Reto No. 3: UFO Sightings

Documento de Análisis:

Participantes del Grupo:

Nicolas Yesid Rivera Lesmes ny.rivera@uniandes.edu.co 2021166756 -> Requerimiento 2

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

Evaluar complejidad:

Requerimiento 1: O (log(n))

Esta función es muy simple y no realiza ningun ciclado directo, solamente realiza un merge sort y una busqueda de rangos. Ambas acciones en este caso tienen la complejidad log(n) por las caracteristicas del merge sort y la bsuqeuda del rango al ser una busqueda en un arbol binario.

```
def Avistamientos_Ciudad(cont, ciudad):

    mapa_ciudad = cont["cityIndex"]
    avistamientos = om.get(mapa_ciudad, ciudad)["value"]

    r = ms.sort(avistamientos, CmpFechaHoraInvertido)



    return avistamientos
```

Requerimiento 2: O (n)

```python
def reqdos(minimo,mayor,cont):

    rango = om.values(cont,minimo,mayor)
    num = 0
    lista = lt.newList("ARRAY_LIST")
    for i in lt.iterator(rango):
        tamaño = lt.size(i)
        num += tamaño
        for ufo in lt.iterator(i):
            lt.addLast(lista,ufo)
    lista_ord = sa.sort(lista,CmpUfoByDuration)
    return lista_ord,num

def req2fl(lista):
    contador = 0
    contador_dos = lt.size(lista) - 3
    nueva_lista = lt.newList("ARRAY_LIST")

    for i in lt.iterator(lista):
        if contador < 3:
            lt.addFirst(nueva_lista,i)
        if contador >= contador_dos:
            lt.addLast(nueva_lista,i)
        contador += 1
    return nueva_lista

def keymaxima(arbol):

    return om.maxKey(arbol)

def ufomaxima(arbol,llave):

    return om.get(arbol,llave)

def num_ufomax(lista):

    return lt.size(lista["value"])
```

En general para el req2 se construyeron distintas funciones para hacer el trabajo modular y que el código sea más limpio. Por otra parte, las funciones que realmente aumentan los tiempos del req 2 son “req2” y “req2fl” en las cuales se centrará la atención para evaluar la complejidad del requerimiento.

“req2”: En esta función se cicla a través de los valores que se encuentren en el rango de duración dada por lo que su complejidad será O(n), por otra parte, se cuenta con un ciclo anidado, pero este recorre listas internas que en general no sobrepasan los 4-5 ciclos por lo que no se tiene en cuenta para la complejidad.

“req2fl”: Esta función solo añade a una lista desde la posición 0-3 y desde N-3 hasta N por lo que solo se hacen 6 operaciones. Sin embargo, el contador sigue sumando y se hacen N comparaciones pero no se toma ninguna acción al respecto.

Requerimiento 3: O (n)

El requerimiento posee una busqeuda binaria log(n), pero tambien posee un ciclado dentro de otr que sirve para separa las obras en una lista de listas, en el peor caso esa lista de listas tendra todos los elementos entonces seria de tamaño n y hacer el doble ciclado para desglosarla por ende tambien seria O(n), la funcion se puede ver a continuacion.

```
def Ufos_Hora(lim_inf, lim_sup, cont):

    hora_inf = dt.strptime(lim_inf, "%H:%M")
    hora_sup = dt.strptime(lim_sup, "%H:%M")
    mapa_hora = cont["hourIndex"]

    valores = om.values(mapa_hora, hora_inf.time(), hora_sup.time())

    lista_horas = lt.newList("ARRAY_LIST")

    for valor in lt.iterator(valores):

        for val in lt.iterator(valor):

            lt.addLast(lista_horas, val)


    return lista_horas
```

Requerimiento 4: O (n):

