian en la figura del controller este resulta siendo O(n).

```
def loadArtists(catalog):
    """
    Carga los artistas.
    """
    booksfile = cf.data_dir + 'MoMA/Artists-utf8-large.csv'
    input_file = csv.DictReader(open(booksfile, encoding='utf-8'))
    for artist in input_file:
        model.addArtist(catalog, artist)
```

Figura 4 – controller.loadArtists()

```
def addArtist(catalog,artist):
    lt.addLast(catalog["artists"], artist)
    info = newArtist(artist)
    mp.put(catalog["artists_ID"], artist["ConstituentID"], info)
```

Figura 5 – model.addArtist()

Lo mismo se puede decir de addArtist() el cual aunque seria O(1) pero como esta dentro de un ciclo en loadArtists() todo resulta O(n)

```
def loadNationality(catalog):
    """
    Esta funcion adiciona un artista a la lista de obras que usan un medio
especifico.
    Los medios se guardan en un Map, donde la llave es el medio y el valor la
lista de obras de ese medio.
    """
    for obra in lt.iterator(catalog["artworks"]):
        codes = obra["ConstituentID"]
        nacionalidades = getArtworksNationality(catalog, codes)
        for nacionalidad in lt.iterator(nacionalidades):
            if nacionalidad == " " or nacionalidad == "":
                nacionalidad = "Nationality unknown"
            existnation = mp.contains(catalog["nationality"], nacionalidad)
            if existnation:
                entry = mp.get(catalog["nationality"], nacionalidad)
                nac = me.getValue(entry)
            else:
                nac = newNation(nacionalidad)
                mp.put(catalog["nationality"], nacionalidad, nac)
            lt.addLast(nac["obras"], obra)
```

Para loadNationality() se recorre la lista del catalogo “artworks” luego tiene que sacar los códigos de la obra en el que esta el recorrido, luego se itera sobre esos códigos para luego conseguir las nacionalidades con el catalogo “artists_ID” en O(1) para finalmente agregar la obra dependiendo de su nacionalidad lo cual quedaría en $O(n_1 + n_2)$ que al final seguiría siendo O(n).

```
def loadData(catalog):
    """
    Carga los datos de los archivos y cargar los datos en la
    estructura de datos
    """
    loadArtists(catalog)
    loadArtworks(catalog)
    loadNationality(catalog)
```
Figura 6 – controller.loadData()

Haciendo que al final la carga de datos sea O(3n).

Requerimiento 1

En el view.py el requerimiento 1 se ejecuta cuando el usuario escoge la opción 2 del menú:

```
elif int(inputs[0]) == 2:
    anio_inicial = input("Ingrese el año inicial: ")
    anio_final = input("Ingrese el año final: ")
    requerimiento1(catalog, anio_inicial, anio_final)
```

Veamos la complejidad de la función requerimiento1:

```
def requerimiento1(catalog, anio_inicial, anio_final):
    """
    Genera una lista cronológicamente ordenada de los artistas en un rango de anios.
    Retorna el total de artistas en el rango cronológico, y los primeros 3 y ultimos 3 artistas del rango.
    """
    org_anio = controller.artistsbyAnioD(catalog, anio_inicial, anio_final)
    print("\n")
    print("Total de artistas en el rango " + str(anio_inicial) + " - " + str(anio_final) + ": " + str(lt.size(org_anio)))
    print("-" * 50)
    last = controller.lastThreeD(org_anio)
    first = controller.firstThreeD(org_anio)
    print ("  Estos son los 3 primeros Artistas encontrados: ")
    printArtistData_Req1(first)
    print("-" * 50)
    print ("  Estos son los 3 ultimos Artistas encontrados: ")
    printArtistData_Req1(last)
```

Veamos la complejidad de las funciones artistbyAnioD(), lastThreeD(), firstThreeD().
La función artistbyAnioD() hace lo siguiente:

```
def artistsbyAnioD(catalog,anio_inicial,anio_final):
    """
    Organiza y retorna los artistas que esten en un rango de una fecha inicial y final.
    """
    artistas = copiarListaD(catalog['artists'],compareAnioD)
    ms.sort(artistas,ordAscArtiAnioD)
    pos = lt.isPresent(artistas,str(anio_inicial))
    pos_f = lt.isPresent(artistas,str(anio_final))
    l = int(pos_f)-int(pos)
    artists = lt.subList(artistas,pos,l+2)
    return artists
```

