

Documento de análisis Reto 3

- **Nombres:**

Estudiante 1: Samuel Josué Freire Tarazona, 202111460, s.freire@uniandes.edu.co ----->

Requerimiento 2 (Individual) = Samuel Josué Freire Tarazona

Estudiante 2: José David Martínez Oliveros, 202116677, jd.martinezol@uniandes.edu.co----->

Requerimiento 3 (Individual)= José David Martínez Oliveros

- **Análisis de complejidad:**

- **Requerimiento 1:**

- Primera parte:

```
total_avistamientos = om.size(catalogo['IndiceCiudad'])
```

- $O(k)$

- Segunda parte:

```
llavevalor = om.get(catalogo['IndiceCiudad'],ciudad)
```

- $O(n)$

- Tercera parte:

```
valor = me.getValue(llavevalor)['FechaIndice']
```

- $O(k)$

- Cuarta parte:

```
valores = om.valueSet(valor)
```

- $O(n)$

- Quinta parte:

```
lista_avistamientos_crudos = me.getValue(llavevalor)['ListaAvistamientos']
```

- $O(k)$

- parte:

```
lista3primeros = lt.subList(valores,1,3)
```

- $O(n)$

- Sexta parte:

```
for c in lt.iterator(lista3primeros):
    for j in lt.iterator(c['ListaAvistamientosporFecha']):
        if int(lt.size(primeros3avistamientos)) < 3:
            lt.addLast(primeros3avistamientos,j)
```

- $O(n)$

- Séptima parte:

```
lista3ultimos = lt.subList(valores,int(lt.size(valores))-2,3)
```

- $O(n)$

- Octava parte:

```
for c in lt.iterator(lista3ultimos):
    for j in lt.iterator(c['ListaAvistamientosporFecha']):
        lt.addLast(ultimos3avistamientos,j)
```

- $O(n)$

- Novena parte:

```
lista_final = lt.subList(ultimos3avistamientos,int(lt.size(ultimos3avistamientos))-2,3)
```

- $O(n)$

- Décima parte:

```
primeros3avistamientos = lt.newList(['ARRAY_LIST'])
for k in lt.iterator(valores):
    for l in lt.iterator(k['ListaAvistamientosporFecha']):
        lt.addLast(primeros3avistamientos,l)
```

- $O(n)$

- Undécima parte:

```
medida = lt.size(lista_avistamientos_crudos)
lista5primeros = lt.subList(ciudades_orden,1,5)
```

- $O(n)$

- **COMPLEJIDAD GENERAL:** $O(n)$

- **JUSTIFICACION:** En este caso la complejidad para este requerimiento, en notación Big O, dio $O(n)$. Esto se debe a que si se revisan cada parte del algoritmo, se podrá ver que la parte que en un dado caso guiara los tiempos del programa es un recorrido. Específicamente, la complejidad cuadrática, resulta porque este recorrido es un recorrido doble. Sin embargo, no queda cuadrática. Puesto que, el recorrido más grande, se hace sobre un rango que se conoce y nunca va superar ese rango. Y se opera con condicionales

para siempre tener esta misma cantidad a recorrer. Sin superar en ningún momento la complejidad lineal. Por lo que cuando se hace el segundo recorrido solo se hace las veces que se definieron. Por lo tanto, la complejidad de este requerimiento resulta ser $O(n)$; porque se hacen recorridos dobles en diferentes listas de valores para poder organizar la información. Sin olvidar que estos recorridos dobles no son cuadráticos, sino lineales, por el rango predefinido.

○ Requerimiento 2:

- Primera parte:

```
maslargas = lt.subList(om.keySet(catalogo['IndiceDuracionseg']),1
```

- $O(n)$

- Segunda parte:

```
final = lt.newList('ARRAY_LIST')
duracion_top = lt.newList('ARRAY_LIST')
for k in lt.iterator(maslargas):
    lt.addFirst(final,k)
    valor = me.getValue(om.get(catalogo['IndiceDuracionseg'],k))['ListaAvistamientos']
    ordenduracion = DuracionMasLargas(k,lt.size(valor))
    lt.addFirst(duracion_top,ordenduracion)
```

