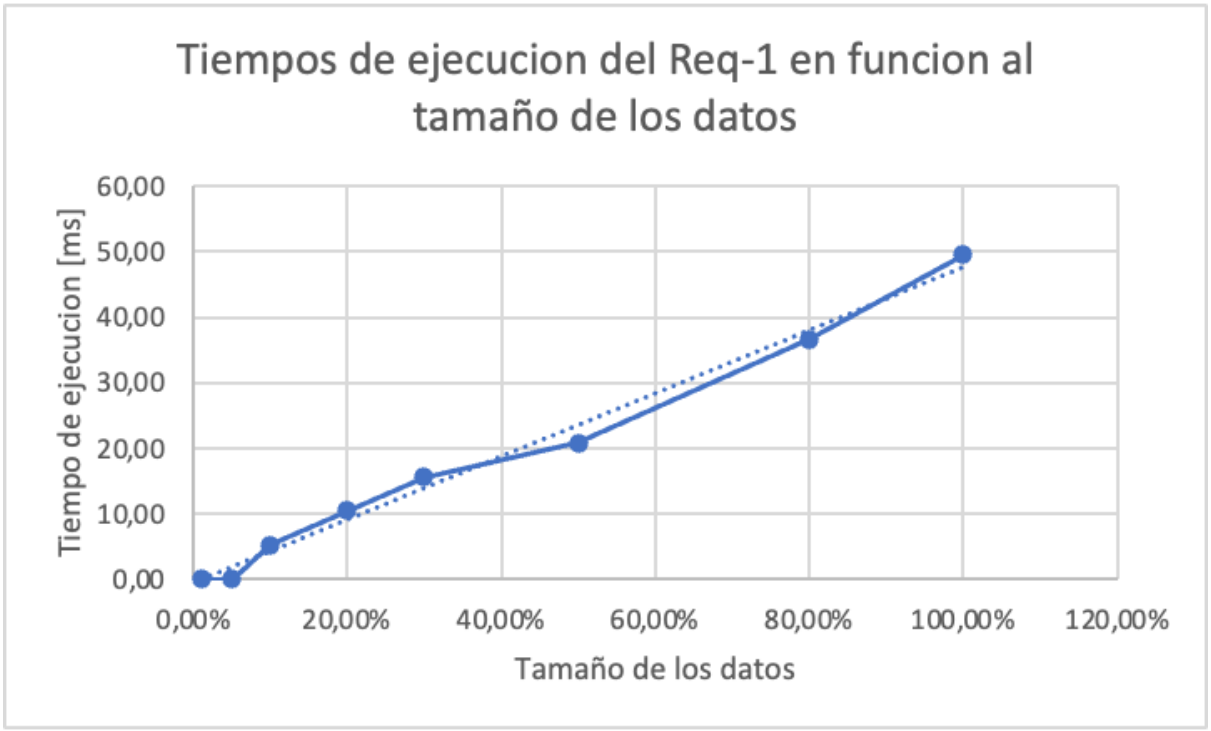
Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609

ANÁLISIS DE COMPLEJIDAD Y TIEMPOS DE EJECUCIÓN

Requerimiento 1 **O((N/M)+(n*log(n)))**

```
def sightingsByCity(expediente,ciudad):
    ciudad=me.getValue(mp.get(expediente["ciudades"],ciudad))
    ciudad=ms.sort(ciudad,cmpDates)
    return ciudad
```

En este requerimiento la complejidad temporal se divide en dos partes. Por un lado, el get() un mapa desordenado (tabla de hash) con manejo de colisiones Separated Chaining tiene una complejidad de O(N/M) siendo N el número de datos y M el tamaño de la tabla. El getValue() tiene una complejidad de O(1) ya que simplemente tiene que retornar el valor de la llave "value" de la entrada retornada. Por último, el merge sort tiene una complejidad temporal, en el peor de los casos, de O(n*log(n)). Por lo que este requerimiento en el peor caso presenta una complejidad de O((N/M)+(n*log(n))).



Para confirmar este comportamiento se hace referencia a la gráfica de pruebas temporales de manera tal que, con diferentes cantidades de datos sobre el tiempo de ejecución de la función, se obtiene un comportamiento bastante parecido al de una gráfica n*log(n).

