**Héctor Gutierrez**

**A01253031**

**Reflexión Actividad 2.3**

En el contexto del problema el uso de listas enlazadas es muy bueno, ya que permiten una inserción y eliminación eficientes de elementos en cualquier punto de la lista. Esto es de suma importancia cuando se están procesando registros de datos y se necesita agregar o eliminar elementos en función de ciertos criterios, además de que las listas enlazadas o encadenas pueden crecer o reducirse dinámicamente según sea necesario.

Las ventajas de las listas doblemente ligadas permiten insertar y eliminar elementos de una manera más eficiente en cualquier posición de la lista, ya que se puede acceder rápido al nodo anterior como al nodo siguiente, lo cual en este caso es muy útil porque estamos insertando nodos en medio de la lista. Otra ventaja de las listas doblemente ligadas es que tienen una ordenación eficiente, especialmente cuando se utiliza el algoritmo de mergesort, por lo tanto, esto si nos hubiera ayudado mucho con respecto al ordenar los datos.

Las desventajas de las listas doblemente ligadas es que ocupan más memoria que las listas simples porque tienen punteros adicionales para los nodos anteriores y también si se llegara a necesitar acceder a elementos específicos de la lista, las listas doblemente ligadas no son lo mejor porque se tiene que recorrer la lista desde el principio o el final para obtener el elemento deseado.

La función de “compareRecord” tiene una complejidad de tiempo de O (1), ya que solo realiza una cantidad constante de operaciones independiente del tamaño de los registros, esta función se utiliza para comparar dos registros de buques y ordenarlos por fecha y ubicación en orden ascendente. La función de “processRecord” depende del número de registros en la lista enlazada, el while recorre todos los registros, por lo tanto, es O(n). Luego dentro del while se involucran búsquedas y asignaciones en el mapa, por lo tanto tiene una complejidad de O (log n) para la búsqueda y O(1) para la asignación, por lo tanto la función en general tiene una complejidad de O(n log n). La función de “processRecord” es para procesar los registros de buques y actualizar el mapa con la cantidad de entradas de buques por mes, también sirve para verificar si la serie a buscar existe en la lista de registros de buques.

La función “deleteList” tiene una complejidad de O(n) porque como elimina todos los nodos de la lista enlazada, esto implica visitar cada nodo una vez solamente. La función de “binarySearch” tiene una complejidad de O(log n), ya que se divide a la mitad en cada iteración de la función. La función de “imprimirCantidadEntradasPorMes” tiene una complejidad de O(n), ya que recorre la lista una vez para extraer el mes y el año de cada fecha y luego recorre el mapa una vez para imprimir la cantidad de entradas de buques por mes. Y ya por último la función de “mergeSort” tiene una complejidad de O (n log n) porque primero en la función de “getMiddle” si divide la lista en dos mitades, lo cual es O(n/2), luego en ambas mitades se ordenan recursivamente llamando a mergeSort. En merge combina dos listas ordenadas y esta requiere un tiempo lineal en función del numero total de elemento en ambas listas, es decir O(n), es porque que mergeSort es O(n log n) por sus funciones complementarias. Y ya para finalizar con la última función que es “sortList” tiene una complejidad de O (n log n) porque utiliza el algoritmo Merge Sort, ya que esta función simplemente llama a “mergeSort”