Reto No. 1: Curando y Explorando el MoMa

Documento de Análisis:

Participantes del Grupo:

Santiago Gustavo Ayala Ciendua s.ayalac@unaindes.edu.co 202110734 -> Requerimiento 3

Nicolas Yesid Rivera Lesmes ny.rivera@uniandes.edu.co 2021166756 -> Requerimiento 4

Evaluar complejidad:

Requerimiento 1: O (1)

Esta función no realiza ningún ciclo n ya que solo tiene ciclos finitos muy pequeños y la carga de datos es realizada anteriormente, esto se ve en la siguiente foto:

```
def getLastArtist(catalog):
    """
    Retorna la lista con los últimos tres artistas de la lista de
    artistas.
    """
    artistas = catalog['artist']
    lastartist = lt.newList("ARRAY_LIST",cmpfunction=None)
    num = lt.size(catalog["artist"])
    i = 0
    while i < 3:
        ultimos = num - i
        book = lt.getElement(artistas, ultimos)
        lt.addLast(lastartist,book)
        i += 1
```

Como se puede ver solo existe un *while* y es finito de 0 a 3 por lo que no hay ciclos y la complejidad sería de O (1)

Requerimiento 2: O (n)

El requerimiento dos usa en su mayoría ciclos limitados o como máximo un ciclo (n), no posee funciones que tengan ciclos dentro de ciclos. Y su mayor complejidad es O (n) como se ve a continuación:

```
def loadObras(catalog):
    """
    Carga las obras del archivo. Se carga el CSV en una variable, posteriormente
    se lee el archivo y se cicla para añadir la obra a una lista con la llamada a
    la función addObras..
    """
    obrasfile = cf.data_dir + 'MOMA/Artworks-utf8-large.csv'
    input_file = csv.DictReader(open(obrasfile, encoding='utf-8'))
    for obra in input_file:
        model.addObras(catalog, obra)
```

O con las funciones de ordenamiento.

Requerimiento 3: O (n^3)

A pesar de que durante la mayoría del requerimiento se utilizaron solo dos bucles y por ende un O (n^2), existe una función que llega a los tres bucles, que será mostrada a continuación:

En esta función en la primera parte podemos ver como se utilizan tres bucles, uno dentro de otro, y aunque estén bajo un condicionamiento del *if* como lo que se tiene en cuenta es el peor caso, entonces la complejidad termina siendo O (n^3)

Requerimiento 4:

Requerimiento 5: O (n^3)

En el requerimiento la mayoría de las funciones tiene una complejidad de O (n), como se ve a continuación:

```
def TamañosObras(ListaDepto):

    for obra in lt.iterator(ListaDepto):

        if obra["diametro"] != "":
            vard = float(obra["diametro"])
            vard1 = (vard/2)/100
            obra["diametro"] = vard1

        if obra["altura"] != "":
            vara = float(obra["altura"])
            vara1 = vara/100
            obra["altura"] = vara1


        if obra["ancho"] != "":
            varc = float(obra["ancho"])
            varc1 = varc/100
            obra["ancho"] = varc1

        if obra["largo"] != "":
            varl = float(obra["largo"])
            varl1 = varl/100
            obra["largo"] = varl1

    return ListaDepto
```

Sin embargo, hay una función que rompe este esquema y tiene tres ciclos, uno dentro de otro, como se ve a continuación:

Y ya que lo que se evalúa en la complejidad es el peor caso, entonces la complejidad de este requerimiento termina siendo de O (n^3).

Pruebas de Velocidad:

Ambientes de pruebas

	Máquina 1	Máquina 2
Procesadores	Intel® Core™ i5-9300H CPU @ 2.4GHz	Intel® Core™ i5-8250U CPU @ 3.7GHz
Memoria RAM (GB)	8 GB	8 GB
Sistema Operativo	Windows 10 Pro-64 bits	Windows 10 Pro-64 bits

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Maquina 1

Resultados (3 toma de datos)

Tamaño de la muestra	Req 1 Tiempo (mseg)	Req 2 Tiempo (mseg)	Req 3 Tiempo (mseg)	Req 4 Tiempo (mseg)	Req 5 Tiempo (mseg)
DATOS SMALL	0	15.625	0.0		328.125
DATOS 5pct	15.625	250.0	15.625		7906.25
DATOS 10 pct	15.625	515.625	62.5		20296.875