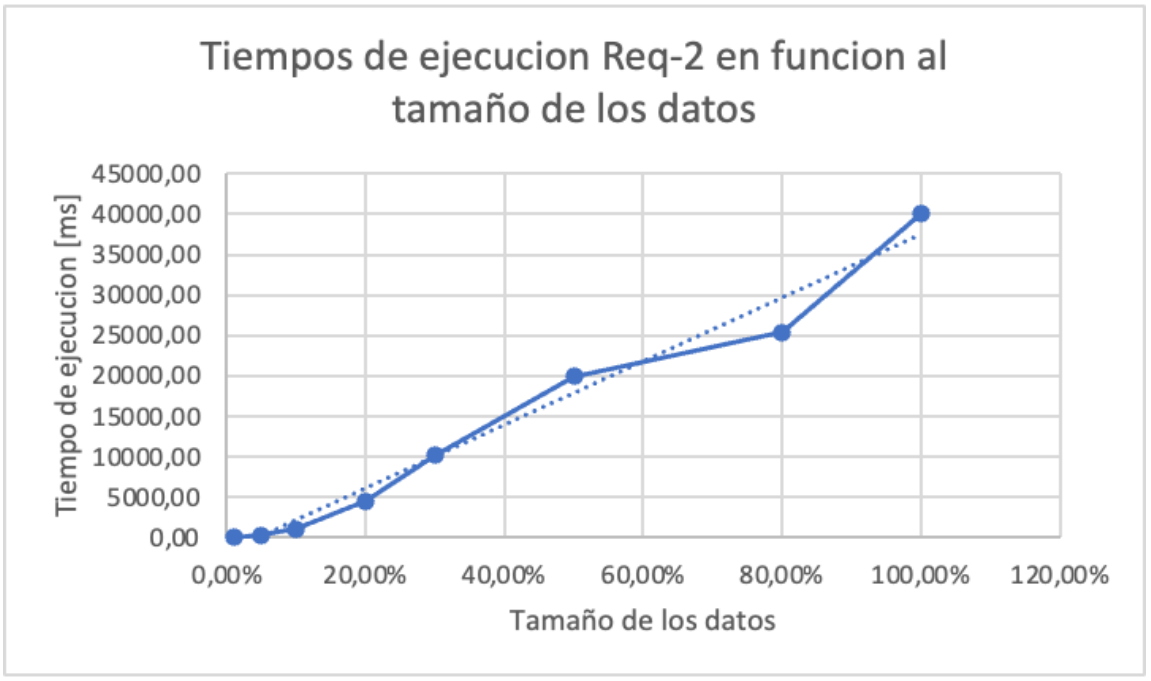
Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609

Requerimiento 2 O((3*log(n))+M+(n*log(n))) - Mateo Cote Canal

```
def avistamientosPorDuracion(expediente,duracionMin,duracionMax,requerimiento):
    if requerimiento=="2":
        llave="duraciones"
    elif requerimiento=="3":
        llave="horaMin"
        duracionMin=(datetime.datetime.strptime(duracionMin, '%H:%M:%S')).time()
        duracionMax=(datetime.datetime.strptime(duracionMax, '%H:%M:%S')).time()
    maxDuracion=mo.maxKey(expediente[llave])
    numAvistamientosMaxKey=lt.size(me.getValue(mo.get(expediente[llave],maxDuracion)))
    avisRango=mo.keys(expediente[llave],duracionMin,duracionMax)
    listaAvistamientos=lt.newList("SINGLE_LINKED")
    for key in lt.iterator(avisRango):
        duracion=me.getValue(mo.get(expediente[llave],key))
        for avis in lt.iterator(duracion):
            lt.addLast(listaAvistamientos,avis)
    if llave=="duraciones":
        listaAvistamientos=ms.sort(listaAvistamientos,cmpDuration)
    else:
        listaAvistamientos=ms.sort(listaAvistamientos,cmpHoraMin)
    return maxDuracion,numAvistamientosMaxKey,listaAvistamientos
```

