Reto 3

Documento de análisis

Est2: Juan Felipe Serrano Martinez – 201921654 – j.serrano@uniandes.edu.co

Est1: Santiago Sinisterra Arias – 202022177 – s.sinisterra@uniandes.edu.co

Análisis de complejidad de códigos:

Requerimiento 1:

```
def getlist3firtandlast(catalog,ciudad):
    resp = lt.newList()
    arbol = me.getValue(mp.get(catalog['locations'],ciudad))
    keyset = om.keySet(arbol)
    iterador = lt.iterator(keyset)
    for key in iterador:
        sighting = me.getValue(om.get(arbol, key))
        lt.addLast(resp,sighting)
    return resp
```

El requerimiento 1 usa principalmente un mapa de arboles, esto se hace para facilitar la busqueda de ciudades especificas. Con esto en mente, conseguir la lista de los avistamientos en orden de una ciudad en especifico vuelve la complejidad de este requerimiento tan solo O(N) siendo N = tamano del arbol que se guardo bajo el nombre de la ciudad.

Complejidad = O(N)

Requerimiento 2:

```python
def eventoDuracion(catalog,smin,smax):
    #obtener máxima duración
    max = (maxKey(catalog['durations']))
    maxsize1 = me.getValue(om.get(catalog['durations'], max))
    maxsize = lt.size(maxsize1)
    mmax = (max, maxsize)
    #obtener valores en rango
    val = om.values(catalog['durations'], smin, smax)
    #iterar lista
    ltiteration = lt.iterator(val)
    #Variables que devolver
    totalsize = 0
    mina = lt.newList('ARRAY_LIST')
    maxa = lt.newList('ARRAY_LIST')
    #Contadores
    minvalues = 0
    maxvalues = 0

    #For de tamaño total y minimos
    for event in ltiteration:
        # For de 3 iteraciones para los 3 primeros valores que encuentre
        if minvalues < 3:
            #Ordenar por pais-ciudad la lista de valores en la duración
            sa.sort(event, compareCities)
            datos = lt.iterator(event)
            for dato in datos:
                #Recorrer cada dato de la lista de la duración
                if minvalues < 3:
                    #Asegura que no se hayan obtenido ya 3 valores minimos
                    lt.addFirst(mina, dato)
                    minvalues = minvalues + 1
                else:
                    break
        #Cuenta el numero total de avistamientos en el rango
        totalsize = totalsize + lt.size(event)


    #While de valores maximos de 3 iteraciones
    while maxvalues < 3:
        evento = lt.lastElement(val)
        sa.sort(evento, compareCities)
        #Se obtiene la ultima duracion registrada
        while lt.size(evento) > 0:
            #while se revisa la lista de la duracion hasta que no tenga más datos
            if maxvalues < 3:
                #Se asegura que solo se tomen los ultimos 3 datos
                ultima = lt.lastElement(evento)
                lt.addFirst(maxa, ultima)
                lt.removeLast(evento)
                maxvalues = maxvalues + 1
            else:
                break
        lt.removeLast(val)
    #Se obtiene el numero de diferentes duraciones
    durations = om.size(catalog['durations'])

    return durations, mmax, totalsize, mina, maxa
```

Con respecto a la complejidad en este requerimiento, es notorio que hay una instancia en la que se encuentran 2 for y un sort entre estos y una donde se encuentran 2 whiles, con lo que sin observar nada más se consideraría una complejidad de $O(N^{5/2})$ entre un for y un shell sort.

Pero si se considera en si lo que ocurre en las instancias:

En la primera que comienza en el for de la línea 249, este recorre todos los valores de duración en sí, pero por el if de la línea 251, solo se recorren los valores, el sort y el segundo for, necesarios para conseguir los primeros 3 datos de entre los valores de duración especificados, para ser una iteración de, considerando M como el número de duraciones consideradas y N el número de datos en la duración, $O(N*M^{3/2})$, igualmente siendo los valores de N o M entre 1 o 3, al mínimo cada valor de 1 tener 1 o más valores guardados en esa duración.

En la segunda que comienza en el while que se considerara de complejidad **P**, en la línea 268 que máximo itera 3 veces considerando que se buscan obtener los últimos 3 valores de los datos, pues este sería el caso en el que los últimos datos se encuentren uno en un valor de duración diferente, en lo que se basa el segundo while, la situación en la que haya una lista, para la cual revisa todos los que estén presentes hasta obtener tres datos, dándole una complejidad similar a **P**, desde el ultimo hasta el primero si no hay 3 datos en una duración, con lo que se consideraría que en el peor de los casos se realizan 9 iteraciones, entre 1 y 3 datos por iteración de revisar las duraciones y los valores en cada una, dando una complejidad $O(P^2)$ considerando que P igual es un valor considerablemente bajo.

