**ANÁLISIS DEL RETO 2**

Juliana Rodríguez Morales – 202421552 – js.rodriguezm1234

Maria Clara Quijano - 202420069 - m.quijanoa

Juan Andrés Lozada - 202510410-j.lozadab

# **Requerimiento 1**

## **Descripción**

Función req\_1Texto

El contenido generado por IA puede ser incorrecto.

Función auxiliar de sort\_criteria para el quick\_sort del req\_1

Texto

El contenido generado por IA puede ser incorrecto.

A partir de un rango de fechas dado (fecha inicial y final de recogida) se deben mostrar los primeros n viajes y últimos n viajes previamente ordenados del más antiguo al más reciente.

La función de sort criteria compara fecha y hora de recogida para organizar del más antiguo al más reciente al momento de utilizar quick\_sort.

|  |  |
| --- | --- |
| **Entrada** | Fecha y hora inicial de recogida  Fecha y hora final de recogida  Tamaño de muestra |
| **Salidas** | Tiempo de ejecución  # de trayectos dentro del rango de fecha/hora  N primeros y N últimos:  Fecha y hora de inicio  Coordenadas iniciales  Fecha y hora final  Coordenadas finales  Distancia recorrida (millas)  Costo total pagado |
| **Implementado (Sí/No)** | Sí, Juliana Rodríguez |

## **Análisis de complejidad**

Solo se analizará parte por parte el req\_1, no sort\_crit.

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorrer la lista con los trayectos. O(n) | O(n) |
| Obtener elemento de Array | O(1) |
| Comparación de fechas en cada viaje | O(1) - por estar dentro del for -> O(n) |
| Añadir elemento en un Array | O(n) |
| Quick sort | O(m logm) |
| ***TOTAL*** | **O(n + m logm)** |

## **Análisis**

En el caso del req 1 se recorrió toda la lista con los trayectos donde se iban filtrando aquellos que cumplieran los criterios dados como parámetro. Para que el código fuera mucho más eficiente utilizamos array lists ya que al comparar las complejidades de las funciones más usadas en el código array list en promedio tiene mejor complejidad. Es decir, al hacer add\_last con array list hay que tener en cuenta que estos se redimensionan al alcanzar su capacidad máxima, siendo su peor caso O(n) al tener que hacer una lista más grande y copiar los elementos en esta. Sin embargo, esto no sucede en cada llamado de función add\_last, por lo que su complejidad sigue siendo mejor a comparación de un get\_element de single linked list O(n) que se hace todo el tiempo. Y este razonamiento es importante ya que el llamado de get\_element se hace en cada iteración del for y add\_last solo cuando el viaje cumple el criterio.

A lo último del requerimiento, después de filtrar todos los viajes se hizo un quick sort para poder organizar la lista según el criterio de ordenamiento requerido. De todos las funciones de ordenamiento escogimos quick sort ya que, al ser in place no consume mucha memoria y su complejidad temporal era la mejor [al ser O(n logn)]. Para poder hacer el quick sort, se tuvo que modificar la función sort\_crit para que accediera a las llaves necesarias de los trayectos, es decir, la llave a la que se accedía en el nuevo sort crit era el criterio de ordenamiento que se pedía. Esta última función no tiene una complejidad mayor a O(1) pues solo compara dos elementos, por lo que no afecta a la complejidad de quick sort.

# **Requerimiento 2**

## **Descripción**

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.Breve descripción de como abordaron la implementación del requerimiento

|  |  |
| --- | --- |
| **Entrada** | Coordenada inicial de latitud del rango.  Coordenada final de latitud del rango.  Tamaño de la muestra (N) de trayectos a mostrar al principio y al final del rango |
| **Salidas** | Tiempo de ejecución  # de trayectos dentro del rango de fecha/hora  N primeros y N últimos:  Fecha y hora de inicio  Coordenadas iniciales  Fecha y hora final  Coordenadas finales  Distancia recorrida (millas)  Costo total pagado |
| **Implementado (Sí/No)** | Sí. Por Maria Clara Quijano |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Filtrado por rango de latitud | O(n) |
| Comparaciones | O(1) por viaje—> O(n) |
| Ordenar | O(klogk) |
| ***TOTAL*** | ***O(n+klogk)*** |