DATOS 20 pct	15.625	1015.625	187.5		51093.75
DATOS 30 pct	15.625	1562.5	375.0		85406.25
DATOS 50 pct	15.625	2531.25	906.25		172421.875
DATOS 80 pct	0.0	4140.625	2203.125		324062.5
DATOS LARGE	15.625	5078.125	3343.75		431859.375

Maquina 2

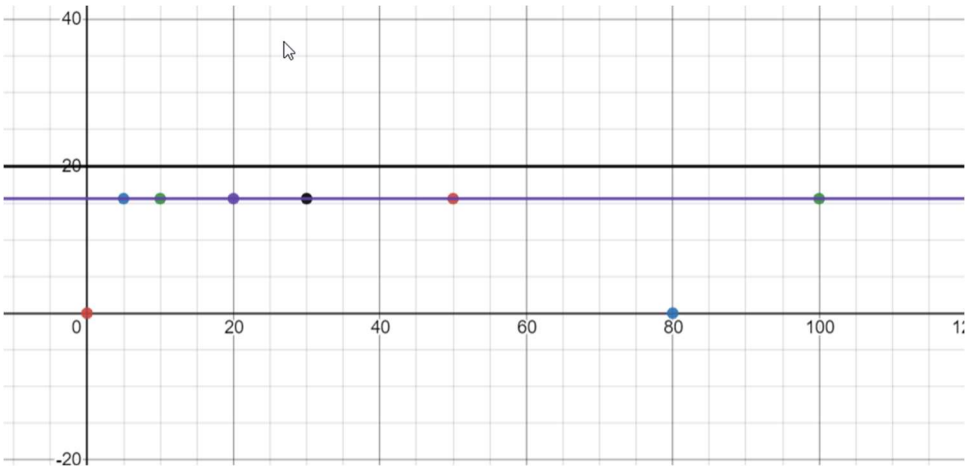
Resultados (3 tomas de datos).

El req 2 se realizó con el algoritmo de ordenamiento Merge Sort y con el TAD Array List

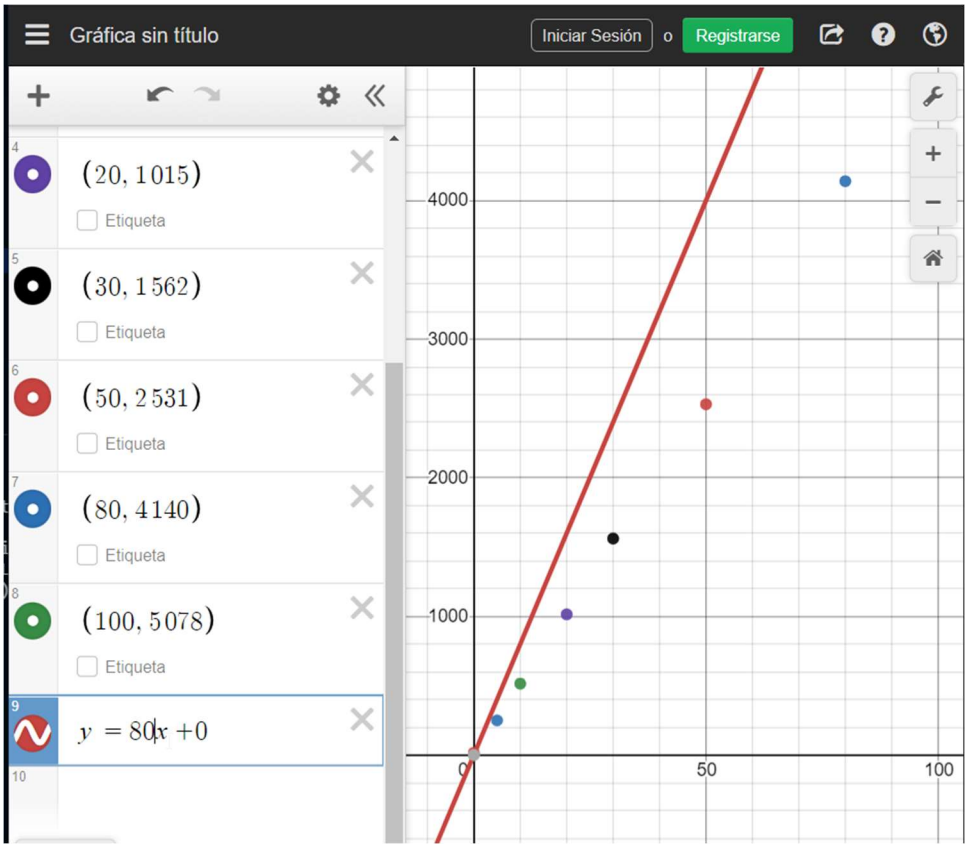
Tamaño de la muestra	Req 1 Tiempo (mseg)	Req 2 Tiempo (mseg)	Req 3 Tiempo (mseg)	Req 4 Tiempo (mseg)	Req 5 Tiempo (mseg)
DATOS SMALL	0.0	31.25	0.0		546.875
DATOS 5pct	0.0	265.625	31.25		7843.75
DATOS 10pct	15.625	515.625	78.125		19328.125
DATOS 20 pct	15.625	1046.875	250.0		53078.125
DATOS 30pct	15.625	1843.75	500.0		81796.875
DATOS 50pct	15.625	2859.375	1031.25		185937.5
LARGE	15.625	5609.375	4359.375		486659.375