Dando en conclusión una complejidad de $O(N*M^{3/2})$.

Requerimiento 3:

```
def getHorasAvistamiento(cont, limMin, limMax):
    solucion = lt.newList()
    arbol = cont['hours']
    setLlaves = om.values(arbol,limMin,limMax)
    llaves =  lt.iterator(setLlaves)
    for llave in llaves:
        subarbol = llave
        keyssub = om.keySet(subarbol)
        keys = lt.iterator(keyssub)
        for key in keys:
            duplisha = om.get(subarbol,key)
            evento = me.getValue(duplisha)
            lt.addLast(solucion,evento)

    return solucion
```

El requerimineto tres requiere dos distintos ordenes, primero orden de la hora en el dia en el que el avistamiento fue registrado y despues de estos avistamientos, si hay multiplles que tengan la misma hora se organizar por fecha de registro de mas antiguo a mas nuevo. Con esto en mente, se planteo utilizar una estructura de un arbol que contiene subarboles, el primer arbol usa llaves la hora de registro del evento y el subarbol usa como llaves las fechas de los eventos. Con esto en mente las complejidades para encontrar los eventos que ocurrieron en un rango de tiempo del dia organizados ademas de eso por fecha si ocurren a la misma hora seria O(N*M), siendo N el tamano del arreglo de las llaves que estan en el rango introducido por el usuario, peor caso es el numero de llaves que guarda el arbol principal, M es el tamano del subarbol de los eventos que ocurrieron a la misma hora, este es considerablemente mas pequeno que N por lo que la complejidad final termina siendo bastante buena.

Complejidad = O(N*M)

Requerimiento 4:

```python
## Funciones para Req4
def getDatesAvistamiento(cont, limMin, limMax):
    solucion = lt.newList()
    arbol = cont['dates']
    setLlaves = om.values(arbol,limMin,limMax)
    llaves =  lt.iterator(setLlaves)
    for llave in llaves:
        subarbol = llave
        keyssub = om.keySet(subarbol)
        keys = lt.iterator(keyssub)
        for key in keys:
            duplisha = om.get(subarbol,key)
            evento = me.getValue(duplisha)
            lt.addLast(solucion,evento)

    return solucion
```

El req4 es muy parecido al req3 ya que estos se basan en la misma estructura, lo unico que cambia del req 3 a este es que el arbol grande guarda las fechas los subarboles guardan los tiempos en el dia en el que sucede el evento. Por esta razon la complejidad de este requerimiento es O(N*M) N siendo el tamano de la lista de las llaves en el rango, peor caso es el tamano del arbol principal y la M seria el tamano del subarbol el cual es mucho menor.

Complejidad: O(N*M)

Requerimiento 5:

```python
def geoviews(catalog, long1, long2, lat1, lat2):
    val = om.values(catalog['latlong'], long1, long2)
    #iterar lista de arboles
    ltiteration = lt.iterator(val)

    #Contadores
    minvalue = 0
    maxvalue = 0
    totalsize = 0
    #Lista para resultados
    resultf = lt.newList('ARRAY_LIST')
    #Se revisa cada arbol de longitud y de paso calcular minimos
    for long in ltiteration:
        valong = om.values(long, lat1, lat2)
        valongg = lt.iterator(valong)
        #Se revisa cada arbol de latitud
        for key in valongg:
            kk = lt.iterator(key)
            #Se revisa cada valor en la lista de la misma latitud y longitud
            for value in kk:
                #Se revisa que no se haya registrado ya los 5 primeros valores
                if minvalue < 5:
                    lt.addLast(resultf, value)
                    minvalue = minvalue + 1
                totalsize = totalsize + 1
    tt = totalsize - 5

    #Se calculan maximos
    while maxvalue < 5 and tt > 0:
        #Se saca la maxima longitud
        event = lt.lastElement(val)
        evento = om.valueSet(event)
        while lt.size(evento) > 0:
            ultima = lt.lastElement(evento)
            ultima = lt.iterator(ultima)
            #Se saca la maxima latitud y revisan los datos de la lista
            for caso in ultima:
                #Si no se han tomado ya todos los valores existentes o tomado 5 valores maximos, se registra
                if maxvalue < 5 and tt > 0:
                    lt.addLast(resultf, caso)
                    lt.removeLast(evento)
                    maxvalue = maxvalue + 1
                    tt = tt - 1
                else:
                    break


        lt.removeLast(val)

    sa.sort(resultf, compareLati)

    return totalsize, resultf
```

Considerando la complejidad temporal en Notación O de este requerimiento, si se considera en sí solo las funciones presentes, sería notorio que hay dos instancias en las que se realizan 3 iteraciones, sea ya un for dentro de un for dentro de un for, o un for dentro de un while dentro de un while, lo cual con facilidad daría una complejidad de $O(N^3)$.