En este caso el maxKey() en un mapa ordenado tipo RBT tiene una complejidad temporal del log(n) ya que, teniendo en cuenta que la altura maxima de un RBT es log(n), en el peor caso se tendría que recorrer la rama más larga para obtener la última llave de esta (la mayor de todas). De nuevo el get() en un mapa ordenado, también sería log(n) ya que en el peor caso se tendría que recorrer una rama entera, la cual puede tener como altura máxima log(n). Operaciones como size() o getValue() tienen como complejidad O(1). Teniendo M como el número de avistamientos resultantes que entraron al rango de las duraciones en segundos, el primer y segundo ciclo tendrían una complejidad O(M), ya que con esos dos se recorren M avistamientos que entraron al rango. Por último, en la parte final de la función se puede entrar a una y solo una de esas dos condiciones y, tendiendo en cuenta que en cualquiera de las dos se realiza un mergesort, la complejidad de esa parte es n*log(n). Por esto, la complejidad total de este algoritmo es de O((3*log(n))+M+(n*log(n)))

Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609



Acorde a lo que se puede apreciar de la gráfica en cuanto a tiempos de ejecución para este requerimiento, se puede llegar a concluir un carácter cuadrático o n*log(n). Esto se debe al crecimiento lento y continuo inicial de los datos. De todas formas no se puede descartar la opción de una identidad cúbica o exponencial debido al cambio abrupto hacia al intervalo final de la gráfica. De todas formas, esta gráfica tiende a lo esperado según el análisis de complejidad.

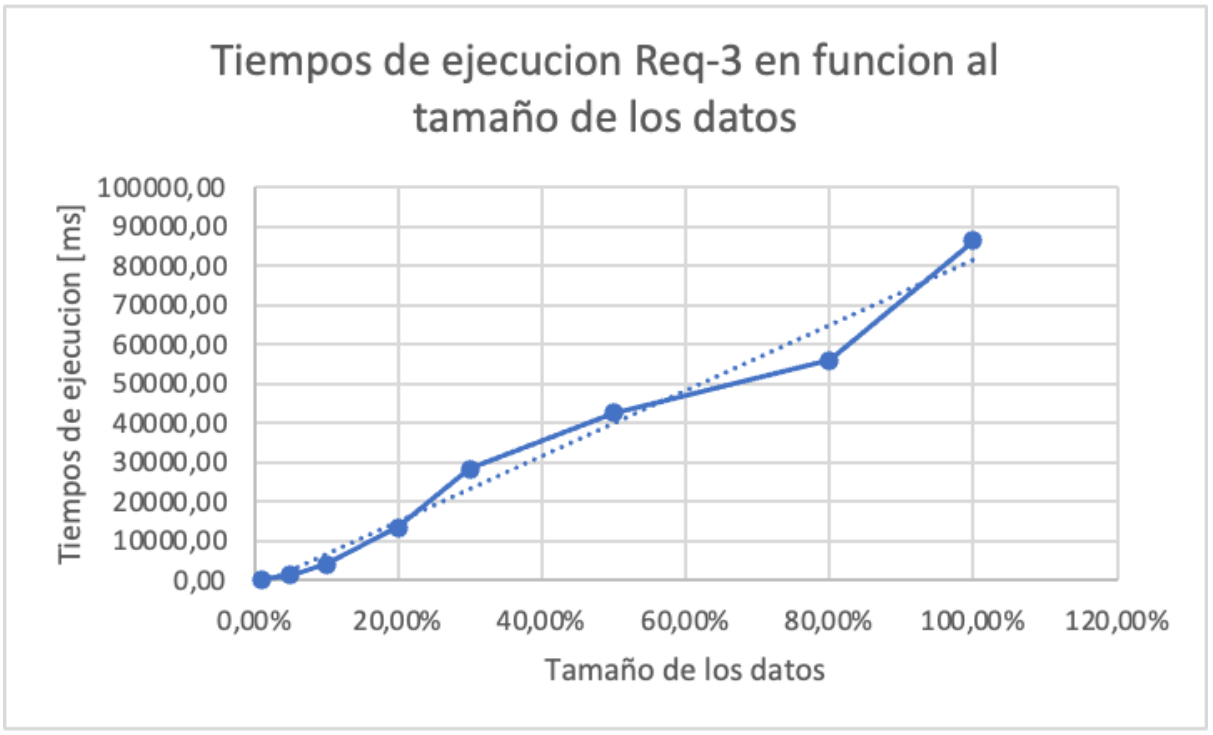
Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609