- $O(n)$

- Tercera parte:

```
medida = om.size(catalogo['IndiceDuracionseg'])
```

- $O(k)$

- Cuarta parte:

```
llaves = om.keys(catalogo['IndiceDuracionseg'],duracion_inicial,duracion_final)
```

- $O(n)$

- Quinta parte:

```
for c in lt.iterator(llaves):
    llavevalor = om.get(catalogo['IndiceDuracionseg'],c)
    cantidades = me.getValue(llavevalor)['ListaAvistamientos']
    total += int(lt.size(cantidades))
```

- $O(n^2)$

- Sexta parte:

```
sublistaprimeros = lt.subList(llaves,1,3)
```

- $O(n)$

- Séptima parte:

```
for primeros in lt.iterator(sublistaprimeros):
    llave_valor_primeros = om.get(catalogo['In
    mapa = me.getValue(llave_valor_primeros)['
    for ciudad in lt.iterator(om.keySet(mapa))
        llavevalorciudad = om.get(mapa,ciudad)
        valorciudad = me.getValue(llavevalorci
    lt.addLast(avistamientosrango,valorciu
```

- $O(n^2)$

- Octavo parte:

```
sublistaultimos = lt.subList(llaves,lt.size(llaves)-2,3)
```

- $O(n)$

- Novena parte:

```
for ultimos in lt.iterator(sublistaultimos):
    llave_valor_primeros = om.get(catalogo['
    mapa = me.getValue(llave_valor_primeros)
    for ciudad in lt.iterator(om.keySet(mapa
        llavevalorciudad = om.get(mapa,ciuda
        valorciudad = me.getValue(llavevalor
    lt.addLast(avistamientosrango,valorc
```

- $O(n^2)$

- Décima parte:

```
for ciudades in lt.iterator(avistamientosrango):
    for internas in lt.iterator(ciudades):
        lt.addLast(avistamientosrangofinal,inter
```

- $O(n^2)$

- Undécima parte:

```

primeros3finales = lt.subList(avistamientosrango,1,3)
ultimos3finales = lt.subList(avistamientosrango,lt.size(avistamientosrango)-2,3)
return total_medida, primeros3finales, ultimos3finales, duracion, top

```

- $O(n)$
- **COMPLEJIDAD GENERAL:** $O(n^2)$
- **JUSTIFICACION:** En este caso la complejidad para este requerimiento, en notación Big O, da $O(n^2)$. Esta complejidad es similar a la anterior. Específicamente, esta complejidad resulta; porque se hacen ciertos recorridos especiales. Estos recorridos son recorridos dobles, sobre listas de valores, que en este caso es similar a la anterior. Esto se debe a que se necesitaba organizar elementos por el valor de la su duración. Esta complejidad cubica, era necesaria para poder entrar a las estructuras interiores del mapa. En este caso, se recorren diferentes listas con diferentes valores, los cuales se encuentran en las estructuras de datos del BST. Esta estructura en este caso, genera esta complejidad, ya que ayuda a organizar los valores por duración e internamente por ciudad y país. Por lo tanto, la complejidad de este requerimiento es de $O(n^2)$. Puesto que, se hacen recorridos dobles, sobre listas de diferentes valores. Sin olvidar, que son cantidades diferentes, ninguna con el total de archivos.

○ Requerimiento 3:

- Primera parte:

```

medida = om.size(catalogo['IndiceDuracionHM'])

```

- $O(k)$
- Segunda parte:

```

mas_antiguas_llaves = lt.subList(om.keySet(catalogo['IndiceDuracionHM']),1,5)

```

- $O(n)$
- Tercera parte:

```

for c in lt.iterator(mas_antiguas_llaves):
    llavevalor = om.get(catalogo['IndiceFecha'],c)
    cantidades = me.getValue(llavevalor)['ListaAvistamientos']
    fechaycantidad = FechaMasAntiguas(c,lt.size(cantidades))
    lt.addLast(top5antiguas, fechaycantidad)

```

- $O(n)$
- Cuarta parte:

```
llaves = om.keys(catalogo['IndiceDuracionHM'],hora_inicial,hora_final)
```

- $O(n)$

- Quinta parte:

```
for c in lt.iterator(llaves):  
    llavevalor = om.get(catalogo['IndiceDuracionHM'],c)  
    cantidades = me.getValue(llavevalor)['ListaAvistamientos']  
    total += int(lt.size(cantidades))
```