Gráficas Generales:

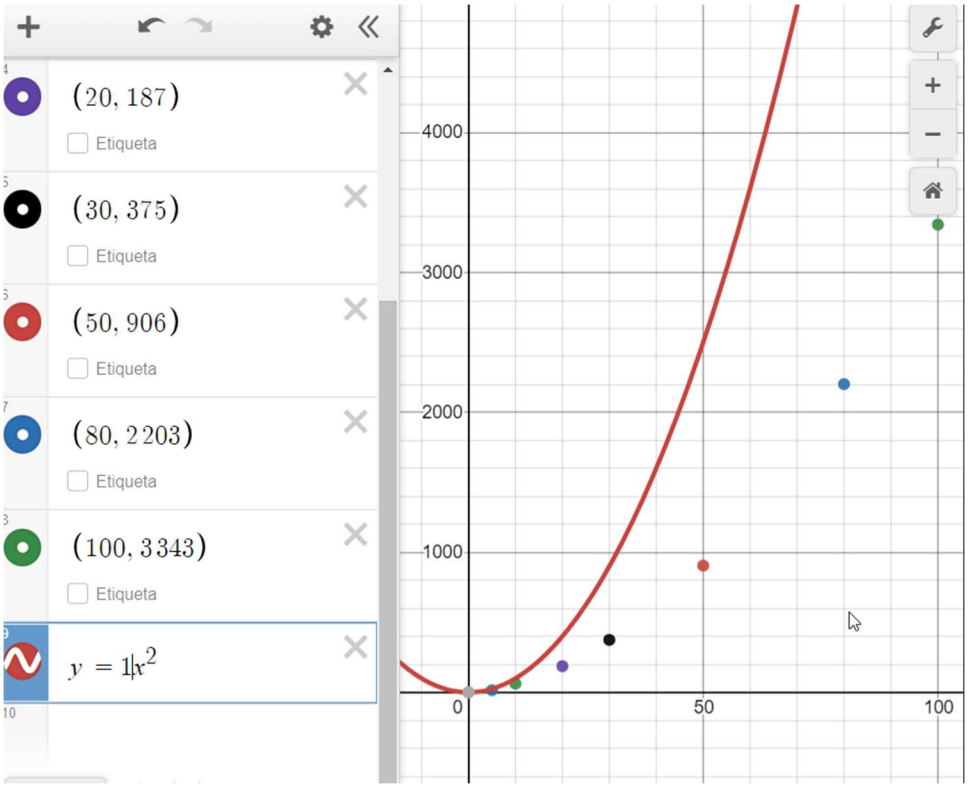
Req1:


40
20
0
20
40
60
80
100
-20

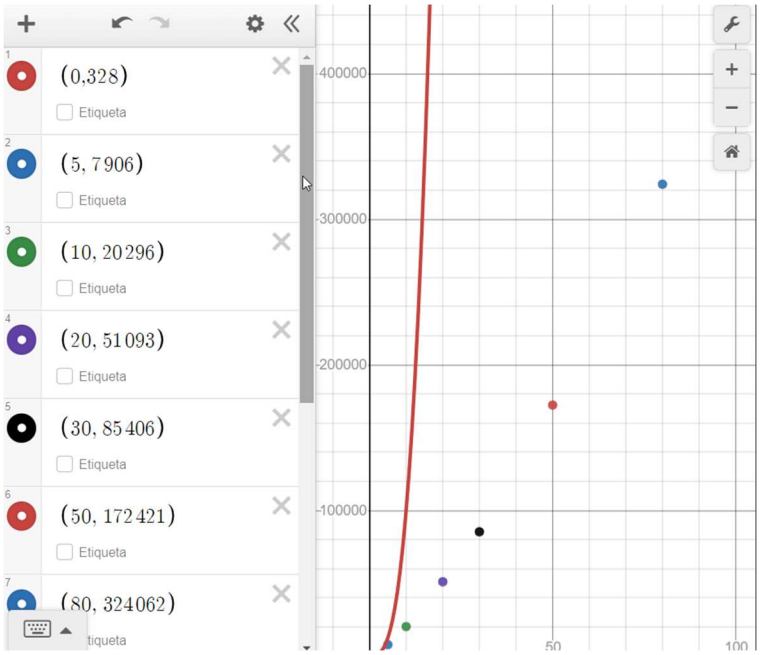
Req2:


Gráfica sin título
Iniciar Sesión
o
Registrarse
(20, 1015)
Etiqueta
(30, 1562)
Etiqueta
(50, 2531)
Etiqueta
(80, 4140)
Etiqueta
(100, 5078)
Etiqueta
y = 80x + 0
4000
3000
2000
1000
0
50
100

Req 3:


(20, 187)
Etiqueta
(30, 375)
Etiqueta
(50, 906)
Etiqueta
(80, 2 203)
Etiqueta
(100, 3 343)
Etiqueta
y = 1x^2
4000
3000
2000
1000
0
50
100

Req 5:


(0,328)
Etiqueta
(5, 7906)
Etiqueta
(10, 20296)
Etiqueta
(20, 51093)
Etiqueta
(30, 85406)
Etiqueta
(50, 172421)
Etiqueta
(80, 324062)
400000
300000
200000
100000
50
100

Preguntas de análisis

1. ¿El comportamiento de los algoritmos es acorde a lo enunciado teóricamente?

En su mayoría se podría decir que si, los algoritmos quedan bajo las funciones que se mencionaron, existe una mínima discrepancia en el requerimiento 3 donde los datos quedan bien debajo de una función cuadrática cuando se dijo que esta era de complejidad O (n^3). Esto ya que como se mencionó, ese tercer ciclo está bajo una condicional, sin embargo, como lo que se toma en cuenta es el peor caso entonces decidimos dejarlo con esa complejidad. De resto se puede ver claramente con las gráficas que la complejidad anunciada fue de lo más adecuado.

2. ¿Existe alguna diferencia entre los resultados obtenidos al ejecutar las pruebas en diferentes máquinas?

Hubo una mínima discrepancia en los números esto debido a que son maquinas diferentes, pero lo importante que son los patrones se mantuvieron muy similares.

3. De existir diferencias, ¿a qué creen que se deben?

Como se mencionó debido a las características intrínsecas de la maquina

4. ¿Cuál Estructura de Datos funciona mejor si solo se tiene en cuenta los tiempos de ejecución de los algoritmos?

A pesar de no haber hecho una tabla, probamos que el ARRAY_LIST suele funcionar más rápido y que el algoritmo de ordenamiento MERGE que fuel el que más usamos también era el más rápido.

5. Teniendo en cuenta las pruebas de tiempo de ejecución por todos los algoritmos de ordenamiento estudiados (iterativos y recursivos), proponga un ranking de los mismo de mayor eficiencia a menor eficiencia en tiempo para ordenar la mayor cantidad de obras de arte.

No hicimos un ranking, ya que eso era del laboratorio pasado, pero si medimos y creemos que el mejor algoritmo de ordenamiento es MERGE por su velocidad.

6. Conclusiones:

Nos parece útiles los TAD, estructuras de datos y algoritmos de ordenamiento, aunque costo un tiempo adaptarse a ellos podemos ver sus puntos fuertes y utilidades. Sin embargo, a veces recaemos en lo simple y no los utilizamos como paso en el requerimiento 3. Queremos resaltar a las ARRAY_LIST y algoritmo MERGE por ser los más veloces y simples, además de ser lo que más utilizamos.