Reto No. 2: Curando y Explorando el MoMa - RESURRECTION

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 (n)

Esta función realiza solo un ciclo significativo de valor n, este ocurre al crear el mapa de las fechas que va a ser utilizado más adelante. Al solo haber ciclos de n y no ciclos dentro de ciclos, la complejidad de este algoritmo es de O(n). El ciclo significativo se muestra a continuación

```
def fechasmap(catalog,inicio,fin):

    fechas = mp.newMap(1000,
                       maptype = "CHAINING",
                       loadfactor = 4,
                       comparefunction = None)


    contador = 0

    for artist in lt.iterator(catalog["artistas"]):
        año = int(artist["BeginDate"])
        if (año>=inicio) and (año <= fin) and (año
            if mp.contains(fechas,año):
                entry = mp.get(fechas,año)
                value = me.getValue(entry)
                lt.addFirst(value,artist)
```

Como se puede ver solo existe un ciclo por lo que la complejidad sería de O (n)

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:

O con las funciones de ordenamiento.

Requerimiento 3: O (n)

El requerimiento en general suele tener ciclos limitados y cortos, que no llegarían a contar como O(n), como por ejemplo este for en las llaves de un map

```
llaves = mp.keySet(Medium_Mejorado)
lista_tamaños = lt.newList("ARRAY_LIST")

for llave in lt.iterator(llaves):
    lista_ob = mp.get(Medium_Mejorado, llave)["value"]
    tamaño = lt.size(lista_ob)
    dict_ayuda2 = {"tecnica": llave, "tamaño": tamaño}
    lt.addLast(lista_tamaños, dict_ayuda2)
```

O también tiene un mini merge sort, sin embargo la razón por la que definitivamente es de complejidad O(n) es porque se crea un nuevo mapa en la función y para crear este mapa toca ciclar las n obras para poder añadirlas como se ve a continuación:

```
Medium_Mejorado = mp.newMap(200,
                            maptype = "PROBING",
                            loadfactor = 0.8,
                            comparefunction = None)

cantidad_obras = 0

for obras in lt.iterator(catalog["obras"]):
    if "," in obras["ConstituentID"]:
        variable = obras["ConstituentID"]
        lista_codigos = variable.split()
        for codigos in lista_codigos:
            if codigo == codigos:
                if mp.contains(Medium_Mejorado, obras["Medium"]):
                    entrada = mp.get(Medium_Mejorado, obras["Medium"]
                    valor = me.getValue(entrada)
                    dict_ayuda = {"titulo": obras["Title"], "fecha":
                    lt.addLast(valor, dict_ayuda)
                    me.setValue(entrada, valor)
                    cantidad_obras += 1
```

Y a pesar de que dentro de esa función existe otro for, este es limitado y mínimo ya que es de un elemento como los codigos de los artistas de una obra. Además las funciones del mapa como get son

O(1) en probing entonces tampoco afecta el cálculo de complejidad. Por ende la complejidad del requerimiento 3 es O(n) por el ciclado de obras al crear el map.

Requerimiento 4:

Requerimiento 5: O (n)

Este requerimiento posee dos grandes ciclos, sin embargo uno de estos ciclos está limitado a las obras de un departamento por lo que no afecta mucho. Pero el otro ciclo si es un ciclo de n ya que recorre los elementos de la lista de obras para poder crear un nuevo map, como se ve a continuación:

```
Departamentos = mp.newMap(200,
                          maptype = "PROBING",
                          loadfactor = 0.8,
                          comparefunction = None)


for obras in lt.iterator(catalog["obras"]):

    if mp.contains(Departamentos, obras["Department"]):
        entrada = mp.get(Departamentos, obras["Department"])
        valor = me.getValue(entrada)
        dict_ayuda = {"titulo": obras["Title"], "artistaid": obras[
        lt.addLast(valor, dict_ayuda)
        me.setValue(entrada, valor)
    else:
```

Entonces al haber solo un ciclado de n la complejidad termina siendo O(n)

Requerimiento 6: O (n)