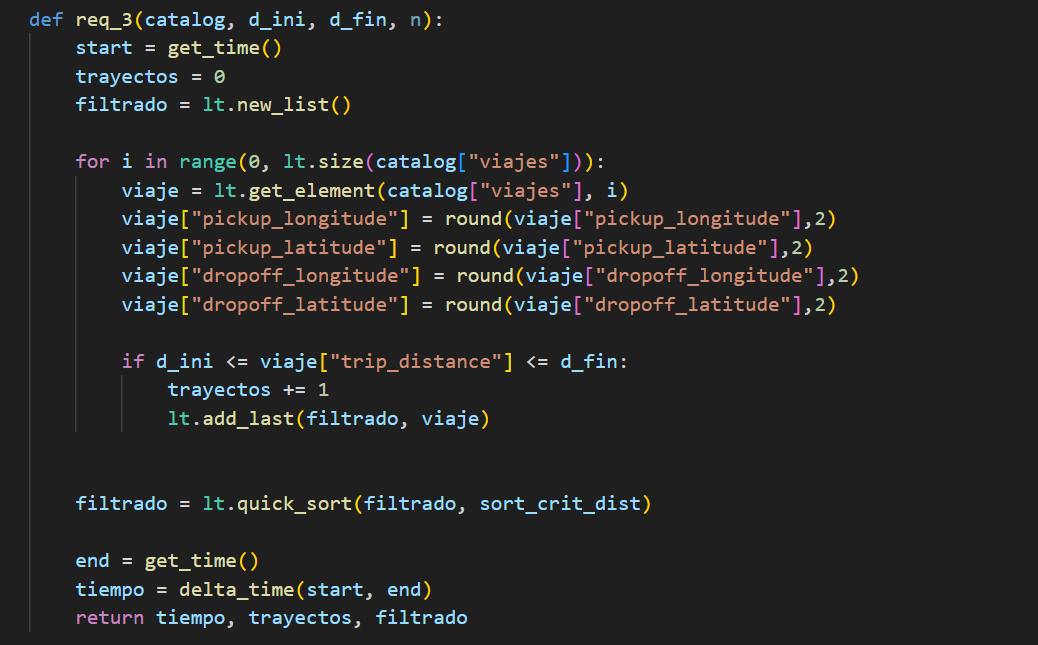
## **Análisis**

Se recorre toda la lista y se utilizan array list porque son más eficientes debido a que puede acceder las posiciones, esto hace que tenga una complejidad de O(1) en promedio. Luego se ordena usando quicksort todo lo que pasa el filtro, el cual tiene una complejidad promedio de O(nlogn), y es eficiente para ordenar datos, incluso cuando hay muchos. Cuando hay un rango de latitud más pequeño, el tiempo tiene un crecimiento lineal, mientras que cuando el rango es mayor tiene mayor crecimiento.

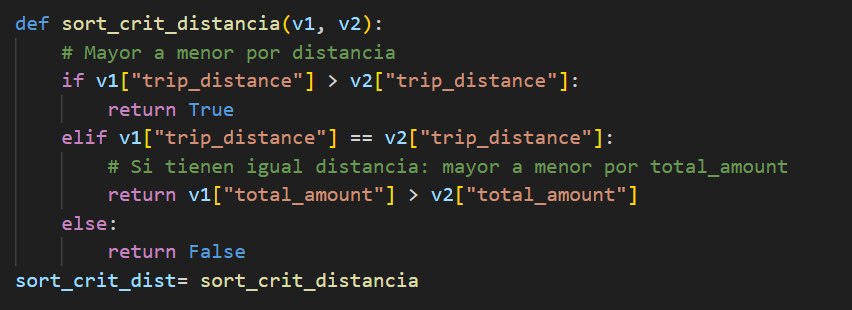
# **Requerimiento 3**

## **Descripción**

Req 3



Función auxiliar de criterio de ordenamiento en el req 3



El **req 3** filtra los trayectos cuya **distancia recorrida** esté entre un valor mínimo y máximo ingresado por el usuario. Luego, aplica una **función de criterio de ordenamiento** que compara los valores de la clave "trip\_distance" para organizar los viajes **de mayor a menor distancia**. Finalmente, muestra el tiempo de ejecución, la cantidad de trayectos encontrados, y presenta los **primeros y últimos viajes** del rango según su distancia.

|  |  |
| --- | --- |
| **Entrada** | Parámetros necesarios para resolver el requerimiento. |
| **Salidas** | Tiempo de ejecución  # de trayectos dentro del rango de fecha/hora  N primeros y N últimos:  Fecha y hora de inicio  Coordenadas iniciales  Fecha y hora final  Coordenadas finales  Distancia recorrida (millas)  Costo total pagado |
| **Implementado (Sí/No)** | Si, Juan Andres Lozada Barragan |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1: Recorrer todos los viajes y filtrar por distancia | O(n) |
| Paso 2: Agregar los viajes que cumplen el filtro a una nueva lista | O(m) |
| Paso 3: Ordenar la lista filtrada con quick sort | O(n log n) |
| ***TOTAL*** | **O(n log n + m)** |

## **Análisis**

El algoritmo tiene una complejidad total **O(n log n)**, dominada por el proceso de ordenamiento.  
El filtrado inicial de trayectos por distancia recorre la lista completa una vez (**O(n)**), mientras que quick sort organiza los resultados según la distancia y el costo, haciendo el proceso eficiente incluso con grandes volúmenes de datos. En pruebas, el tiempo de ejecución crece moderadamente al aumentar el número de trayectos.