Las operaciones de complejidad mayor son la función copiarListaD y el mergesort. La función copiarListaD hace lo siguiente:

```
def copiarListaD(lista, cmpf):
    """
    Copia una lista con nueva cmpfunction cmpf
    """
    copia = lt.newList("ARRAY_LIST",cmpf)
    for elemento in lt.iterator(lista):
        lt.addLast(copia,elemento)
    return copia
```

Allí hay un ciclo que itera sobre la lista completa. En este caso la lista es catolog[‘artists’] por lo que la complejidad de esta función es O(n). El mergesort también se hace sobre n datos, la copia de artists, así que tiene complejidad O(n log(n)). En definitiva, esta función artistbyAnioD() tiene complejidad O(n log(n)).

Las funciones firstThreeD() y lastThreeD() son muy sencillas:

```
def firstThreeD(lista):
    """
    Retorna una lista con los tres primeros elementos de una lista.
    """
    first = lt.subList(lista,1,3)
    return first

def lastThreeD(lista):
    """
    Retorna una lista con los 3 ultimos elementos de una lista.
    """
    last = lt.subList(lista,lt.size(lista)-2,3)
    return last
```

En general, pues, el requerimiento 1 tiene complejidad O(n log(n)).

<u>Requerimiento 2</u>

```
elif int(inputs[0]) == 3:
        startDate = input("Fecha de Inicio (YYYY-MM-DD): ")
        finishDate = input("Fecha Final (YYYY-MM-DD): ")
        start_time = time.process_time()
        artworksBydate(catalog, startDate, finishDate)
        stop_time = time.process_time()
        elapsed_time_mseg = (stop_time - start_time)*1000
        print ("Tiempo transcurrido: " + str(elapsed_time_mseg))
```

El requerimiento 2 empieza cuando el usuario ingresa 3 y le pide una fecha inicial conjunto con una fecha final. Aquí se llama a la función de artworksBydate() la cual contiene y llama a las funciones organizeArtworksbyDate(), lastThreeD() y fisrtThreeD().

```
def artworksBydate(catalog, startDate, finishDate):
    """
    Genera una lista cronológicamente ordenada de las obras adquiridas
    por el museo en un rango de fecha. Retorna el total de obras en el rango
cronológico,
    total de obras adquiridas por compra y las primeras 3 y utimas 3 obras del
rango. ()()()
    """
```

```
    org_dates = controller.organizeArtworkbyDate(catalog, startDate,
finishDate)
    last = controller.lastThreeD(org_dates)
    first = controller.firstThreeD(org_dates)
    print("\n")
    print("Total de obras en el rango " + str(startDate) + " - " +
str(finishDate) + ": " + str(lt.size(org_dates))+ "\n")
    print("-" * 84 + "\n")
    print("Total de obras compradas en el rango: " +
str(controller.countPurchase(org_dates)) + "\n")
    print("-" * 84)
    print(("-" * 21) + "Estos son las 3 primeras Obras encontradas" + ("-" *
21) + "\n")
    printArtworkData(first)
    print("-" * 84)
    print(("-" * 22) + "Estos son las 3 ultimas Obras encontradas" + ("-" *
21) + "\n")
    printArtworkData(last)
    print("-" * 84)
```

Figura 7- artworksBydate()

La función organizeArtworksbyDate(), utiliza las fechas entregadas por el usuario para realizar un delta para encontrar el rango de fechas en las que se busca las obras. Luego se realiza un recorrido por cada una de esas fechas y se va llamando a la función getArtworksbyDate() la cual encuentra las obras ubicadas en el mapa del catalogo “date_artworks”.

```
def organizeArtworkbyDate(catalog, startDate, finishDate):
    """
    Organiza y retorna las obras que esten en un rango de
    una fecha inicial y final.
    """
    org = lt.newList()
    d_0 = date.fromisoformat(startDate)
    d_f = date.fromisoformat(finishDate)
    delta = d_f - d_0
    for day in range(delta.days + 1):
        new_day = d_0 + timedelta(days=day)
        new_date = new_day.strftime("%Y-%m-%d")
        getArtworksbyDate(catalog, new_date, org)
    return org
```

Figura 8 – model.organizeArtworksbyDate()

```
def getArtworksbyDate(catalog, date, org):
    """
    """
    fechas = catalog["dates_artworks"]
    existdate = mp.contains(fechas, date)
    if existdate:
        pareja = mp.get(fechas, date)
        valor = me.getValue(pareja)
```

```
        obras = valor["obras"]
        for obra in lt.iterator(obras):
            lt.addLast(org, obra)
```
Figura 9 – model.getArtworksbyDate()

En la función de la figura 9 se busca que las obras según la fecha entregada por el programa o usuario lo cual tiene un algoritmo de O(1) pero debido a que esta función actualiza la lista de la función organizeArtworksbyDate() recorriendo las obras del valor de la llave encontrada, lo cual vuelve a la función O(n).