En este requerimiento a pesar de que normalemente seria de caracteristicas log(n) al poseer el mismo probelma que el requerimeitno 3 del doble ciclado que equivale al numero total de elementos , su complejidad se vuelve O(n) en el peor de los casos, como se ve en la foto a continuación

```
def Ufos_Dia(lim_inf1, lim_sup1, cont):

    fecha_inf = dt.strptime(lim_inf1, "%Y-%m-%d")
    fecha_sup = dt.strptime(lim_sup1, "%Y-%m-%d")
    mapa_fecha = cont["dateIndex"]

    valores = om.values(mapa_fecha, fecha_inf.date(), fecha_sup.date())

    lista_fechas = lt.newList("ARRAY_LIST")

    for valor in lt.iterator(valores):

        for val in lt.iterator(valor):

            lt.addLast(lista_fechas, val)



    return lista_fechas
```

Requerimiento 5: O (n)

Este requerimiento es de complejidad O (n) ya que posee dos ciclados que en su peor caso serian de tamaño n. El primero es el doble ciclado para desglosar las listas, el mismo problema del requerimiento 3 y 4. Y el segundo es el ciclado por la lista desglosada para buscar las laittudes. A pesar de poseer dos ciclados que en su peor caso serian de tamaño n, no esatn uno dentro de otro por lo que su complejidad final es O (n). Esto se evidencia a continuación

```python
def Ufos_Coordenadas(inf_long, max_long, min_lat, max_lat, cont):

    mapa_lugar = cont["placeIndex"]
    #print(mapa_lugar)
    #print(inf_long, max_long)
    valores = om.values(mapa_lugar, inf_long, max_long)
    #value = om.get(mapa_lugar, inf_long)
    lista_lugares_1 = lt.newList("ARRAY_LIST")

    for valor in lt.iterator(valores):

        for val in lt.iterator(valor):

            lt.addLast(lista_lugares_1, val)

    #print(valores)
    r = ms.sort(lista_lugares_1, CmpLat)

    lista_lugares2 = lt.newList("ARRAY_LIST")

    for ufos in lt.iterator(lista_lugares_1):
        latitud = round(float(ufos["latitude"]), 2)
        if latitud <= max_lat and latitud >= min_lat:
            lt.addLast(lista_lugares2, ufos)

    return lista_lugares2
```

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 11 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	0.0	0.0	0.0	0.0	0.0	

DATOS 5pct	0.0	46.875	0.0	0.0	15.625	
DATOS 10 pct	0.0	78.125	15.625	0.0	31.25	
DATOS 20 pct	15.625	171.875	31.25	15.625	62.5	
DATOS 30 pct	15.625	421.875	62.5	15.625	171.875	
DATOS 50 pct	15.625	609.375	78.125	15.625	187.5	
DATOS 80 pct	15.625	718.75	78.125	31.25	328.125	
DATOS LARGE	31.25	1031.25	125.0	46.875	359.375	

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 Siempre se mantuvo entre los valores de 5.40G y 5.60G incluso al cambiar el tamaño de los archivos						
DATOS 5pct							
DATOS 10 pct							
DATOS 20 pct							
DATOS 30 pct							
DATOS 50 pct							
DATOS 80 pct							
DATOS LARGE							

Maquina 2

Resultados (3 tomas de datos).

El reto 3 se realizó con RBT en la totalidad de sus árboles.