# **Requerimiento 4**

## **Descripción**

Función req\_4

Texto

El contenido generado por IA puede ser incorrecto.

Función creación de la Tabla Hash para el req\_4

Texto

El contenido generado por IA puede ser incorrecto.

Función sort\_crit para la organización en el quick\_sort del req\_4

Texto

El contenido generado por IA puede ser incorrecto.

A partir de una fecha de terminación de trayecto dada debo filtrar todos los viajes que hayan terminado en esa fecha y mostrar todos los trayectos antes o después (criterio dado por el usuario) de una hora de terminación dada. Deben mostrarse los primeros n trayectos y últimos n trayectos previamente ordenados del más reciente al más antiguo.

La función mapa\_req4 crea la tabla de hash a utilizar para el cumplimiento del req\_4. La llaves son las fechas de terminación de los trayectos, y su valor un array\_list con todos los viajes que terminaron en esa fecha.

La función de sort criterio viajes compara fecha y hora de recogida para organizar del más antiguo al más reciente al momento de utilizar quick\_sort.

|  |  |
| --- | --- |
| **Entrada** | Fecha de terminación  Criterio de comparación (ANTES, DESPUES)  Tiempo de terminación  Tamaño de muestra |
| **Salidas** | Respuesta Tiempo de ejecución  # de trayectos dentro del rango de fecha/hora  N primeros y N últimos:  Fecha y hora de inicio  Coordenadas iniciales  Fecha y hora final  Coordenadas finales  Distancia recorrida (millas)  Costo total pagado |
| **Implementado (Sí/No)** | Sí, Juliana Rodríguez |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Carga del mapa | O(n) |
| Obtener lista (mp.get) | O(1) |
| Recorrido de lista (for) | O(m) |
| Obtener elemento de Array | O(1) |
| Comparación Antes/Después | O(1) |
| Agregar elemento a un Array | O(m) |
| Quick Sort | O(k logk) |
| ***TOTAL*** | ***O(n + m + k log k)*** |

## **Análisis**

La función de la carga del mapa se da al comienzo del req4, la razón de utilizar una tabla de hash, sin tener en cuenta que es una restricción, es para que no se tenga que recorrer toda la lista de viajes nuevamente. Sin embargo, dado que la creación de mapas no se pudo poner en el load data en sí mapa\_req4 recorre todo el catálogo de viajes también pone cada viaje en cierto lugar del mapa, lo que afecta la complejidad del req 4. Calculado, su complejidad es de O(n)

Ya en el código propiamente del req4 se utiliza un for que recorre toda la lista obtenida como valor de la llave que tiene la misma fecha como la del parámetro. Aunque utiliza funciones del mapa que no afectan mucho su complejidad, se determina que el for tiene complejidad de O(m).

Por último, se utilizó un quick sort para organizar la lista de los trayectos, y como se mencionó anteriormente, quick\_sort tiene una complejidad de O(m logm), sin embargo como la lista que organiza solo tiene los elementos antes o después de la hora de terminación, su complejidad va por otra variable O(k logk).