Por otro lado, ya volviendo a la función de la figura 7, las funciones lastThreeD() y firstThreeD() funcionan utilizando subList() teniendo un algoritmo de complejidad de O(n).

```
def firstThreeD(lista):
    """
    Retorna una lista con los tres primeros elementos de una lista.
    """
    first = lt.subList(lista,1,3)
    return first

def lastThreeD(lista):
    """
    Retorna una lista con los 3 ultimos elementos de una lista.
    """
    last = lt.subList(lista,lt.size(lista)-2,3)
    return last
```
Figura 10 – firstThreeD() y lastThreeD()

Lo cual haría que el requerimiento 2, en su totalidad, sea de $O(n_1 + n_2 + n_3)$ o O(n).

<u>Requerimiento 3 (Daniel):</u>

En el view.py el requerimiento 3 se ejecuta cuando el usuario escoge la opción 4 del menú:

```
elif int(inputs[0]) == 4:
    nombre = input("Ingrese el nombre del artista: ")
    requerimiento3(catalog,nombre)
```

Veamos la complejidad de la función requerimiento3:

```
def requerimiento3(catalog,nombre):
    artworks = controller.artworksbyArtistD(catalog,nombre)
    print("\n")
    print("Total de obras del artista " + str(nombre) + ": " + str(lt.size(artworks)))
    lista = controller.artworksbyMediumD(artworks)
    medios = controller.contarMediosD(artworks)
    medio_max = controller.medioMaxD(lista)
    first = controller.firstThreeD(lista)
    last = controller.lastThreeD(lista)
    print("-" * 50)
    print("Total de medios usados por el artista en sus obras: " + str(medios))
    print("-" * 50)
    print("La técnica más usada por el artista es: " + str(medio_max))
    print("-" * 50)
    print("Listado de obras con la técnica más usada: ")
    print("-" * 50)
    print("Tres primeros: ")
    printArtworkData_Req3(first)
    print("-" * 50)
    print("Tres ultimos: ")
    printArtworkData_Req3(last)
```

Veamos la complejidad de las funciones artworksbyArtistD(), artworksbyMediumD(), contarMediosD() y medioMaxD().

La función artworksbyArtistD() hace lo siguiente:

```
def artworksbyArtistD(catalog,nombre):
    """
    Retorna una lt con las obras de un artista por su nombre.
    """
    copia = copiarListaD(catalog['artists'],compareNamesD)
    artista = getElementbyparameterD(copia,nombre)
    ide = artista['ConstituentID']
    artist = mp.get(catalog['obbyArt'], ide)
    if artist:
        obras = me.getValue(artist)['obras']
    return obras
```

La función getElementbyparameterD():

```
def getElementbyparameterD(lista, parameter):
    """
    Retorna un elemento de una lista dado un parametro, no lo elimina de la lista
    """
    pos = lt.isPresent(lista, parameter)
    if pos > 0:
        element = lt.getElement(lista, pos)
        return element
    else:
        return None
```

Como las listas usadas están implementadas con “ARRAY_LIST” y las operaciones de maps tienen complejidad O(1), vemos que la operación más compleja en esta función es copiarListaD(). Ed. la complejidad de artworksbyArtistD() es O(n).
La función artworksbyMediumD() hace lo siguiente:

```
def artworksbyMediumD(obras):
    """
    Retorna un lt con las obras que usan la tecnica o medio mas recurrente en obras.
    """
    # contamos numero de veces que aparece cada medio y agregamos este numero a una lista (numeros)
    medios = listaMediosD(obras)
    numeros = lt.newList("ARRAY_LIST")
    for medio in lt.iterator(medios):
        i = 0
        for obra in lt.iterator(obras):
            if obra['Medium'] == '':
                obra['Medium'] = 'Ninguno'
            if (medio['Medium'] == obra['Medium']):
                i += 1
        lt.addLast(numeros,i)
    # ordenamos la lista y sacamos el numero mas grande
    ms.sort(numeros,cmpfunction=ordenAscendenteD)
    num = lt.lastElement(numeros)
    # hallamos el medio (medio) correspondiente a este numero mas grande (num)
    for medio in lt.iterator(medios):
        i = 0
        for obra in lt.iterator(obras):
            if obra['Medium'] == '':
                obra['Medium'] = "Ninguno"
            if (medio['Medium'] == obra['Medium']):
                i += 1
        if i == num:
            m = medio
            break
    # hacemos una lista (lista) con las obras que usan el medio hallado (medio)
    lista = lt.newList("ARRAY_LIST")
    if m is not None:
        for obra in lt.iterator(obras):
            if obra['Medium'] == '':
                obra['Medium'] = "Ninguno"
            if (obra['Medium'] ==  m['Medium']):
                lt.addLast(lista,obra)
        return lista
```