- $O(n^2)$

- Sexta parte:

```
valores = om.values(catalogo['IndiceDuracionHM'],hora_inicial,hora_final)
```

- $O(n)$

- Séptima parte:

```
valores_primeros = lt.subList(valores,1,3)
```

- $O(n)$

- Octava parte:

```
for k in lt.iterator(valores_primeros):  
    lista = k['ListaAvistamientos']  
    for j in lt.iterator(lista):  
        lt.addLast(listaavistamientosprimeros,j)
```

- $O(n)$

- Novena parte:

```
primeros_orden = sortDuracionRango(listaavistamientosprimeros)
```

- $O(n \log n)$

- Décima parte:

```
for l in lt.iterator(finales):  
    lista = l['ListaAvistamientos']  
    for t in lt.iterator(lista):  
        lt.addLast(listaavistamientosultimos,t)
```

- $O(n)$

- Undécima parte:

```
ultimos_orden = sortDuracionRango(listaavistamientosultimos)
```

- $O(n \log n)$

- Undécima parte:

```
lista_final = lt.subList(ultimos_orden,int(lt.size(ultimos_orden))-2,3)
```

- $O(n)$

- **COMPLEJIDAD GENERAL:** $O(n^2)$
- **JUSTIFICACION:** En este caso, la complejidad de este requerimiento, en notación Big O resulta en $O(n^2)$. Esto se debe a que cada recorrido este hecho sobre un rango que ya se definió. Por lo que, el recorrido interno o segundo recorrido se va a ser las veces ya conocidas dando al final una complejidad lineal. Por lo tanto, la complejidad $O(n^2)$ que sale en este requerimiento, resulta de diversos recorridos dobles que se hacen a lo largo del algoritmo para poder extraer información de la estructura de datos de Mapas. Sin olvidar que estos recorridos son sobre datos ya conocidos o rangos ya definidos.

○ **Requerimiento 4:**

- Primera parte:

```
medida = om.size(catalogo['IndiceFecha'])
```

- $O(k)$

- Segunda parte:

```
mas_antiguas_llaves = lt.subList(om.keySet(catalogo['IndiceFecha']),1,5)
top5antiguas = lt.subList('ARRAY LIST')
```

- $O(n)$

- Tercera parte:

```
for c in lt.iterator(mas_antiguas_llaves):
    llavevalor = om.get(catalogo['IndiceFecha'],c)
    cantidades = me.getValue(llavevalor)['ListaAvistamientos']
    fechaycantidad = FechaMasAntiguas(c,lt.size(cantidades))
    lt.addLast(top5antiguas,fechaycantidad)
```

- $O(n)$

- Cuarta parte:

```
llaves = om.keys(catalogo['IndiceFecha'],fecha_inicial,fecha_final)
total = 0
```

- $O(n)$

- Quinta parte:

```
for c in lt.iterator(llaves):
    llavevalor = om.get(catalogo['IndiceFecha'],c)
    cantidades = me.getValue(llavevalor)['ListaAvistamientos']
    total += int(lt.size(cantidades))
```

- $O(n^2)$

- Sexta parte:

```
valores = om.values(catalogo['IndiceFecha'],fecha_inicial,fecha_final)
valores_primeros = lt.subList(valores,1,3)
```

- $O(n)$

- Séptima parte:

```
valores_primeros = lt.subList(valores,1,3)
```

- $O(n)$

- Octava parte:

```
for k in lt.iterator(valores_primeros):
    lista = k['ListaAvistamientos']
    for j in lt.iterator(lista):
        lt.addLast(listaavistamientosprimeros,j)
```

- $O(n)$

- Novena parte:

```
primeros_orden = sortDuracionRango(listaavistamientosprimeros)
```

- $O(n \log n)$

- Décima parte:

```
primero3 = lt.subList(primeros_orden,1,3)
finales = lt.subList(valores,lt.size(valores)-2,3)
```

- $O(n)$

- Undécima parte:

```
for l in lt.iterator(finales):
    lista = l['ListaAvistamientos']
    for t in lt.iterator(lista):
        lt.addLast(listaavistamientosultimos,t)
```

