Problema B

Juan Felipe Rubio Perdomo – 201718384
Jf.rubio@uniandes.edu.co
Juan Andrés Avelino Ospina – 201812676
ja.avelino@uniandes.edu.co

I Algoritmo de solución:

Para este problema, se desarrolló su solución dentro del main de la clase java. El problema se resolvió con dos algoritmos específicos. En primer lugar, se lee la cantidad N y se prepara para leer el resto de líneas, teniendo para la linea k-sima un valor k.a, k.b y k.índice. En este momento, se utiliza una clase embebida que almacena los tres valores y además implementa la interfaz Comparable para poder proceder a organizarlos con el método Collections.sort() de la clase Collections. Esto se realiza con la finalidad de obtener un arreglo ordenado por el valor "a" que en el caso de tener valores iguales empata con la denominación de "b" y después por índice. Después de realizar este ordenamiento, el problema se termina de solucionar con el famoso algoritmo de LIS. Es decir, "Longest Increasing Subsecuence". Sin embargo, se adapta para que se recorra decresientemente el arreglo.

Ya sin más, se imprime la longitud del contraejemplo maximal y en la siguiente linea los índices del contraejemplo maximal.

Anotación del desarrollo del método de resolución del problema: (esto se hizo basado en todo el código realizado en el main)

Contexto: A lo largo de toda la ejecución del problema se tendrá un arreglo de `Ordenador` que recibirá los datos ingresados desde consola y separará los valores haciendo uso del `split` por el criterio de "". Por otro lado, se tendrán las variables iniciales para hacer el recorrido del arreglo y lograr calcular la longitud del contraejemplo maximal. Por otro lado, se tendrá la variable `len` que contendrá el tamaño y el arreglo `tailTable` que tendrá los valores del contraejemplo. Sin embargo, es en `arre` dónde se guardan los índices que al final serán impresos en consola.

Precondición: Al inicio de la ejecución se tendrá un arreglo con valores ingresados por consola y de tamaño n definido también desde consola.Así mismo se tendrán todas las variables explicadas anteriormente para ser tratadas a lo largo del programa. Se verán así:

```
ArrayList<Ordenador> ord = new ArrayList<Ordenador>();
int n = Integer.parseInt(arregloRecibido[n-1]);
String[] data = br.readLine().split(" ");
ord.add(new     Ordenador(Integer.parseInt(data[0]),
Integer.parseInt(data[1]), Integer.parseInt(data[2])));
```

Cota: El programa consiste en diferentes cotas. En primer lugar, se tiene la primera cota que evalua que el programa siga recibiendo casos de prueba. Por otro lado, en la segunda parte del algoritmo se ejecuta un ordenamiento por lo que se evaluán todos los elementos del arreglo y la cota sería N y en último lugar se tiene la ejecución del algoritmo LIS que también tiene como cota N.

Avance y Ejecución: El avance en el recorrido será i—pues la posición inicial es el primer elemento del arreglo y se seguirá aumentando recorriendo todos los elementos.

Postcondición: Se espera que al final de la ejecución las variables declaradas inicialmente hayan cambiado su valor de tal forma que el atributo de candidato declarado a lo largo del programa, corresponda a la longitud máxima del contraejemplo maximal. Por otro lado, se espera que el arreglo arre tenga la información de los índices en orden que pertenecen a este contraejemplo maximal.

II Análisis de complejidades espacial y temporal:

Una vez implementado el progama, se procedió a calcular la complejidad temporal y espacial de la solución teniendo en cuenta los algoritmos utilizados y la manera en que se resolvió el problema, obteniendo lo siguiente:

$T(n) = O(n \log n)$

$S(n) = O(n)$

Estas complejidades se calculan teniendo en cuenta las operaciones que se realizan en el desarrollo del programa que resuelve el problema. Partiendo del hecho de que se crea un arreglo basado en los números ingresados en consola, este tendrá un tamaño n. Por otro lado, para resolver el problema se debe de recorrer el arreglo completamente.

De esta manera, el problema representa un ordenamiento en el cual se recorre todo el arreglo, hasta terminar satisfactoriamente con una complejidad de $O(n \log n)$ ya que el sort que se utiliza es QuickSort. Esto representa una complejidad temporal de $O(n \log n)$ en el primer algoritmo del problema. Sin embargo, la segunda parte también tiene una complejidad que sería también de $O(n \log n)$ debido a que LIS utiliza búsqueda binaria. Por ende, tendríamos una complejidad de $O(2n \log n)$ y por reglas de Big O notation no tenemos en cuenta la constante.

Para tener en cuenta, la complejidad espacial es sencilla de calcular ya que tenemos que retornar los índices de el contraejemplo más grande. Que en el peor caso puede ser de tamaño n. Y para guardar los índices se utilizó un arreglo por ende la complejidad sería de $O(n)$.

III Comentarios finales:

Al ejecutar el algoritmo varias veces con diferentes casos y entradas, se encontró que el tiempo de ejecución es efectivo y funciona de acorde a lo esperado.