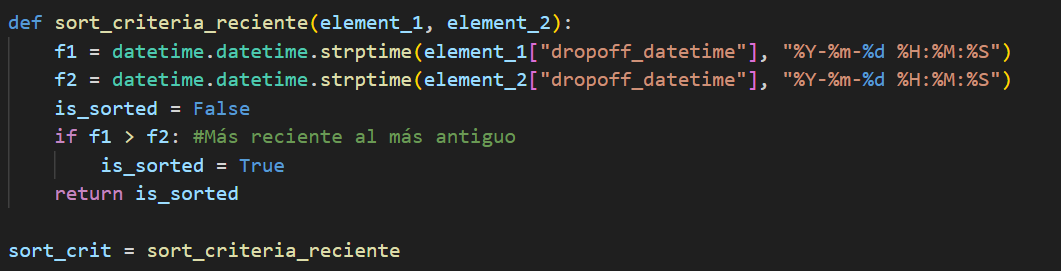
# **Requerimiento 5**

## **Descripción**

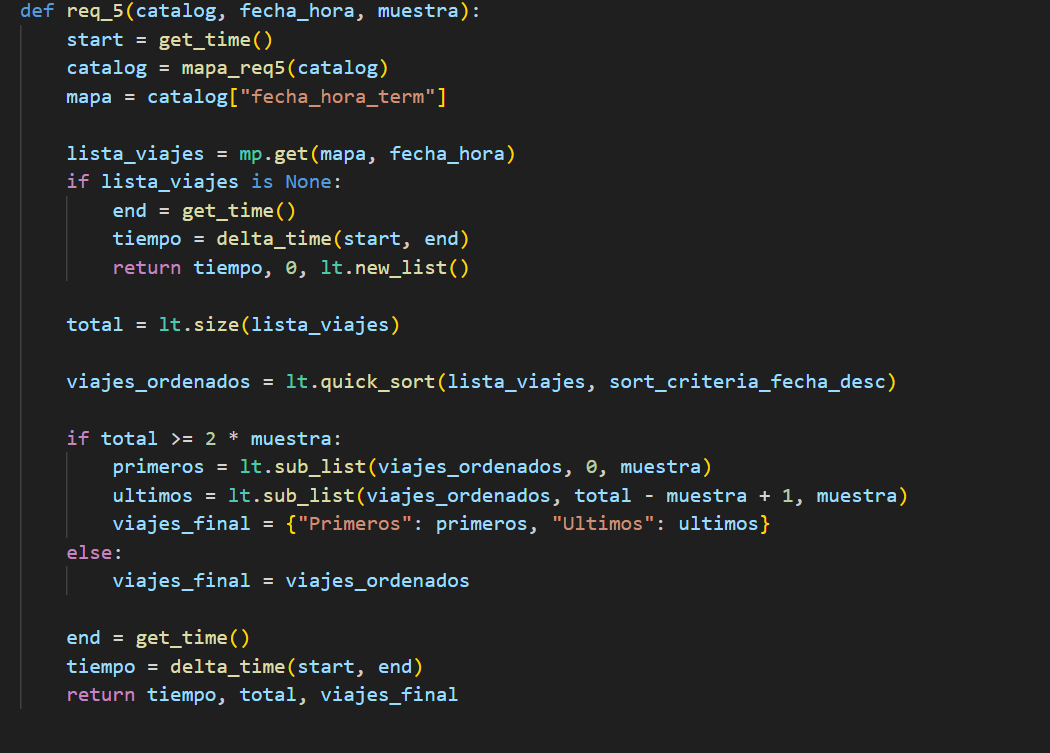
Función auxiliar de creacion del mapa del req 5.



Función de sort criteria del req 5



Función del req 5



Se implementó una tabla hash para agrupar los viajes según su fecha y hora de terminación, facilitando búsquedas rápidas. Luego, se ordenaron los trayectos de esa hora con Quick Sort, del más reciente al más antiguo, y se mostraron los primeros y últimos *N* resultados solicitados.

|  |  |
| --- | --- |
| **Entrada** | Parámetros necesarios para resolver el requerimiento. |
| **Salidas** | Tiempo de ejecución  # de trayectos dentro del rango de fecha/hora  N primeros y N últimos:  Fecha y hora de inicio  Coordenadas iniciales  Fecha y hora final  Coordenadas finales  Distancia recorrida (millas)  Costo total pagado |
| **Implementado (Sí/No)** | Si, Juan Andrés Lozada Barragan |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1: Crear el mapa hash con las fechas y horas de terminación | O(n) |
| Paso 2: Buscar la llave correspondiente a la hora dada | O(1) |
| Paso 3: Ordenar los trayectos por fecha de terminación (quick sort) | O(m log m) |
| Paso 4: Seleccionar los primeros y últimos *N* trayectos | O(N) |
| ***TOTAL*** | **O(n + N + m log m)** |