La primera complejidad sería fatal para el tiempo de ejecución, pues es considerablemente alta si no se analiza la composición del código, pues en la primera instancia que comienza en la línea 300, el primer for recorre las longitudes, y el segundo de la línea 304 las latitudes registradas en

cada longitud, con lo que, en el tercer for se recorren los datos presentes en esa longitud y latitud en específico, con lo que se observa que solamente se recorren todos los datos presentes entre las longitudes y latitudes especificadas, que si son estos M datos de longitud, N de latitud en cada uno que no deberían ser excesivos, y de 1 a 3 datos en una misma latitud y longitud que se considerara un valor P, resulta en complejidad O(MNP).

En la segunda instancia se realizan 3 iteraciones, pero estas solo ocurren hasta que 5 datos o menos si no hay 10 datos en el rango especificado hayan sido obtenidos, con lo que la complejidad en realidad sería de R datos de longitud, mínimo 1 máximo 5, S datos de latitud, mínimo 1 máximo 5, y T datos de eventos en esa localización, mínimo 1 y en la situación más absurda 5. Con lo que se obtiene una complejidad O(RST).

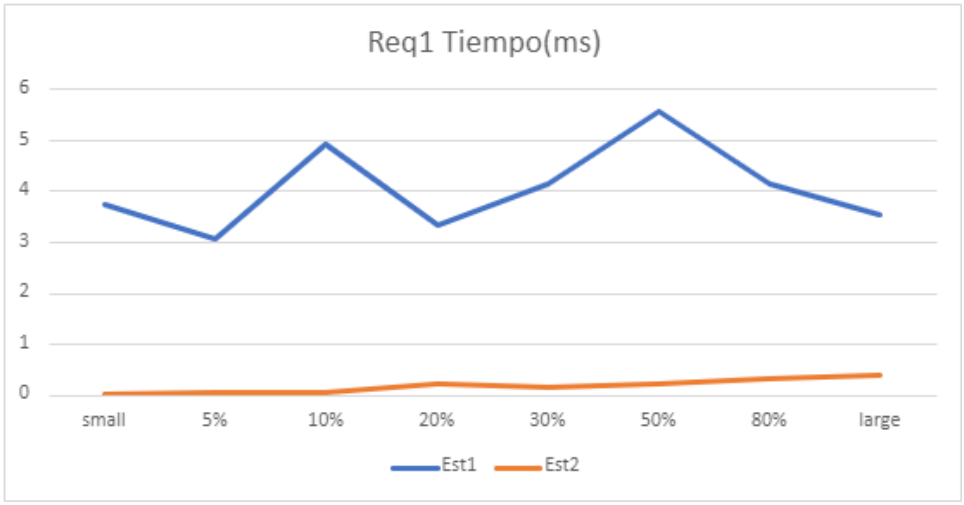
Por último, hay un Shell sort de complejidad $O(n^{3/2})$ respecto a los datos resultantes n que son menor o iguales a 10.

Pero considerando complejidades y la que gastara más recursos, sería en conclusión O(MNP).

Prueba de tiempos de ejecución:

Requerimiento 1:

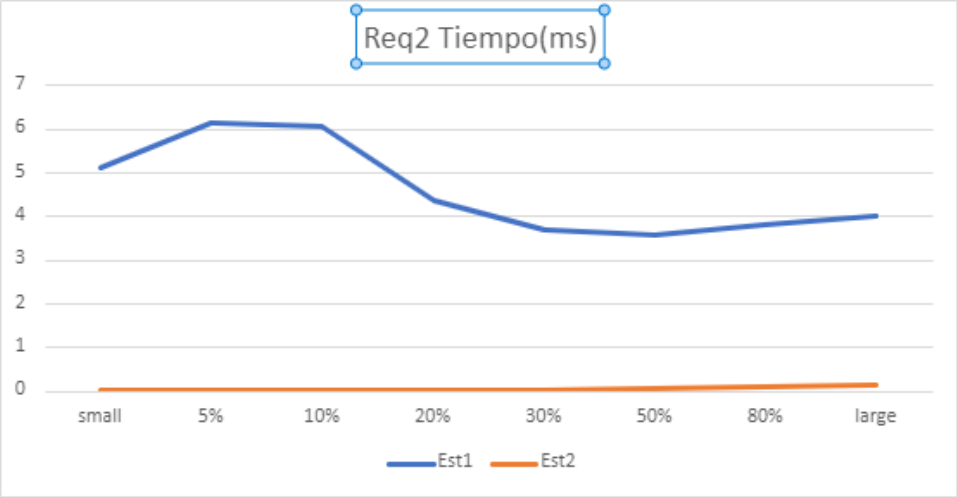
Req1 Tiempo (ms)		
Tamaño	Est1	Est2
small	3,729599714	0,0115018
5%	3,077382326	0,040006399
10%	4,931311369	0,06701183
20%	3,337337255	0,22854018
30%	4,160262823	0,15852785
50%	5,560162067	0,2170384
80%	4,159150839	0,3370590
large	3,54363656	0,3835673



En el analisis de tiempo del requerimiento 1 podemos ver un comportamiento bastante paresido en ambos tests de los estudiantes, esto quiere decir que no hay mayores cambios y que el algoritmo funciona bien con todos los datos.

Requerimiento 2:

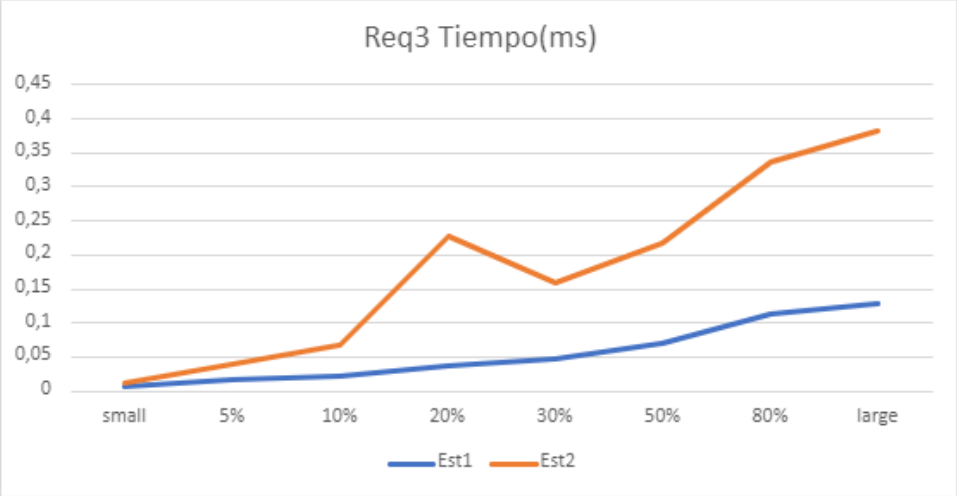
Req2 Tiempo (ms)		
Tamaño	Est1	Est2
small	5,116994143	0,0044994
5%	6,165177345	0,0060008
10%	6,08835578	0,0110018
20%	4,361696243	0,0195038
30%	3,685471773	0,0325053
50%	3,576239109	0,0745130
80%	3,801268578	0,1065187
large	4,000256538	0,1345243



Analizando al grafica de los tiempos del requerimiento dos podemos ver como en este tampoco hay demasiada variabilidad, no solo eso pero nunca pasa de 4 segundos para ninguno de los dos estudiantes por lo que se conlcuye que es un rendimiento adecuado.

Requerimiento 3:

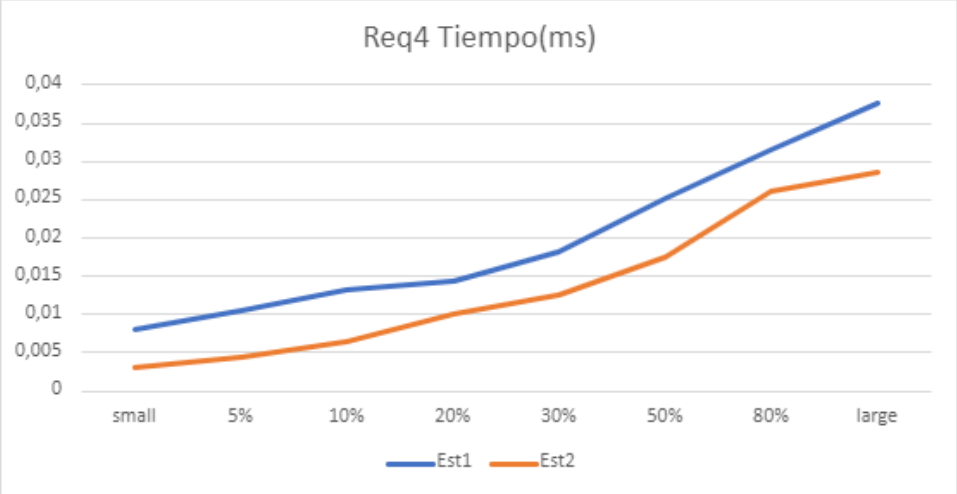
Req3 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,007021666	0,004000
5%	0,017819166	0,009501
10%	0,02215004	0,017502
20%	0,037003517	0,025004
30%	0,048152447	0,173029
50%	0,069853544	0,248043
80%	0,114447594	0,445079
large	0,128329277	0,112519



En el requrimiento 3 pasa algo raro y es que el estudiante dos tiene mayor demora que el 1 lo cual no estaba pasando anteriormente, de todas maneras el tiempo de ejecucion es tan bajo que la verdad no se nota la diferencia del aumento de los datos por lo que se concluye que es aceptable.

Requerimiento 4:

Req4 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,008079767	0,002999544
5%	0,010522842	0,00450182
10%	0,013339281	0,006500959
20%	0,014407158	0,010002613
30%	0,018088341	0,012503
50%	0,025217056	0,017502785
80%	0,031472445	0,026004314
large	0,037665606	0,028506

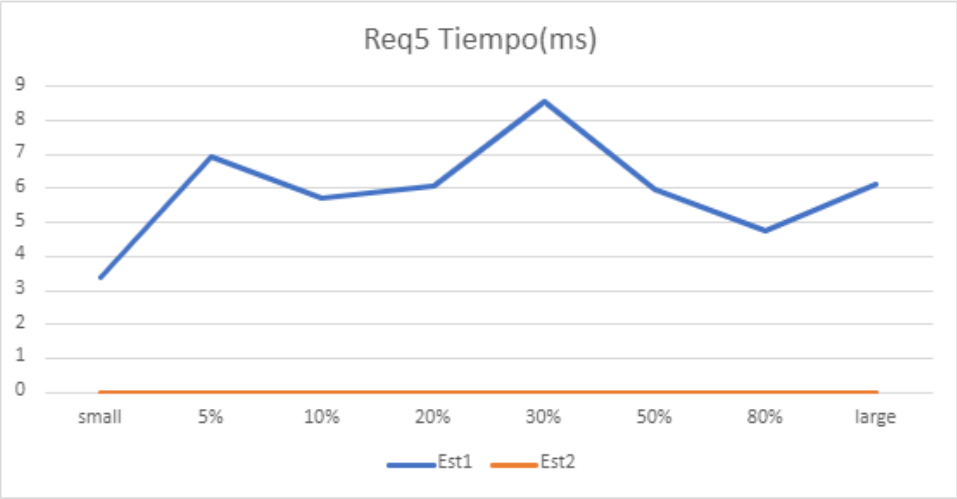


El requerimiento 4 debido a que es muy paresido al req 3 tiene un rendimiento similar aunque este es incluso mas rapido, esto se puede deber a que el numero de elementos guardados por fecha disminuye los repetidos y gracias a solo ser una busqueda en el arbol grande hace que este sea mas eficientes. Nuevamente la variacion de tiempos entre tamanos de datos es casi la misma por lo que se concluye que es una solucion valida.

Requerimiento 5:

Req5 Tiempo (ms)		
Tamaño	Tiempo (ms)	Est2
small	3,363271236	0,00150
5%	6,948454142	0,00200
10%	5,700587273	0,00150

20%	6,075069666	0,00100
30%	8,552584887	0,001498461
50%	5,988081694	0,00100
80%	4,730304956	0,00150
large	6,145842552	0,00150



El requerimiento 5 es probablemente el que mas variabilidad tiene de todos los requerimientos sin embargo este no se pasa de 6 por lo que tampoco es muy grabe, es interesante notar que en este requerimiento el estudiante 2 casi no vio cambios en el desempeno del algoritmo sin importar los datos. El tiempo de ejecucion sigue siendo muy bueno por lo que se considera una solucion valida para el req 5 tambien.

Respecto a la implementación:

El requerimiento 2 fue realizado por el estudiante 1, y el requerimiento 3 fue realizado por el estudiante 2.