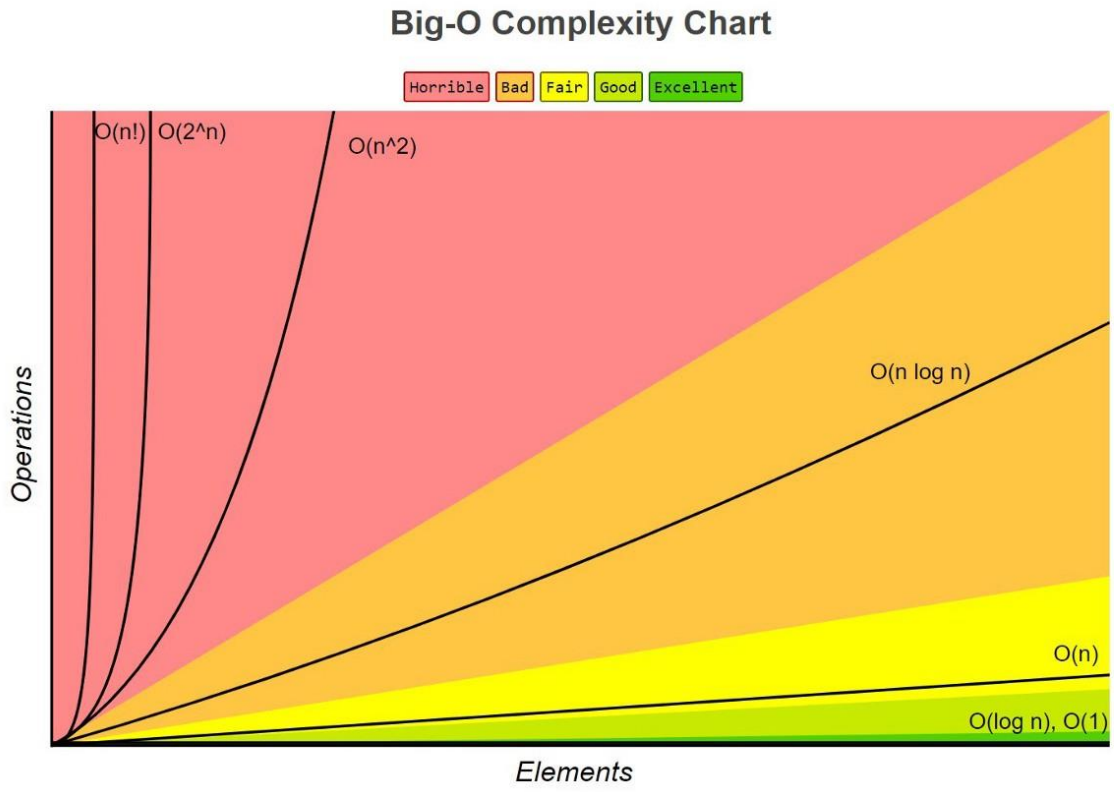
Documento de análisis reto 4

r.rincon@uniandes.edu.co – 202120414

b.raisbeck@uniandes.edu.co – 202120398



Requerimiento 1:

La complejidad de este requerimiento es O(n) ya que depende de la cantidad de estaciones que existan. Además, se usan unas funciones auxiliares que soportan la consulta del grafo y de condicionales, pero no afectan directamente a la complejidad temporal de manera drástica.

```
    grafo = catalog['Connections Directed']
    for i in lt.iterator(mp.keySet(catalog['Stations'])):
        info = me.getValue(mp.get(catalog['Stations'], i))
```

```
def addCasualMembers(catalog, trip, station):
    contadorCasual = 0
    if trip['Start Station Name'] == station:
        if trip['User Type'] == 'Casual Member':
                contadorCasual += 1
    mp.put(catalog['contador'], trip['Start Station Name'], contadorCasual)
```

Requerimiento 3:

Acá en este requerimiento la complejidad con lista de adyacencia: O(V+E) en caso de que se hubiera usado matriz hubiera sido de $O(V^2)$

```
def requerimiento3(catalog):
    componentes = scc.KosarajuSCC(catalog['Connections Directed'])
    numeroComponentes = scc.connectedComponents(componentes)
    return componentes, numeroComponentes
```

Requerimiento 4:

Para este requerimiento usamos el algoritmo de Dijkstra el cual tiene una complejidad de E log V para su peor caso en lista de adyacencia. Este nos soluciona la ruta mínima para un grafo dado.

```
def requerimiento4(catalog, nameOrigen, nameDestino):
    #headers = [['Station ID','Station Name','Out Trips','In Trips', 'R
    lista = lt.newList()
    origen = me.getValue(mp.get(catalog['Stations'], nameOrigen))
    solve = dijsktra.Dijkstra(catalog['Connections Directed'], origen)
    destino = me.getValue(mp.get(catalog['Stations'], nameDestino))
    destinoFinal = dijsktra.pathTo(solve, destino)
    finalTime = dijsktra.distTo(solve, destino)
    print("-------------------------------------------------")
    for i in range(st.size(destinoFinal)):
        edge = st.pop(destinoFinal)
        lt.addLast(lista, edge)
    print("Number of routes:", lt.size(lista))
    print("Total time:", finalTime,"[sec]")
    print("Total time:", round(finalTime/60,2),"[min]")
```

Requerimiento 5:

Para este requerimiento usamos ordered maps o arboles los cuales nos ayudan primero a ordenar la información por keys los cuales son las fechas y de esta forma obtener una complejidad en su peor caso de O (log n). Además, para solucionar las fechas repetidas solamente agregamos la información de la misma Key en una lista en el Value para evitar remplazar la información con la nueva key.

```
if trip['User Type'] == 'Annual Member':
    fecha = dt.strptime(trip['Start Time'], '%m/%d/%Y %H:%M')
    dia = fecha.strftime('%m/%d/%Y')
    secondfecha = mp.newMap()
    if om.contains(catalog['Req5'], dia):
        secondfecha = me.getValue(mp.get(catalog['Req5'], dia))
        mp.put(secondfecha, trip['Trip Id'], trip)
    else:
        mp.put(secondfecha, trip['Trip Id'], trip)
    om.put(catalog['Req5'], dia, secondfecha )
```

Requerimiento 6:

Por último, usamos HashMaps como estructura ya que necesitábamos filtrar la información por la ID de la bicicleta y al ser única pues nos facilita buscarlo. Además, igual que el requerimiento anterior simplemente agregamos a una lista la información repetida para evitar sumar complejidad al requerimiento el cual seria en total un O (1).

```
if mp.contains(catalog['Req6'], trip['Bike Id']):
    secondtrip = me.getValue(mp.get(catalog['Req6'], trip['Bike Id']))
    mp.put(secondtrip, trip['Trip Id'], trip)
else:
    mp.put(secondtrip, trip['Trip Id'], trip)
    mp.put(catalog['Req6'], trip['Bike Id'], secondtrip)
```