## **Análisis**

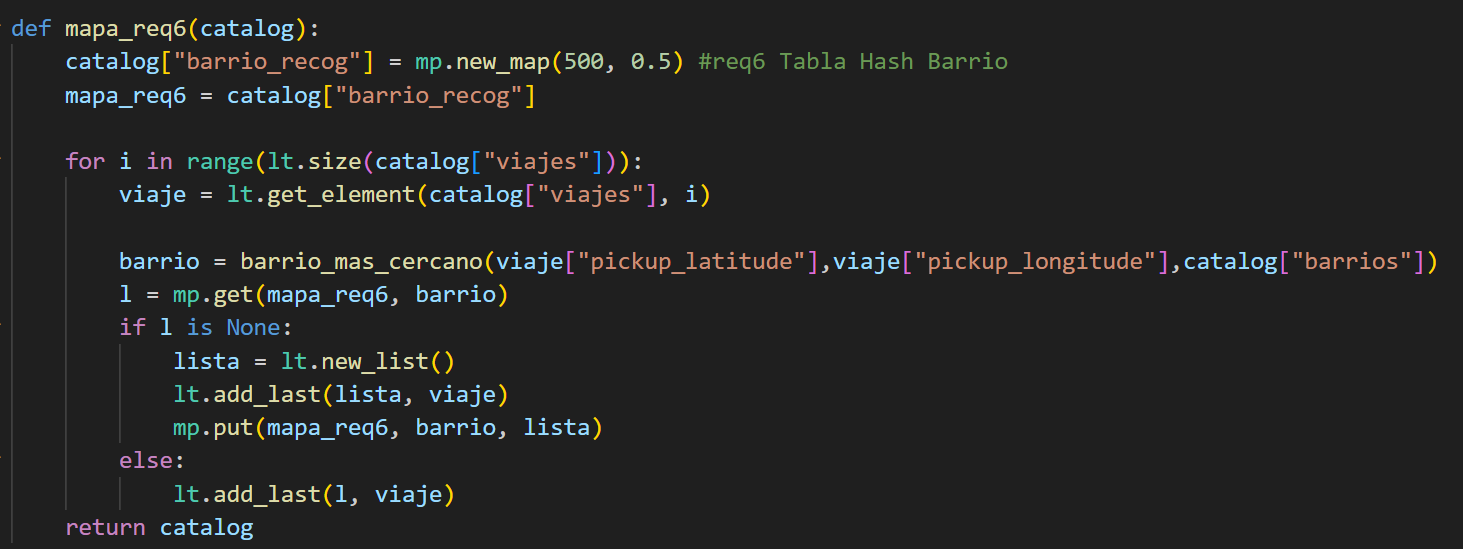
El algoritmo crea una tabla hash que agrupa los viajes según su hora de terminación, permitiendo acceder a los trayectos de una hora específica en tiempo casi constante. Luego, los ordena de más reciente a más antiguo mediante Quick Sort.  
 Su complejidad total es **O(n + m log m)** y, en las pruebas, mostró un tiempo de respuesta rápido incluso con grandes volúmenes de datos, gracias al uso eficiente de la tabla hash.

# **Requerimiento 6**

## **Descripción**

Función req6Texto

El contenido generado por IA puede ser incorrecto.

Función creación tabla hash

Función Sort\_Crit

Texto

El contenido generado por IA puede ser incorrecto.

|  |  |
| --- | --- |
| **Entrada** | Barrio de recogida (ej.: “Tribeca”, “Midtown”).  • Hora inicial del rango de recogida (formato "%H" ej.:"09").  • Hora final del rango de recogida (formato "%H" ej.:"12")  • Tamaño de la muestra (N) de trayectos a mostrar al principio y al final del rango |
| **Salidas** | · Tiempo de la ejecución del requerimiento en milisegundos.  · Número total de trayectos que cumplieron el filtro de barrio y hora de recogida.  · Mostrar la siguiente información de cada uno de los N primeros trayectos y de los N últimos trayectos en el rango:  o Fecha y tiempo de recogida  o Latitud y longitud de recogida  o Fecha y tiempo de terminación  o Latitud y longitud de terminación  o Distancia recorrida (millas)  o Costo total pagado |
| **Implementado (Sí/No)** | Sí, Maria Clara Quijano |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Mapa\_req6 | O(n) |
| Filtrado por hora | O(m) |
| Comparación | O(k) |
| Ordenamiento filtrados | O(klogk)) |
| Retorno | O(1) |
| ***TOTAL*** | ***O(n+klogk)*** |

## **Análisis**

Se utiliza una función auxiliar que utlizan mapas hash para agrupar los viajes por barrio de recogida ya que es más fácil de encontrar el barrio que se está buscando con complejidad O(1). Luego se utiliza un array list para guardar los viajes para que sea más fácil de acceder a los indicen con menor complejidad. Adicionalmente, se pueden usar las funciones del array list creadas para facilitar agregar elementos, eliminar y más. Con el quicksort logramos ordenar los viajes filtrados del más antigua al más reciente con una complejidad de solo O(nlogn) y de manera rápida.