Tecnológico de Monterrey

REFLEXIÓN - ACT 2.3

DANTE HERNÁNDEZ RAMÍREZ - A01668070

Programación de estructuras de datos y algoritmos fundamentales

(Gpo 573)

23 de Enero del 2026

Reflexión

El programa está basado en la resolución del problema de la actividad integral 1.4, se tomaron en cuenta los mismos 16 mil registros. Se realizaron cambios desde la entrega de la última entrega de la misma, se sustituyó la implementación del vector, y se implementó una lista ligada doble en su lugar.

A pesar de que el código tenía el mismo objetivo, las implementaciones y las modificaciones realizadas cambiaron totalmente la lógica con la cuál se realizó como equipo. La comparación entre algoritmos de ordenamiento también afecta la complejidad total del código.

A continuación se desglosan algunos de los cambios más significativos de las diferentes clases para el funcionamiento del proyecto:

BITÁCORA:

- Cada registro se agrega con addFirst() a la lista ligada doble antes mencionada. Quedando los datos invertidos en el archivo.
- Se separan los datos con Istringstream. Se separan en hora, y en IP y puerto usando find(“:”) y substr().

ESTRUCTURAS DE DATOS - DOUBLY LINKED LIST:
Se cambia el vector por la lista doblemente ligada: Head y Tail con sus respectivos nodos. alterando la complejidad del código:

- addFirst es 0(1).
- El recorrido para imprimir es 0(n).
- A diferencia de la actividad anterior, no existen índices.

En este caso, en una lista ligada doble, los algoritmos de ordenamiento son QuickSort y BinarySearch sufren una pequeña alteración: debido a la implementación de nodos, el costo y la complejidad sube.

Aún con eso, la lista ligada tiene a su favor la flexibilidad en memoria y en operaciones de inserción o eliminación, lo cuál es efectivamente buena para el algoritmo merge.

ESTRUCTURAS DE DATOS - QUICKSORT:

- Usa un swap para dirigir los datos

El funcionamiento del algoritmo es:

1. Se toma el pivote
2. Se recorre el algoritmo desde low hasta high
3. Se coloca el pivote en su posición final
4. Se ordena recursivamente de izquierda a derecha

En promedio, el algoritmo suele ser 0(n log(n)), siendo su peor caso 0(n^2) si el pivote queda al extremo.

BÚSQUEDA BINARIA:

Se usan dos funciones:

- binarySearchNode(target) (encuentra una coincidencia “hacia el inicio”)
- binarySearchLastNode(target) (tiende a ir hacia la última coincidencia)

En sí, la búsqueda binaria sigue bajo la lógica de que si está ordenado, se puede descartar media lista al momento de hacer cada iteración, siendo su complejidad 0(n log(n)), siendo los siguientes casos:

- Cuando se llega a la mitad es 0(1)
- En el caso de la lista ligada, hallar la mitad cuesta 0(n)