En ella se usa la función listaMediosD():

```
def listaMediosD(obras):
    """
    Retorna lista con representantes unicos para cada medio que se usa en una lista de obras
    """
    first = lt.firstElement(obras)
    if first['Medium'] == '':
        first['Medium'] = "Ninguno"
    medios = lt.newList("ARRAY_LIST",cmpfunction=compareMediumD)
    lt.addLast(medios,first)
    for obra in lt.iterator(obras):
        i = 0
        if obra['Medium'] == '':
            obra['Medium'] = "Ninguno"
        for medio in lt.iterator(medios):
            if (str(obra['Medium']) == str(medio['Medium'])):
                i += 1
        if (i == 0):
            lt.addLast(medios,obra)
    return medios
```

En ambas funciones hay ciclos que iteran sobre las obras del artista y los medios de tales obras dadas por la función artworksbyArtistD() de antes. También hay algunos mergesort sobre algunas listas pequeñas. En particular podemos decir que la complejidad de artworksbyMediumD() no pasa de O(n log(n)), pues los ciclos operan sobre datos recortados (solo las obras de un artista particular aunque asumimos que son n y solo los medios que usan tales obras, asumimos son log(n)).

Las funciones contarMediosD() y medioMaxD() son muy sencillas:

```
def contarMediosD(obras):
    """
    Cuenta la cantidad de medios que se usan en una lista de obras
    """
    medios = listaMediosD(obras)
    num = lt.size(medios)
    return num
```

```
def medioMaxD(obras):
    """
    """
    medio = lt.firstElement(obras)
    return medio['Medium']
```

Así, podemos concluir que el requerimiento 3 tiene complejidad O(n log(n)).

Requerimiento 4 (Nicolas):

```
elif int(inputs[0]) == 5:
        start_time = time.process_time()
        tamañoNacionalidades(catalog)
        stop_time = time.process_time()
        elapsed_time_mseg = (stop_time - start_time)*1000
        print ("Tiempo transcurrido: " + str(elapsed_time_mseg))
```

Figura 11 – Requerimiento 4

```
def tamañoNacionalidades(catalog):
    """
    """
    top = controller.topNacionality(catalog)
    first = lt.firstElement(top)
    count = 0
    print(("-" * 15) + "Top Nacionalidades con más obras obras" + ("-" * 15) +
"\n")
    for x in range(1, 11):
        nac = lt.getElement(top, x)
        count += 1
        print(str(count) + ". " + str(nac["nacionalidad"]) + ": " +
str(lt.size(nac["obras"]))+ "\n")
    top_1 = mp.get(catalog["nationality"], first["nacionalidad"])
    valor = me.getValue(top_1)
    obras = valor["obras"]
    first_three = controller.firstThreeD(obras)
    last_three = controller.lastThreeD(obras)
    print("-" * 84)
    print(("-" * 21) + "Estos son las 3 primeras Obras encontradas" + ("-" *
21) + "\n")
    printArtworkData(first_three)
    print("-" * 84)
    print(("-" * 22) + "Estos son las 3 ultimas Obras encontradas" + ("-" *
21) + "\n")
    printArtworkData(last_three)
```

Figura 12 – tamañoNacionalidades()

Una vez que inicia el requerimiento 4 se llama a la función tamañoNacionalidades(), la cual empieza llamando a la función topNationality() el cual recorre cada una de las parejas llave valor del catalogo “nacionality” luego agrega a una lista nueva llamada “top”, finalmente los sortea con merge sort, que es O(n log n), y devuelve la lista “top” ordenada. Pero debido al recorrido por las llaves es O(n).

```
def topNacionality(catalog):
    """
    Organiza el Top de Nacionalidades con más obras y tambien organiza
    alfabeticamente las obras de una nacionalidad.
    """
    keys = mp.keySet(catalog["nationality"])
    top = lt.newList()
```

```
    for key in lt.iterator(keys):
        entry = mp.get(catalog["nationality"], key)
        nac = me.getValue(entry)
        lt.addLast(top, nac)
    ms.sort(top, cmpfunction=compareSizes)
    return top
```
Figura 13 – model.topNacionality()

Esto en conjunto con las demás funciones en la función tamañoNacionalidades() hace que el requerimiento 4 sea de O(n).