- $O(n)$

- Undécima parte:

```
ultimos_orden = sortDuracionRango(listaavistamientosultimos)
```

- $O(n \log n)$

- Undécima parte:

```
lista_final = lt.subList(ultimos_orden,int(lt.size(ultimos_orden))-2,3)
```

- $O(n)$

- **COMPLEJIDAD GENERAL:** $O(n^2)$

- **JUSTIFICACION:** En este caso, la complejidad de este requerimiento, en notación Big O resulta en $O(n^2)$. Caso similar a lo que sucede en requerimiento anterior. Esta complejidad se debe a los recorridos que se hacen internos. Ya que, al tener arboles BST, la búsqueda elementos tiene una complejidad de $O(N)$. Por lo que al hacer una búsqueda en un recorrido, se va a generar una complejidad cuadrática. Por lo tanto, la complejidad $O(n^2)$ que sale en este requerimiento, resulta de diversos recorridos dobles que se hacen a lo largo del algoritmo para poder extraer información de la estructura de datos de Mapas. Sin olvidar que estos recorridos son sobre datos ya conocidos o rangos ya definidos.

○ **Requerimiento 5:**

- Primera parte:

```
valores = om.values(catalogo['IndiceLongitud'],longitud_inicial,longitud_final)
```

- $O(n)$

- Segunda parte:

```
for c in lt.iterator(valores):
    valores_latitud = om.values(c['Latitud'])
    for j in lt.iterator(valores_latitud):
        total += lt.size(j['ListaAvistamientos'])
```

- $O(n^2)$

- Tercera parte:

```
for c in lt.iterator(valores):
    valores_latitud = om.values(c['L
    if lt.size(valores_latitud) >= 1
        for j in lt.iterator(valores
            lt.addLast(orden,j)
```

- $O(n^2)$

- Cuarta parte:

```
orden_latitud_longitud = sortlatitudinterna(orden)
```

- $O(n \log n)$

- Quinta parte:

```
primeros5 = lt.subList(orden_latitud_longitud,1,5)
```

- $O(n)$

- Sexta parte:

```
for k in lt.iterator(primeros5):
    for l in lt.iterator(k['Lista
        lt.addLast(resultados,l)
```

- $O(n)$

- Séptima parte:

```
ultimos5 = lt.subList(orden_latitud_longitud,lt.size
```

- $O(n)$

- Octava parte:

```
for k in lt.iterator(ultimos5):
    for l in lt.iterator(k['List
        lt.addLast(resultados,l)
```

- $O(n)$

- **COMPLEJIDAD GENERAL:** $O(n^2)$

- **JUSTIFICACION:** En este caso, la complejidad de este requerimiento, en notación Big O, dio $O(n^2)$. Esta complejidad es muy parecida al requerimiento anterior. Esta complejidad resulta de ciertos recorridos especiales a lo largo del algoritmo. Específicamente, este

recorrido es un recorrido doble. Estos recorridos se hacen sobre diferentes listas de valores. Cada uno de ellos para poder entrar en cierta información contenida en la estructura de datos, que en este caso se usa un Mapa ordenado específicamente un BST. Sin olvidar, que esta complejidad o recorridos no se hacen sobre la totalidad de los datos, sino sobre ciertas listas reducidas que contiene información fundamental. Además, esta complejidad no hace que el tiempo sea mayor, sino que salen tiempos completamente normales. Por lo tanto, la complejidad $O(n^2)$ que salió en este requerimiento, está dada por ciertos recorridos cúbicos que se hacen.

- **Pruebas de tiempo:**

Pruebas de tiempo				
Req 1	phoenix	las vegas	seattle	Promedio
	31,25	0	0	10,4166667
Req 2	20-80	30-150	100-120	Promedio
	31,25	31,25	62,5	41,6666667
Req 3	20:00-22:00	22:00-23:59	14:00--19:00	Promedio
	2843,75	1593,75	921,87	1786,45667
Req 4	1/01/1900-20	1929-01-01 1	1939-01-01 1	Promedio
	93,75	46,88	31,25	57,2933333
Req 5	109,05,,,,-103	122,00,,,,-100	10,,,50,,,20,,,	Promedio
	234,38	1062,5	62,5	453,126667