Requerimiento 3 **O((3*log(n))+M+(n*log(n)))** - **Mateo Cote Canal**

```
def avistamientosPorDuracion(expediente,duracionMin,duracionMax,requerimiento):
    if requerimiento=="2":
        llave="duraciones"
    elif requerimiento=="3":
        llave="horaMin"
        duracionMin=(datetime.datetime.strptime(duracionMin, '%H:%M:%S')).time()
        duracionMax=(datetime.datetime.strptime(duracionMax, '%H:%M:%S')).time()
    maxDuracion=mo.maxKey(expediente[llave])
    numAvistamientosMaxKey=lt.size(me.getValue(mo.get(expediente[llave],maxDuracion)))
    avisRango=mo.keys(expediente[llave],duracionMin,duracionMax)
    listaAvistamientos=lt.newList("SINGLE_LINKED")
    for key in lt.iterator(avisRango):
        duracion=me.getValue(mo.get(expediente[llave],key))
        for avis in lt.iterator(duracion):
            lt.addLast(listaAvistamientos,avis)
    if llave=="duraciones":
        listaAvistamientos=ms.sort(listaAvistamientos,cmpDuration)
    else:
        listaAvistamientos=ms.sort(listaAvistamientos,cmpHoraMin)
    return maxDuracion,numAvistamientosMaxKey,listaAvistamientos
```

Como esta función es exactamente la misma del requerimiento anterior (se adaptó la misma función para que pudiera realizar los requerimientos 2 y 3), las complejidades no cambian, el único factor que hace que este requerimiento sea más costoso temporalmente es la variedad de las llaves, ya que en el requerimiento anterior había muchos más datos que compartían la misma cantidad de duración en segundos, mientras que para este requerimiento es menos común que varios datos compartan tanto la hora, como el minuto y el segundo exacto en los que ocurrieron. Es debido a esto, que los dos requerimientos comparten la misma complejidad temporal aun cuando uno tarda más que el otro.

Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609



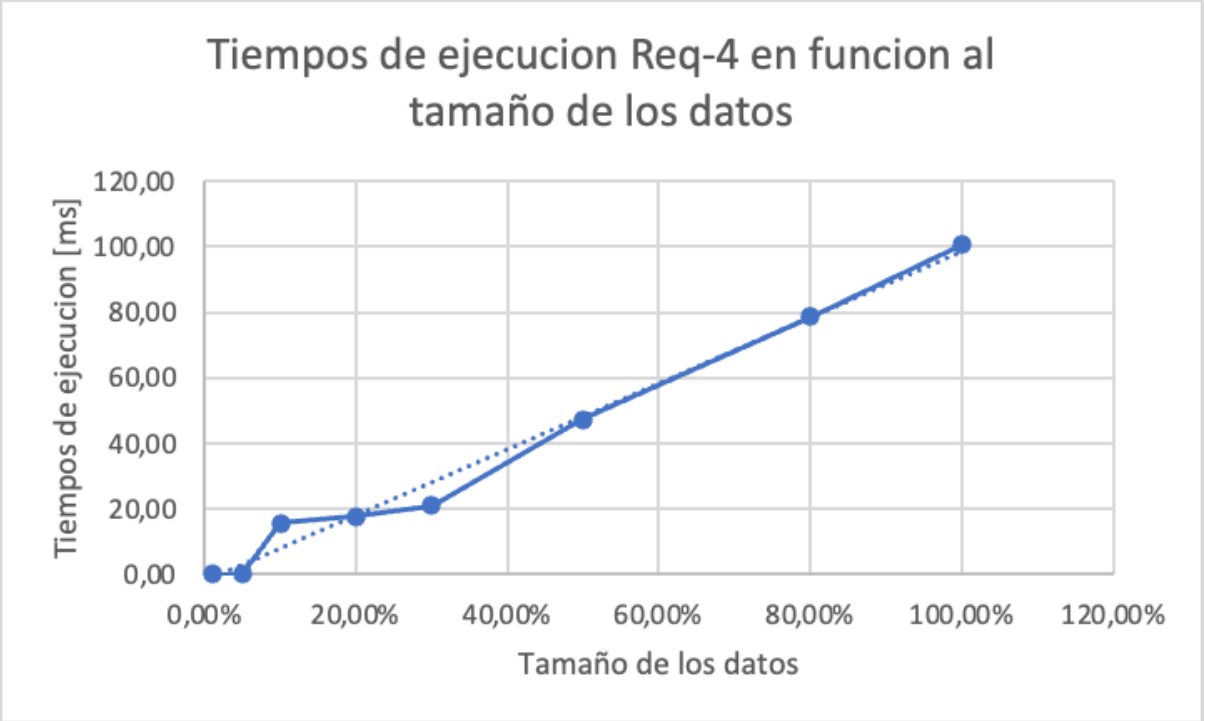
En cuanto a esta grafica se podria decir practicamente lo mismo que sobre el requerimiento #2 ya que se utiliza la misma función a pesar de que se cargan distintos tipos de datos. Es por esta razón que las gráficas tienen un comportamiento tan similar ya que la única variable cambiante es el parámetro. En este caso, también se tendría una gráfica con un comportamiento similar a n*log(n) o cuadrática debido a que se sobrepasa la línea de tendencia y se acerca más hacia el límite superior de una identidad cuadrática. De todas formas y una vez más, no se puede descartar el caso en que la identidad de la gráfica se n*log(n) ya que ignorando el cambio del intervalo final la gráfica tiende a lo esperado según el análisis de complejidad temporal.

Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609

Requerimiento 4 **O((log(n))+(M))**

```
def avistamientosEnRango(expediente,fechaInicio,fechaFin):
    fechaInicio=(datetime.datetime.strptime(fechaInicio, '%Y-%m-%d')).date()
    fechaFin=(datetime.datetime.strptime(fechaFin, '%Y-%m-%d')).date()
    avistamientosFechas=lt.newList("SINGLE_LINKED")
    fechasEnRango=mo.keys(expediente["fechas"],fechaInicio,fechaFin)
    for date in lt.iterator(fechasEnRango):
        avistamientos=me.getValue(mo.get(expediente["fechas"],date))
        for sighting in lt.iterator(avistamientos):
            lt.addLast(avistamientosFechas,sighting)
    numAvistamientos=lt.size(avistamientosFechas)
    return numAvistamientos,avistamientosFechas
```

Para este requerimiento se tiene que: una operación keys() en un mapa ordenado toma log(n) en el peor caso. Adicional a esto, los dos ciclos for recorren todos los avistamientos que caen en el rango de fechas ingresado, por lo que su complejidad es M, siendo M el total de avistamientos que entran al rango. Por esto, la complejidad temporal total de este requerimiento es de O((log(n))+(M))



Acorde a lo que se puede observar en la gráfica, se podría llegar a la conclusión que esta función posee un carácter n*log(n) o en su defecto lineal. Según la notación big O y teniendo en cuenta la línea de tendencia, los datos se inclinan hacia el carácter n*log(n) en el caso de que este sea su límite superior ya que podría estar sobrepasando la tendencia lineal. De todas formas el lento crecimiento de la gráfica y el cambio abrupto en el intervalo inicial podrían