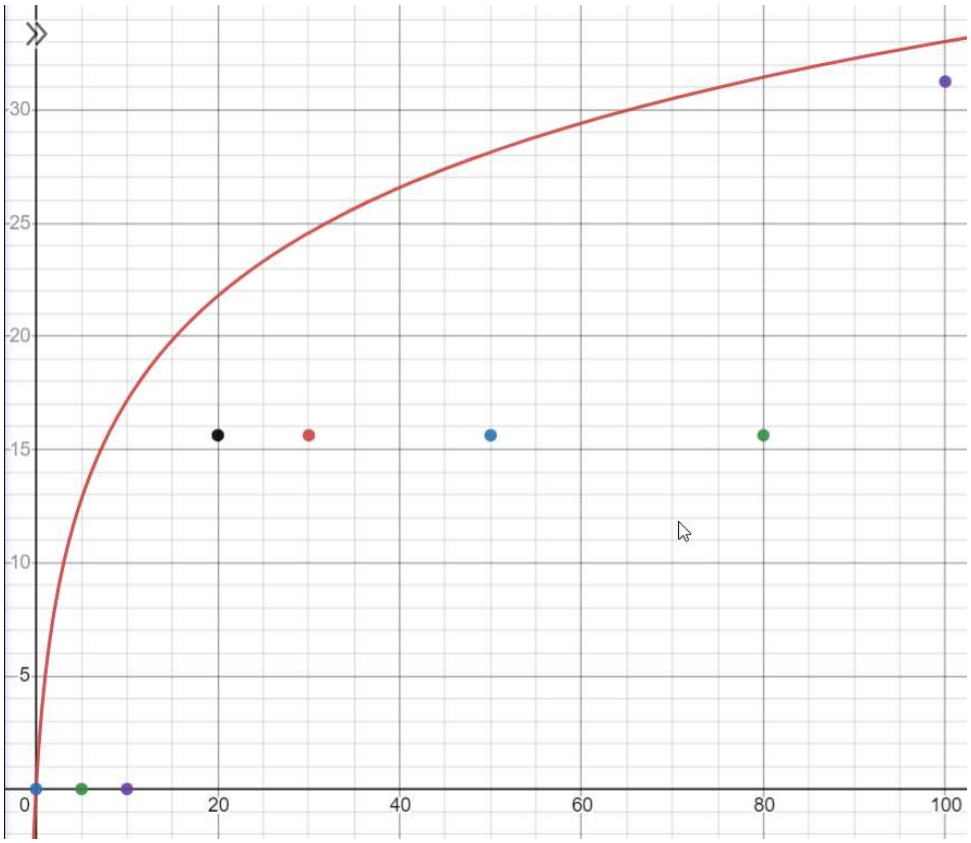
Tamaño de la muestra	Req 1 Tiempo (mseg)	Req 2 Tiempo (mseg)	Req 3 Tiempo (mseg)	Req 4 Tiempo (mseg)	Req 5 Tiempo (mseg)	Req 6 Tiempo (mseg)
DATOS SMALL	0.0 mseg	15.625 mseg	0.0 mseg	0.0 mseg	0.0 mseg	
DATOS 5pct	15.625 mseg	15.625 mseg	0.0 mseg	0.0 mseg	15.625 mseg	
DATOS 10pct	0.0 mseg	31.25 mseg	0.0 mseg	0.0 mseg	15.625 mseg	
DATOS 20 pct	0.0 mseg	62.5 mseg	15.625 mseg	15.625 mseg	46.875 mseg	
DATOS 30pct	15.625 mseg	187.5 mseg	15.625 mseg	15.625 mseg	31.25 mseg	
DATOS 50pct	0.0 mseg	171.875 mseg	31.25 mseg	15.625 mseg	109.375 mseg	
DATOS 80pct	15.625 mseg	281.25 mseg	46.875 mseg	15.625 mseg	93.75 mseg	
LARGE	15.625 mseg	343.75 mseg	31.25 mseg	15.625 mseg	93.75 mseg	

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: 5.85 - 5.86 G					
DATOS 20 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 5.95 - 6.03 G					
DATOS 30 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.09 - 6.16 G					
DATOS 50 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.23 - 6.27 G					
DATOS 80 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango:					