Muy parecido al requerimiento 5, este posee varios ciclos limitados y minimos, pero a la vez posee un ciclado grande para hacer un map y un ciclado mediano para otro map más pequeño. Pero como estos son ciclos separados seria 2n pero en complejidad O eso se reduce a O(n), el ciclado grande del map se muestra a continuación.

```
Madre = mp.newMap(200,
                  maptype = "PROBING",
                  loadfactor = 0.8,
                  comparefunction = None)

for obras in lt.iterator(catalog["obras"]):

    if mp.contains(Madre, obras["Date"]):
        entrada = mp.get(Madre, obras["Date"])
        valor = me.getValue(entrada)
        lt.addLast(valor, obras)
        me.setValue(entrada, valor)
    else:
```

Entonces como se puede verificar la complejidad es O(n)

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)	Req 6 Tiempo (mseg)
DATOS SMALL	15.625		0		31.25	31.5
DATOS 5pct	35.5		15.625		281.25	265.625
DATOS 10 pct	62.5		31.5		531.25	531.25

DATOS 20 pct	93.75		46.875		1218.75	1078.125
DATOS 30 pct	125		60.25		1640.625	1750
DATOS 50 pct	168.25		93.75		3187.5	3000
DATOS 80 pct	234.35		171.85		5000	5156
DATOS LARGE	327.625		359.375		6031.25	6268.75

Tamaño de la muestra	Req 1 Memoria	Req 2 Memoria	Req 3 Memoria	Req 4 Memoria	Req 5 Memoria	Req 6 Memoria
DATOS SMALL	Muy poca variabilidad para medir – No hay tiempo suficiente para que se registre un uso extensivo de la memoria					
DATOS 5pct						
DATOS 10 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 5.78 -5.85 G					
DATOS 20 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 5.86 -5.95 G					
DATOS 30 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.03 - 6.09 G					
DATOS 50 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.16 - 6.23 G					
DATOS 80 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.27 - 6.36 G					
DATOS LARGE	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.41 - 6.51 G					

Maquina 2

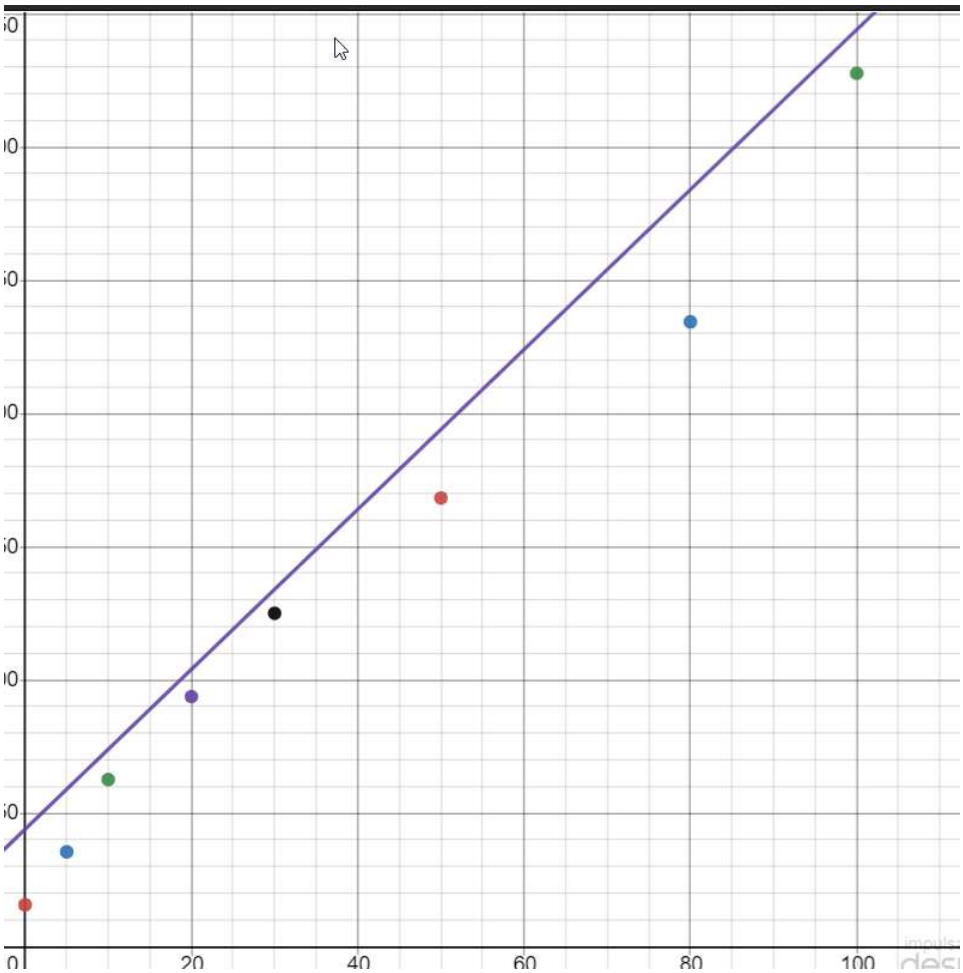
Resultados (3 tomas de datos).

