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.martinezo1@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(k)
    - 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)
    - 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^2)

- Septima 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^2)

- 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^2)

- Undécima parte:

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

  - O(n)

- **COMPLEJIDAD GENERAL**: O(n^2)
- **JUSTIFICACION**: En este caso la complejidad para este requerimiento, en notación Big O, dio O(n^2). 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. Es decir, se va a recorrer un datos tantas veces se recorran los otros. Para este algoritmo, los dos recorridos recorren una lista de valores, específicamente, recorren loístas de valores que cuentan con una valor similar. Estos recorridos dobles surgen de la organización usada por la estructura BST, la cual permite organizar los valores por su ciudad e internamente por fecha. Por lo tanto, la complejidad de este requerimiento resulta

ser O(n^2); porque se hacen recorridos dobles en diferentes listas de valores para poder organizar la información.

- **Requerimiento 2:**
  - Primera parte:

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

    - 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)
  - Sexta parte:

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

    - O(n)

- Septima 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)
```

  - O(n)

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

- **JUSTIFIACION**: 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 4:**
  - Primera parte:

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

    - O(k)
  - Segunda parte:

```
mas_antiguas_llaves = lt.subList(om.keySet(catalogo['IndiceFecha']),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['IndiceFecha'],fecha_inicial,fecha_final)
```

  - 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)

- Sexta parte:

```
valores = om.values(catalogo['IndiceFecha'],fecha_inicial,fecha_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^2)

- Novena parte:

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

  - O(nlog$n$)

- 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^2)
- Undécima parte:

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

  - O(nlog$n$)
- Undécima parte:

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

  - O(n)
- **COMPLEJIDAD GENERAL**: O(n^2)
- **JUSTIFIACION**: En este caso, la complejidad de este requerimiento, en notación Big O resulta en O(n^2). Caso similar, a lo que sucedió con las complejidades anteriores. Esta complejidad, de O(n^2), resulta nuevamente en recorridos especiales. Específicamente estos recorridos son dobles. Estos recorridos se dan en listas de diferentes valores. Cada uno de estos recorridos son necesarios para poder sacar información para su impresión. Esta complejidad cuadrática, no es sobre la totalidad de los datos sino que se hacen sobre listas reducidas, de alguna u otra manera. Realmente, la complejidad obtenida, es necesaria más que todo para la impresión. Específicamente, en este caso se usa la estructura BST, la cual permite organizar los datos por fechas y organizar todo al final con un m par de recorridos. 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.

- **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['ListaAvistamie
```

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