Universidad de los Andes
Facultad de Ingeniería
Reto 3- UFO Sightings
Estructuras de datos y algoritmos – sección 02

Estudiante 1:

Nombre: Sofia Torres Ramírez.
Código: 2020140872
Correo: s.torres21@uniandes.edu.co

Estudiante 2:

Nombre: Ana Margarita Flórez Ruiz.
Código: 201922242
Correo: a.florezr@uniandes.edu.co

• Análisis de complejidad temporal en Notación O para cada uno de los requerimientos.

Incluir una breve justificación de la complejidad temporal dada.

REQ 1 (Grupal): Contar los avistamientos en una ciudad

```
#Req 1-Contar avistamientos en una ciudad
def addCity (catalog,ufo):
    city= ufo["city"]
    index= catalog
    present= mp.contains(index,city)

    if not present:
        datetime= om.newMap(omaptype="RBT",
                            comparefunction=compareDates)
        count= 0
        info={"count":count, "datetime":datetime}
        mp.put(index,city,info)

    entry= mp.get(index,city)
    value= me.getValue(entry)
    addDate(value["datetime"],ufo)
    count= om.size(value["datetime"])
    value["count"]= count
```

Para el requerimiento 1 se tiene una complejidad es O(n) debido a que para encontrar los datos en el view se hace un recorrido dentro del map creado y además dentro de la tabla RBT de fechas dentro de las ciudades, haciendo que esté organizado por fechas haciendo que obtener los datos organizados sea más fácil en el view.

REQ 2 (Individual): Contar los avistamientos por duración
Realizado por Sofia Torres Ramírez.

```python
#Req2,Contar los avistamientos por duración
#número de avistamientos con la duración más larga
def longduration (catalog):
    durationtree=catalog["Duration"]
    maxduration=om.maxKey(durationtree)
    key_value=om.get(durationtree,maxduration)
    lst=me.getValue(key_value)
    numviews=lt.size(lst)
    return numviews

#mostrar avistamientos en un rango de duración
def sightings_in_range(catalog, segmin, segmax):
    durationtree=catalog["Duration"]
    range=om.values(durationtree,segmin,segmax)
    list_answer=lt.newList("ARRAY_LIST")
    for lst in lt.iterator(range):
        for sightings in lt.iterator(lst):
            lt.addLast(list_answer,sightings)
    firstthree=lt.subList(list_answer,1,3)
    lastthree=lt.subList(list_answer,(lt.size(list_answer)-2),3)
    return firstthree, lastthree
```

La complejidad de este requerimiento es O(log(N)) debido a que se puede ver que en la variable “range” se toma un rango de datos (en este caso de duraciones) pertenecientes al número de datos principal, porque se trabaja sobre una cantidad de datos más pequeña que la inicial. Ya sobre este intervalo se aplican dos ciclos, sin embargo, estos solo guardan un dato por iteración por lo tanto no se demora mucho en realizarlo.

REQ 3 (Individual): Contar avistamientos por Hora/Minutos del día
Realizado por Ana Margarita Florez Ruiz

```
#Req 3- Contar avistamientos por hora/minuros del día
def addHour(map, ufo):
    date= ufo["datetime"]
    date= dt.datetime.fromisoformat(date).time()
    present= om.contains(map, date)

    if present:
        entry= om.get(map, date)
        lista= me.getValue(entry)
        lt.addLast(lista, ufo)
    else:
        avistamientos= lt.newList("ARRAY_LIST")
        lt.addLast(avistamientos, ufo)
        om.put(map,date,avistamientos)
```

El requerimiento 3 cuenta con una complejidad de O(n) debido que para extraer los datos de la tabla RBT creada se debe hacer un recorrido de los datos a pesar de que estos ya estén ordenados por fecha.

REQ 4 (Grupal): Contar los avistamientos en un rango de fechas

```
#Req 4- Contar avistamientos en un rango de fechas
def addDate(map, ufo):
    date= ufo["datetime"]
    date= dt.datetime.fromisoformat(date)
    present= om.contains(map, date)

    if present:
        entry= om.get(map, date)
        lista= me.getValue(entry)
        lt.addLast(lista, ufo)
    else:
        avistamientos= lt.newList("ARRAY_LIST")
        lt.addLast(avistamientos, ufo)
        om.put(map,date,avistamientos)
```

El requerimiento 4 tiene una complejidad de O(n) ya que en el view se recorre la tabla RBT creada para obtener los datos, como es una tabla RBT ya está organizada por fechas por lo que obtener los datos es un simple recorrido a la tabla.

REQ 5 (Grupal): Contar los avistamientos de una Zona Geográfica

```
#Req 5.Contar los avistamientos de una Zona Geográfica

def num_in_range(catalog, longmin, longmax, latmin, latmax):
    longitudetree=catalog["Longitude"]
    longituderange=om.values(longitudetree,longmin,longmax)
    lst_in_range=lt.newList("ARRAY_LIST")
    for longitude in lt.iterator(longituderange):
        longitude=longitude["latitudeindex"]
        latituderange=om.values(longitude,latmin,latmax)
        if lt.size(latituderange)>=1:
            for elem in lt.iterator(latituderange):
                lt.addLast(lst_in_range,elem)
        else:
            lt.addLast(lst_in_range,latituderange)
    answer=lt.newList("ARRAY_LIST")
    for element in lt.iterator(lst_in_range):
        if not lt.isEmpty(element):
            for ufo in lt.iterator(element):
                lt.addLast(answer,ufo)
    return answer

def total_in_area(lst_in_range):
    size=lt.size(lst_in_range)
    return size
```

```
def total_in_area(lst_in_range):
    size=lt.size(lst_in_range)
    return size

def sorting_list(lst_in_range):
    sorted_list=mg.sort(lst_in_range, cmpBylongitude)
    return sorted_list

def cmpBylongitude(longitude1, longitude2):
    if longitude1["longitude"]!="" and longitude2["longitude"]!="":
        long1=float(longitude1["longitude"])
        long2=float(longitude2["longitude"])
        return long1<long2

def fivefirstlast(sorted_list):
    fivefirst=lt.subList(sorted_list,1,5)
    fivelast=lt.subList(sorted_list,(lt.size(sorted_list)-4),5)
    return fivefirst,fivelast
```

La complejidad de este requerimiento es O(log(N)) debido a que se puede ver que la variable "longituderange" se toma un rango de datos (en este caso de longitudes) pertenecientes al número de datos principal, porque se trabaja sobre una cantidad de datos más pequeña que la inicial. Ya sobre este intervalo se corre un ciclo, sin embargo, este ciclo es para obtener un número de datos aún menor, y sobre este intervalo se trabaja para conseguir lo que pide el requerimiento.