Requerimiento 5:

En el view.py el requerimiento 5 se ejecuta cuando el usuario escoge la opción 6 del menú:

```
elif int(inputs[0]) == 6:
    department = input("Ingrese el nombre del departamento del museo: ")
    requerimiento5(catalog,department)
```

Veamos la complejidad de la función requerimiento5:

```
def requerimiento5(catalog,department):
    dep = controller.artworksbyDepartmentD(catalog,department)
    peso = lt.removeLast(dep)
    costo = lt.removeLast(dep)
    tamano = lt.removeLast(dep)
    antiguas = controller.masAntiguasD(catalog,department)
    costosas = controller.masCostosasD(catalog,department)
    print('Total de obras para transportar: ' + str(tamano))
    print("-" * 50)
    print('Costo total estimado de transportar las obras (USD): ' + str(costo))
    print("-" * 50)
    print('Peso total estimado de las obras (kg): ' + str(peso))
    print("-" * 50)
    print('Las 5 obras más antiguas a transportar son: ')
    print("-" * 50)
    printArtworkData_Req5(antiguas)
    print("-" * 50)
    print('Las 5 obras más costosas a transportar son: ')
    print("-" * 50)
    printArtworkData_Req5(costosas)
```

Veamos complejidad de las funciones artworksbyDepartmentD(), masAntiguas(), masCostosas().

La función artworksbyDepartmentD() hace lo siguiente:

```
def artworksbyDepartmentD(catalog,department):
    """
    Dado un departamento retorna una lista con el total de sus obras, su costo total de transporte y su peso total.
    """
    dep = mp.get(catalog['department'],department)
    if dep:
        costo = me.getValue(dep)['costo']
        peso = me.getValue(dep)['peso']
        obras = me.getValue(dep)['obras']
    resultado = lt.newList("ARRAY_LIST")
    lt.addLast(resultado,lt.size(obras))
    lt.addLast(resultado,int(costo))
    lt.addLast(resultado,int(peso))
    return resultado
```

En particular, solo usa consultas sobre maps y operaciones sencillas de listas. En efecto, artworksbyDepartmentD() tiene complejidad O(1).

La función masAntiguas() hace lo siguiente:

```
def masAntiguasD(catalog,department):
    """
    Retorna una lista con las 5 obras mas antiguas de las obras del department.
    """
    dep = mp.get(catalog['depAntiguas'],department)
    if dep:
        obras = me.getValue(dep)['obras']
    ms.sort(obras,ordAscArtwDateD)
    antiguas = lt.newList()
    i = 0
    while i < 5:
        o = lt.removeFirst(obras)
        lt.addLast(antiguas,o)
        i += 1
    return antiguas
```

Esta función usa operaciones de maps y operaciones sencillas de listas (el removeFirst se hace sobre una lista “SINGLE_LINKED” así que tiene complejidad O(1)). La operación más compleja es, por tanto, el mergesort que se hace sobre obras. En este punto tales obras son una lista pequeña, recortada preliminarmente para solo contener obras antiguas, por lo que el merge aunque tiene complejidad O(n log(n)), este n será pequeño. Podemos entonces decir, sin problemas, que la complejidad de masAntiguas() es de O(n log(n)).
La función masCostosas() es muy parecida a masAntiguas():

```
def masCostosasD(catalog,department):
    """
    Retorna una lista con las 5 obras mas costosas de las obras del department.
    """
    dep = mp.get(catalog['depCosto'],department)
    if dep:
        obras = me.getValue(dep)['obras']
    ms.sort(obras,ordAscArtwCostD)
    costosas = lt.newList()
    i = 0
    while i < 5:
        o = lt.removeLast(obras)
        lt.addLast(costosas,o)
        i += 1
    return costosas
```

En este caso el removeLast se ejecuta sobre una “ARRAY_LIST” y el merge se hace sobre una lista recortada preliminarmente de solo obras costosas. Así, la complejidad es la misma que en el caso de masAntiguas(). Esto es, la complejidad de masCostosas() es de O(n log(n)).