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)	Req 6 Tiempo (mseg)
DATOS SMALL	15.625		0		30.25	31.25
DATOS 5pct	30.5		15		278.265	245.375
DATOS 10pct	67.25		31.25		556.375	581
DATOS 20 pct	103.275		49.875		1128.5	1108.225
DATOS 30pct	120.75		70.625		1664.5	1950.25
DATOS 50pct	165		98		3387	3000
DATOS 80pct	255.5		184.85		5138	5357.625
LARGE	349.325		394.75		6131.75	6468

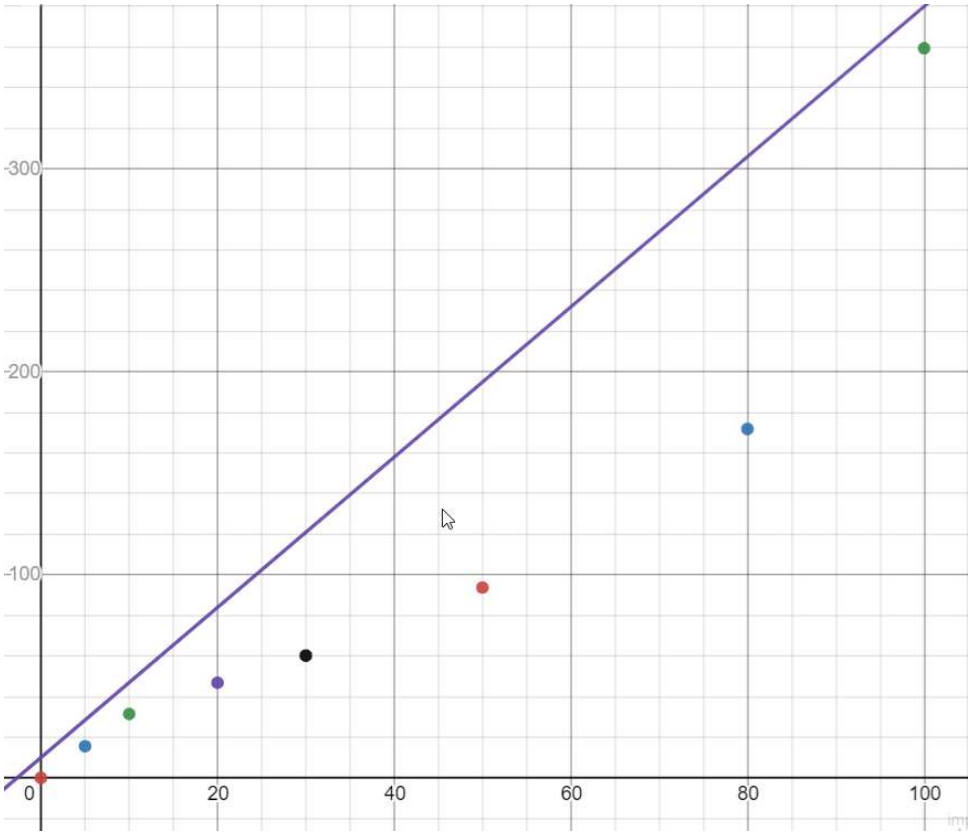
Gráficas Generales:

Req1:

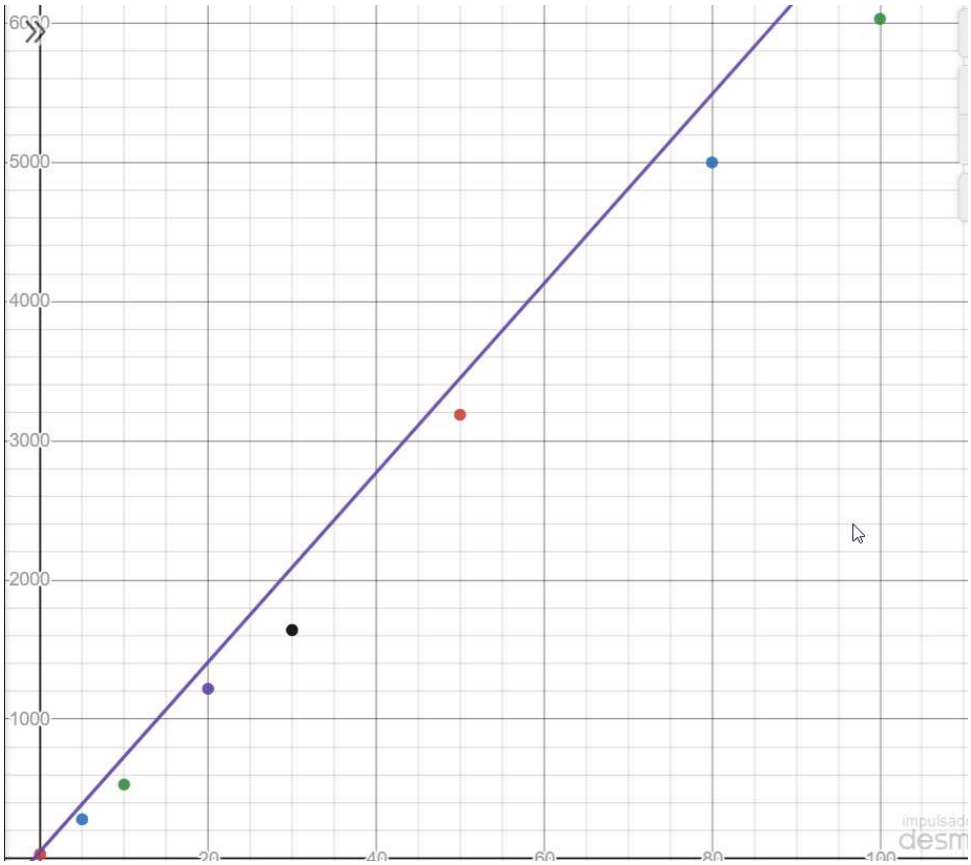


Req2:

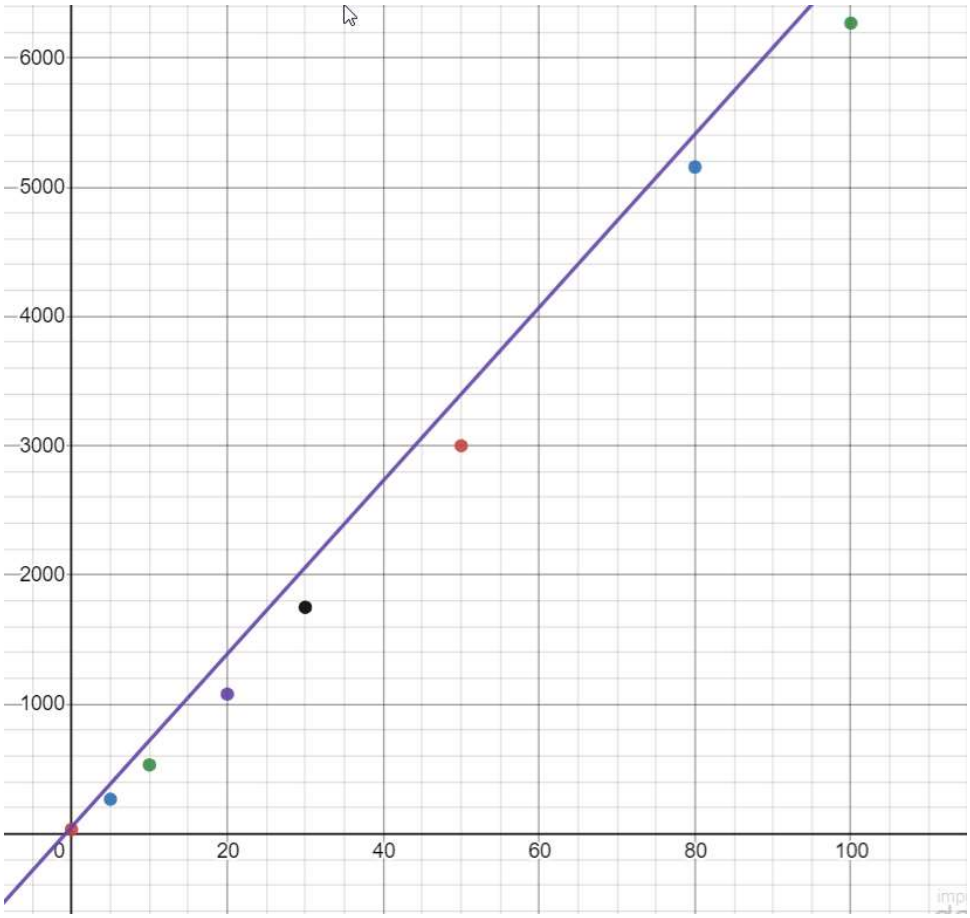
Req 3:



Req 5:



Req 6:



Comparación vs el Reto 1:

Requerimiento 1:

Sorpresivamente en este reto 2 el requerimiento 1 fue más demorado y de mayor complejidad que en el reto 1. En el reto 1 la complejidad era de O(1), pero en este es de O(n) y por ende los tiempos tambien fueron más largos en su totalidad en este reto. Inferimos que esto pasa ya que en este caso para el reuqerimiento 1 se hizo un ciclado de n obras para hacer un mapa de fechas, cosa que no se hizo en el primero, donde el mayor ciclado que habia era un while limitado. En general una sorpresa y puede que sea mejor quedarse con el codigo del reto 1

Requerimiento 2:

Requerimiento 3:

Los tiempos y complejidad para este reto fueron muy menores ya que paso de ser O(n^3) a O(n), esto debido a que en este reto con el hecho de hacer un mapa util que tenga de llaves los medium, quita muhco la necesidad de varios ciclos en las listas. Ya que se puede acceder a los valores con las llaves de las tecnicas en vez de hacer ciclados y listas extra para que queden los medios con sus respectivas obras. En general muy buena mejora y definitivamente mejor usar maps a listas en el requerimiento

Requerimiento 4:

Requerimiento 5:

En este requerimiento tyambien mejoraron mucho los tiempos y la complejidad, al igual se logro pasar de ser O(n^3) a O(n), y con la misma razon que el requerimiento 3. Ya que solo se necesito un ciclado para hacer el map de utilidad y no hubo necesidad de ciclos dentro de ciclos para hacer sublistas u ordenar datos. En general otra gran mejora y definitivamente mejor usar maps a listas en el requerimiento.

Preguntas de análisis

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

Analizando en todas menos el requerimiento 1 el rendimiento es el esperado. Se esperaba una mejora de complejidad y tiempo y se obtuvo bajando todas las complejidades alrededor de O(n), lástima que para el requerimiento 1 en el reto 1 era de O(1) y en este caso aumento

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 mecanismo de colision utilizaron?

En su gran mayoria, sino todo se utilizó probing, ya que se vio que el esapcio no fue un problema como se ve en la tabla y lo que se buscaba era una gran eficiencia en cuanto a tiempo

5. Conclusiones:

Definitivamente una gran mejora y de gran utilidad los maps, solo se necista O(n) para crearlos y ahora muchos ciclos y lineas de codigo. Sin mencionar la gran mejora de eficiencia de timepo que trae consigo. Para un futuro seria bueno corregir y si acaso realizar los maps en la carga de datos para que los requerimeintos tomen aun menor tiempo y sean de menor complejidad