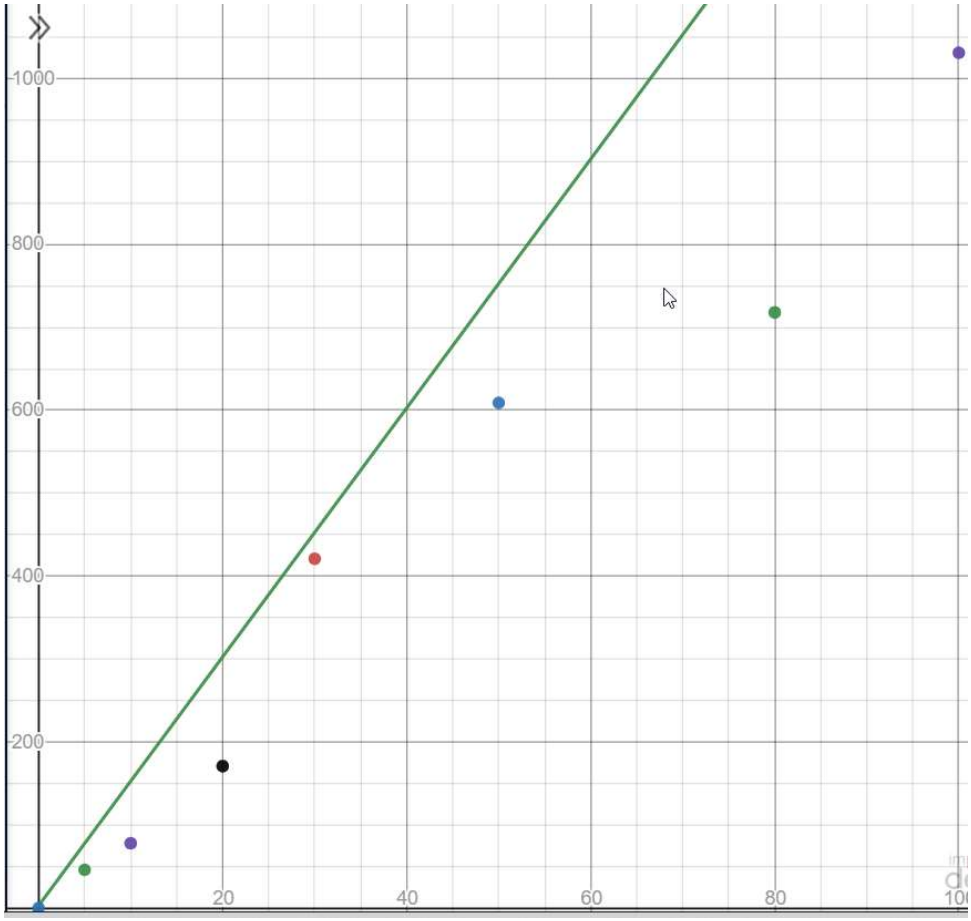
	6.36 - 6.41 G
DATOS LARGE	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.5 - 6.61 G

Gráficas Generales:

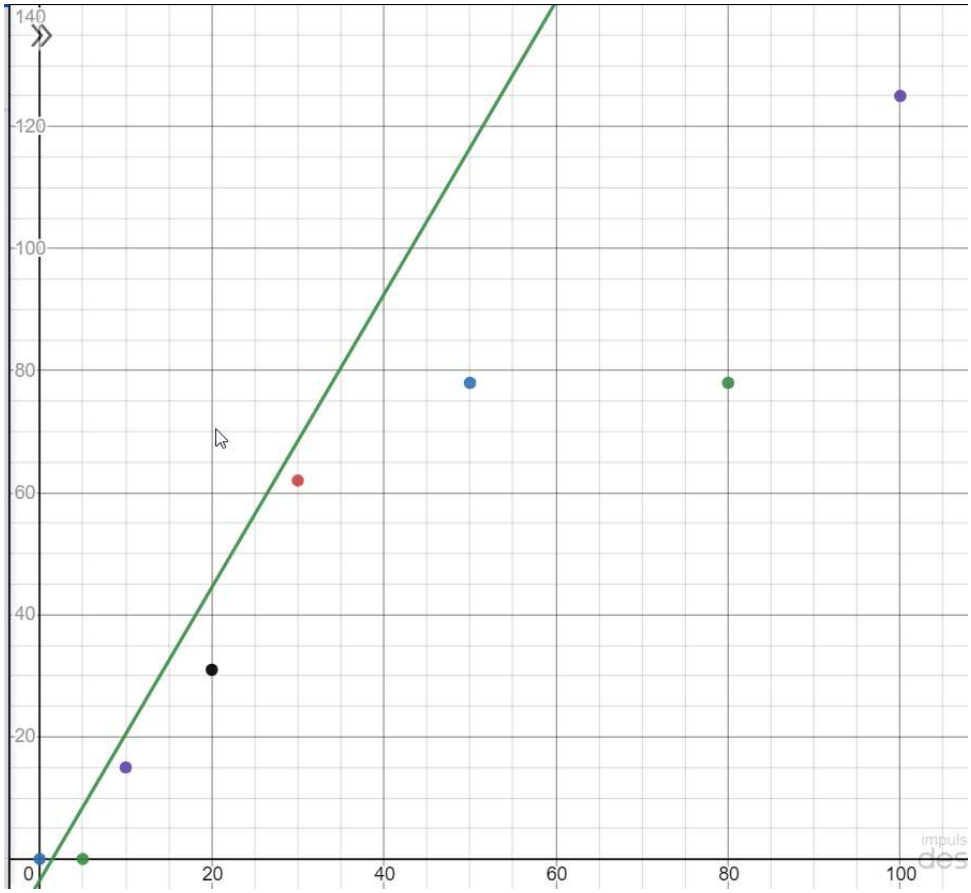
Req1:



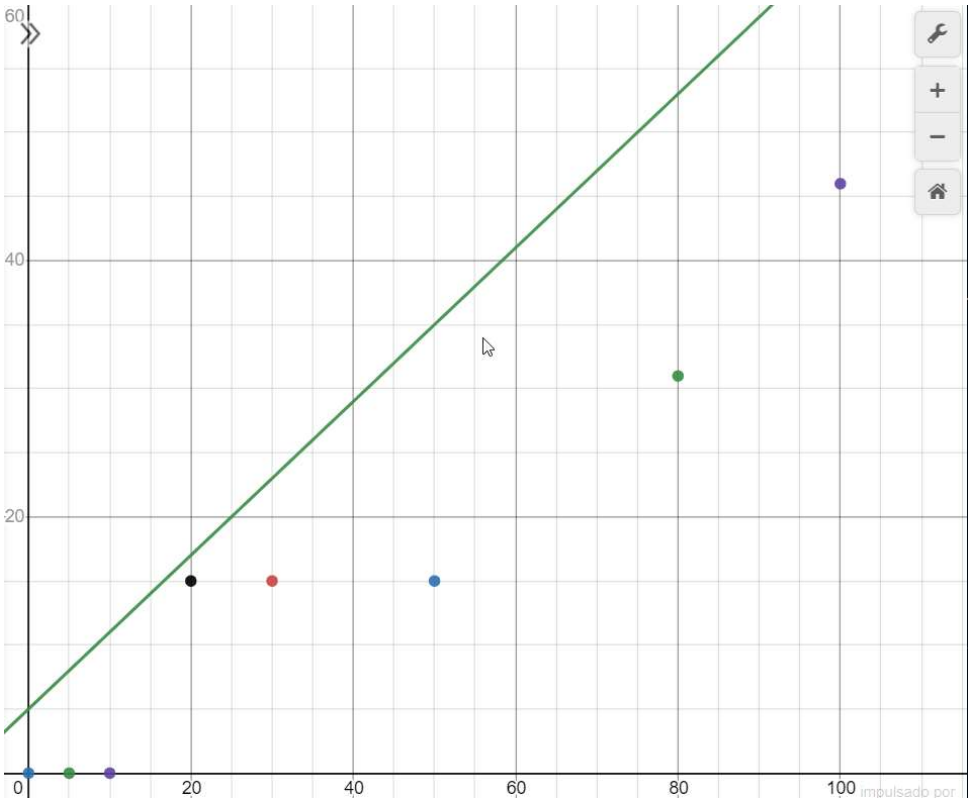
Req2:



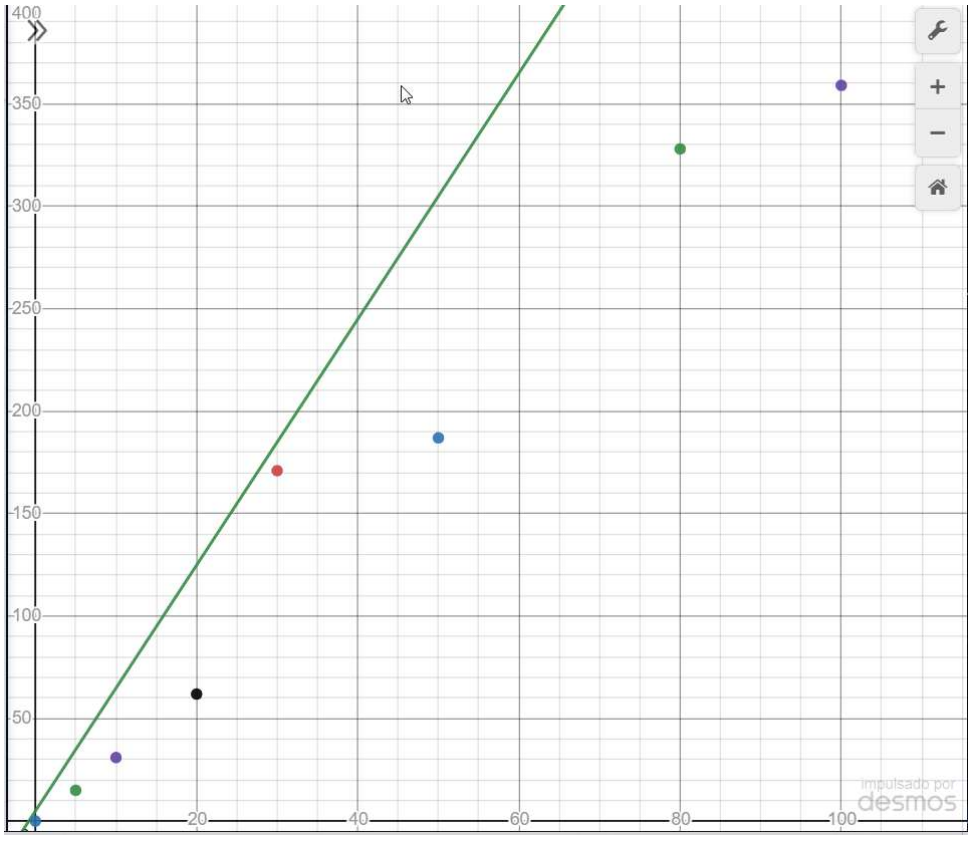
Req 3:



Req 4:


60
40
20
0
20
40
60
80
100

Req 5:


400
350
300
250
200
150
100
50
20
40
60
80
100
desmos

Preguntas de análisis

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

Analizando en todas hubo un rendimiento aceptable, en específico el requerimiento 1 se destaca al tener complejidad log(n) y si alguno está bajo lo esperado es el requerimiento 2 con el mayor tiempo de todos.

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

Hubo unas diferencias ya que en una maquina dieron unos resultados muy variados, mientras que en la otra fueron muy uniformes.

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

Esto probablemente debido al gasto de memoria ya que en un computador la memoria no subió de 5.60 mientras que en la otra se subió por encima de 6 G esto demostrando como a costo de espacio el tiempo disminuye.

4. ¿Cuál tipo de árbol utilizaron?

Utilizamos el tipo de árbol “RBT” debido a que este, aunque sea más complejo, brinda una mejor estructura de los datos haciendo que las funciones de búsqueda en el mismo tomen menos tiempo contribuyendo además con los tiempos de cada requerimiento para que la eficiencia general del reto sea la mejor posible.

5. Comparación con retos anteriores:

Sin duda alguna, respecto a los retos anteriores el tiempo de desarrollo fue menor y globalmente la complejidad para lograr los requerimientos que se indican en el reto fue baja comparada a los anteriores retos. Lo anterior lo explicamos gracias a las facilidades que brindan los árboles y a la variedad de funcionalidades que vienen incorporadas en las librerías haciendo de tareas como buscar por rangos que en un hash table, por ejemplo, se tenga que ciclar por todo el hash table para sacar los valores en el rango a simplemente con una función sacar el rango sin prácticamente demora alguna.

6. Conclusiones:

Definitivamente una gran mejora y de gran utilidad los maps, solo se necesia O(n) para crearlos y ahorra muchos ciclos y líneas de código. Sin mencionar la gran mejora de eficiencia de tiempo que trae consigo. Para un futuro se puede mejorar ese doble ciclado que estorba en los requerimientos 3-5 lo cual podría traer una gran baja de tiempo. Toca mencionar que en una maquina por cuestiones de uso no se comprometido la memoria y se vio en los tiempos más largos, pero en la otra se usó más memoria y los tiempos fueron impresionantes. Sin embargo, en las graficas se ve la complejidad y se ve que todo quedo de manera adecuada. A mejorar a futuro evitar ciclados innecesarios. En general una gran mejora y avance al respecto de los retos anteriores.