En definitiva, podemos decir que la complejidad del requerimiento 5 es de O(n log(n)).

Pruebas de rendimiento

Datos en milisegundos.

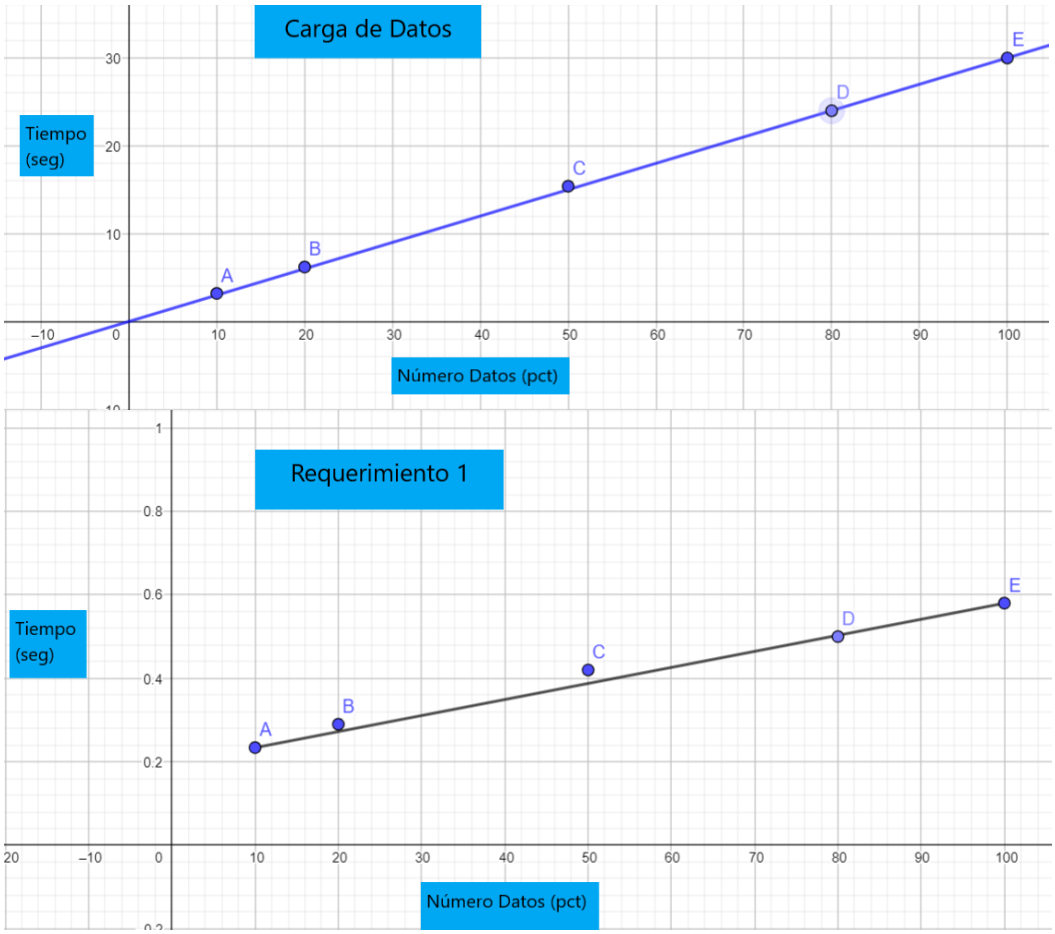
Tamaño de Datos	Carga de Datos	Req. 1	Req. 2	Req. 3	Req. 4	Req 5.
-10pct	3203.125	234.375	62.5	31.25	15.625	31.25
-20pct	6218.75	296.875	62.5	93.75	15.625	93.75
-50pct	15406.25	421.875	62.5	312.5	46.875	640.625
-80pct	23906.25	500.0	78.125	687.5	46.875	1468.75
Large	29875.0	578.125	93.75	1015.625	46.875	2375.0