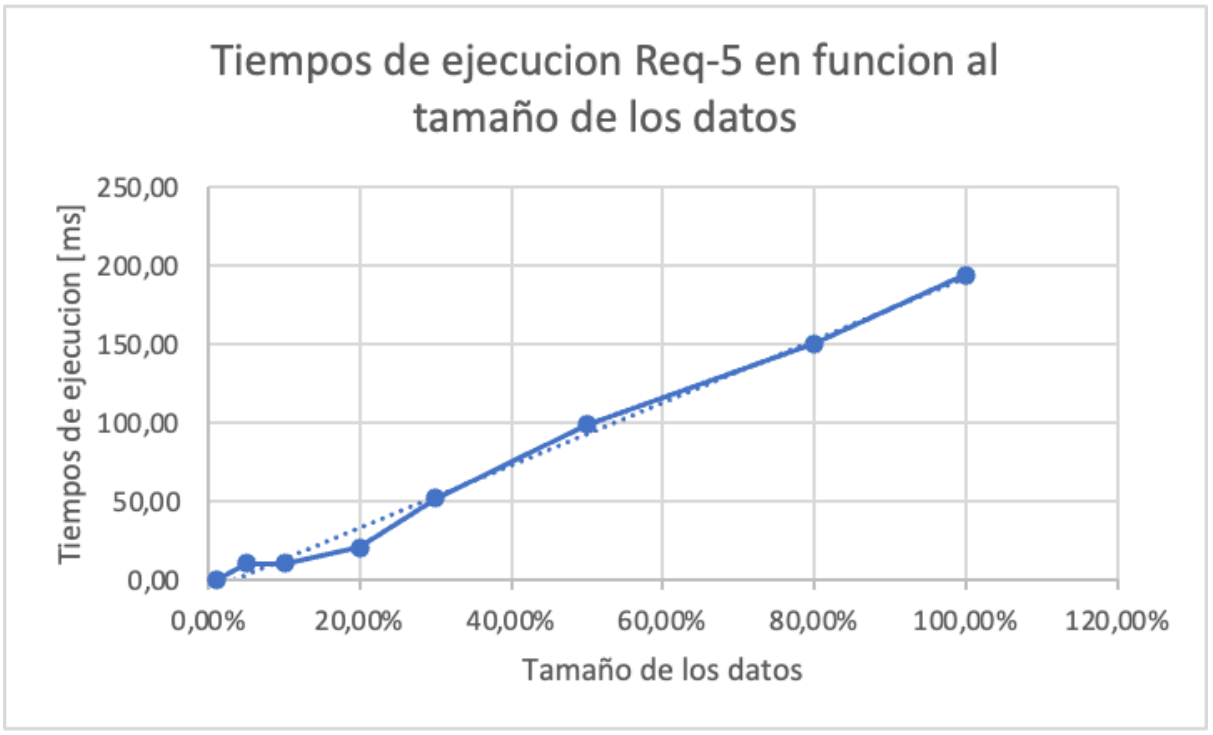
ser indicadores de un comportamiento logarítmico por lo que lo mas seguro sea la opción de una gráfica n*log(n).

Requerimiento 5 **O((log(n))+(M)+(n*log(n)))**

```
def avistamientosZona(expediente,longMin,longMax,latMin,latMax):
    longKeys=mo.keys(expediente["longitudes"],longMin,longMax)
    avistamientosLongLat=lt.newList("SINGLE_LINKED")
    for long in lt.iterator(longKeys):
        lista=me.getValue(mo.get(expediente["longitudes"],long))
        for avis in lt.iterator(lista):
            lat=float(avis["latitude"])
            lat=round(lat,2)
            if latMin<=lat<=latMax:
                lt.addLast(avistamientosLongLat,avis)
    num=lt.size(avistamientosLongLat)
    avistamientosLongLat=ms.sort(avistamientosLongLat,cmpLongLat)
    return avistamientosLongLat,num
```

En este requerimiento se usa nuevamente la operación keys() para un mapa ordenado, cuya complejidad temporal es O(log(n)). Con los dos siguientes ciclos for (el primero que recorre todas las llaves que retorna keys(), es decir las llaves de los avistamientos que entran al rango de longitudes y latitudes, y el segundo que recorre todos los avistamientos de la llave actual del ciclo anterior), se recorren todos los avistamientos que entran al rango de longitudes y al de latitudes ingresados, por lo que, tomando M como el total de dichos avistamientos, se tiene una complejidad de M. Por último, se ordenan dichos avistamientos por longitud y latitud, por lo que, usando un mergesort, se tiene una complejidad de O(n*log(n)). Finalmente se tiene que la complejidad temporal total de este algoritmo es de O((log(n))+(M)+(n*log(n)))

Diego Acosta Corredor - 202110516
Mateo Cote Canal - 202022609



Al analizar esta gráfica se puede observar como tiene un comportamiento lineal, casi entre un comportamiento logarítmico y uno cuadrático (este último más visible con una muestra menor de datos), por lo que es acorde al análisis de complejidad hecho acerca de este requerimiento, lo cual presentaba operaciones con complejidad log(n) y n*log(n).