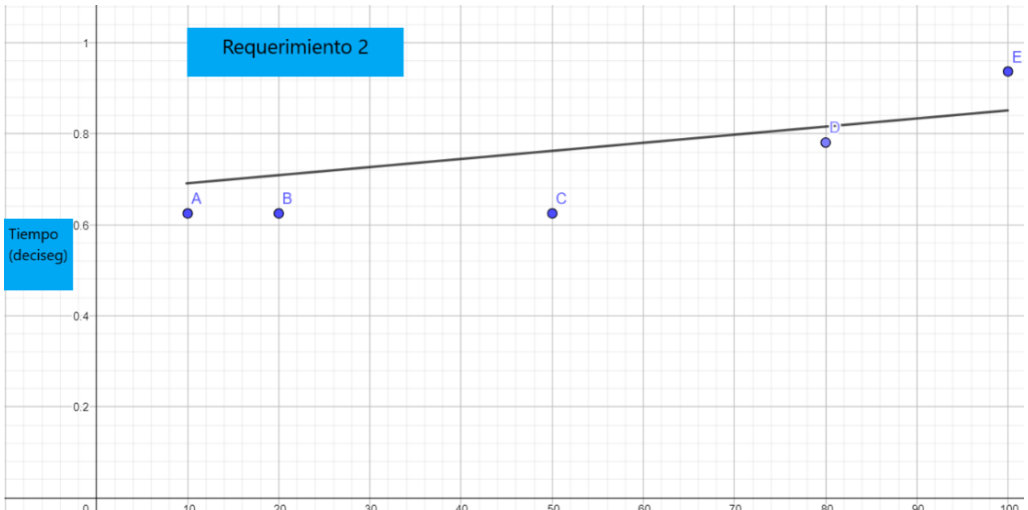
Esto constituye una mejora sustancial con respecto al rendimiento del reto1 cuyos tiempos fueron:

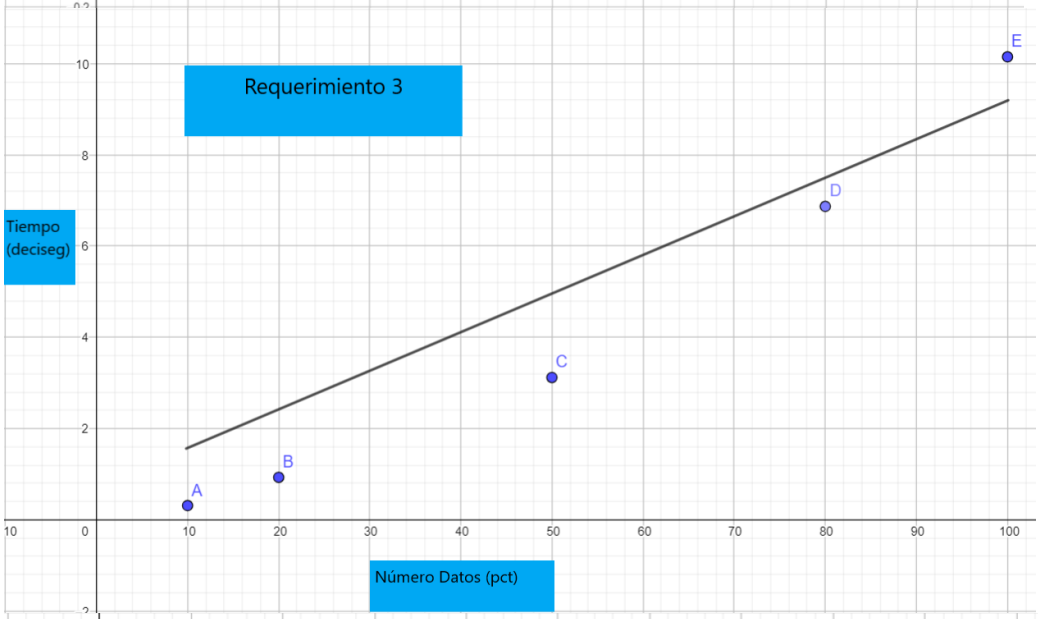
Datos en milisegundos.

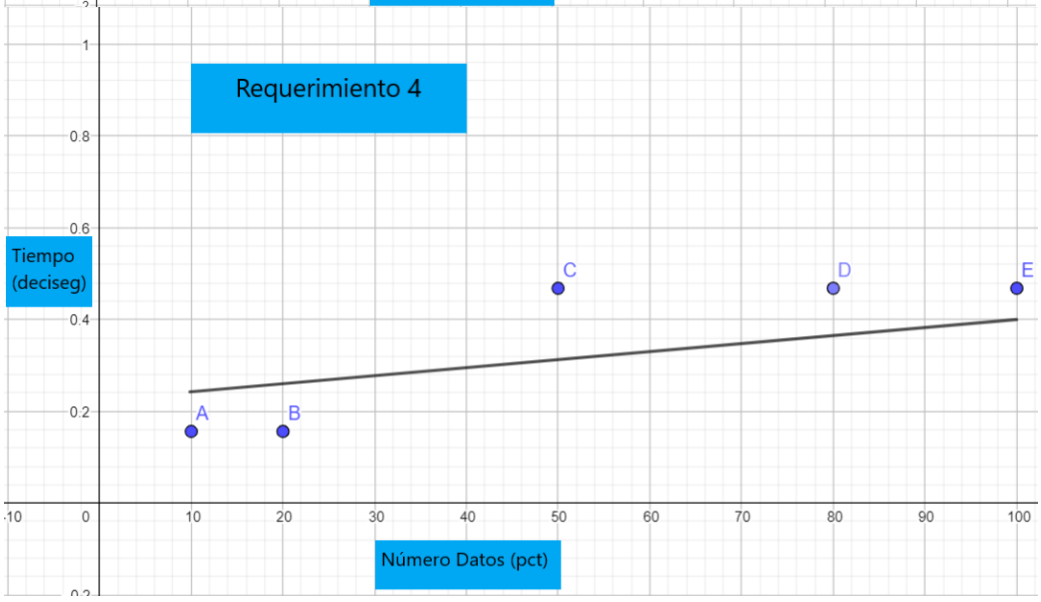
Tamaño de Datos	Carga de Datos	Req. 1	Req. 2	Req. 3	Req. 4	Req 5.
-10pct	43873.51	5601.40	74999.26	3561.70	26934.91	36254.41
-20pct	102715.95	9976.10	298371	14207	101192.36	140737.40
-50pct	310026.015	19562.22	947765	87791.8	567074.65	825329.32
Large	693926.50	30043.71	2300472.88	334712.47	2093100.97	1850678.43

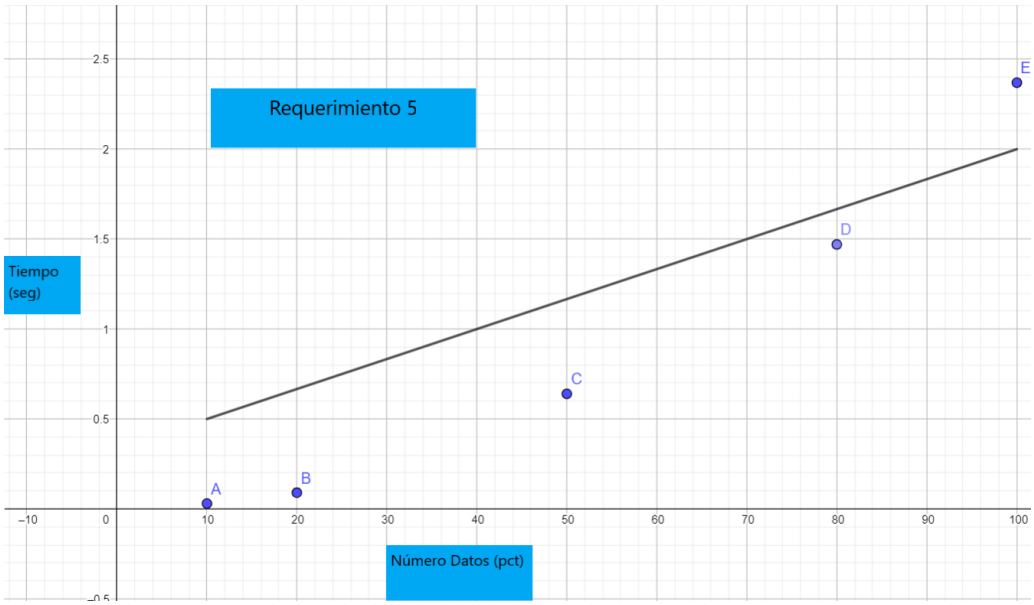
Gráficas




Requerimiento 2
Tiempo (deciseg)
Número Datos (pct)
A
B
C
D
E


Requerimiento 3
Tiempo (deciseg)
Número Datos (pct)
A
B
C
D
E


Requerimiento 4
Tiempo (deciseg)
Número Datos (pct)
A
B
C
D
E


Requerimiento 5
Tiempo (seg)
Número Datos (pct)
A
B
C
D
E
2.5
2
1.5
1
0.5
−10
0
10
20
30
40
50
60
70
80
90
100