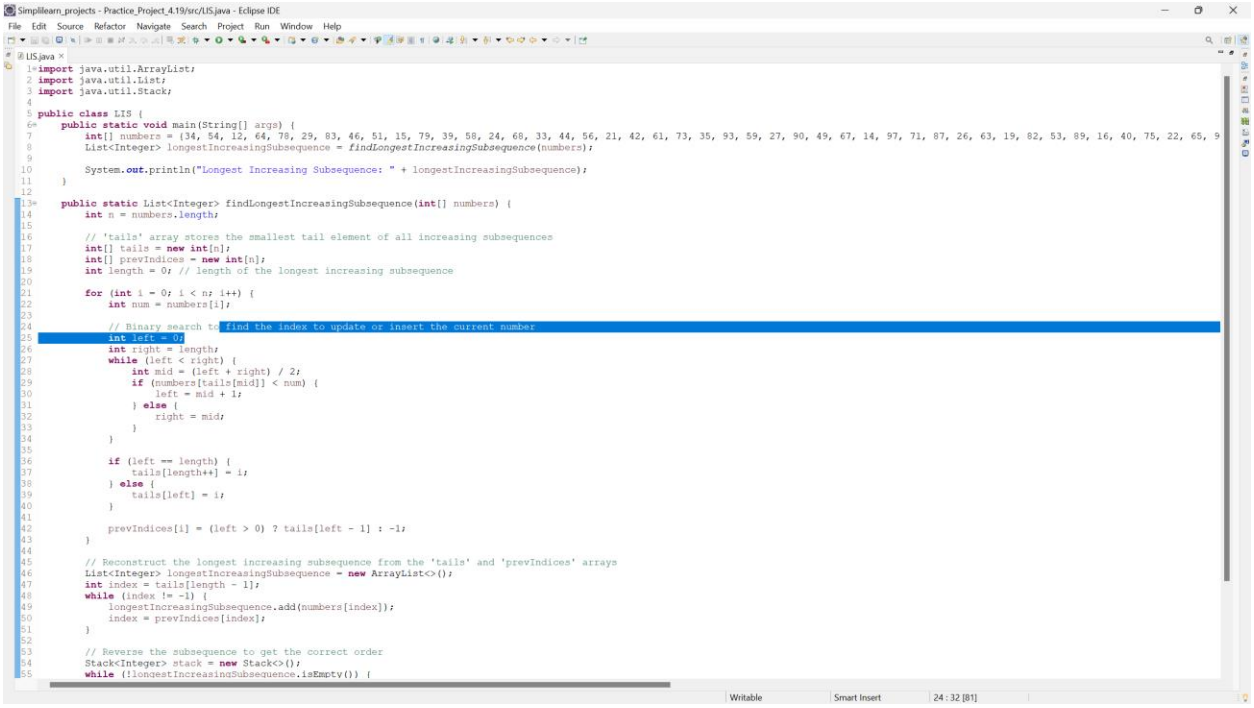
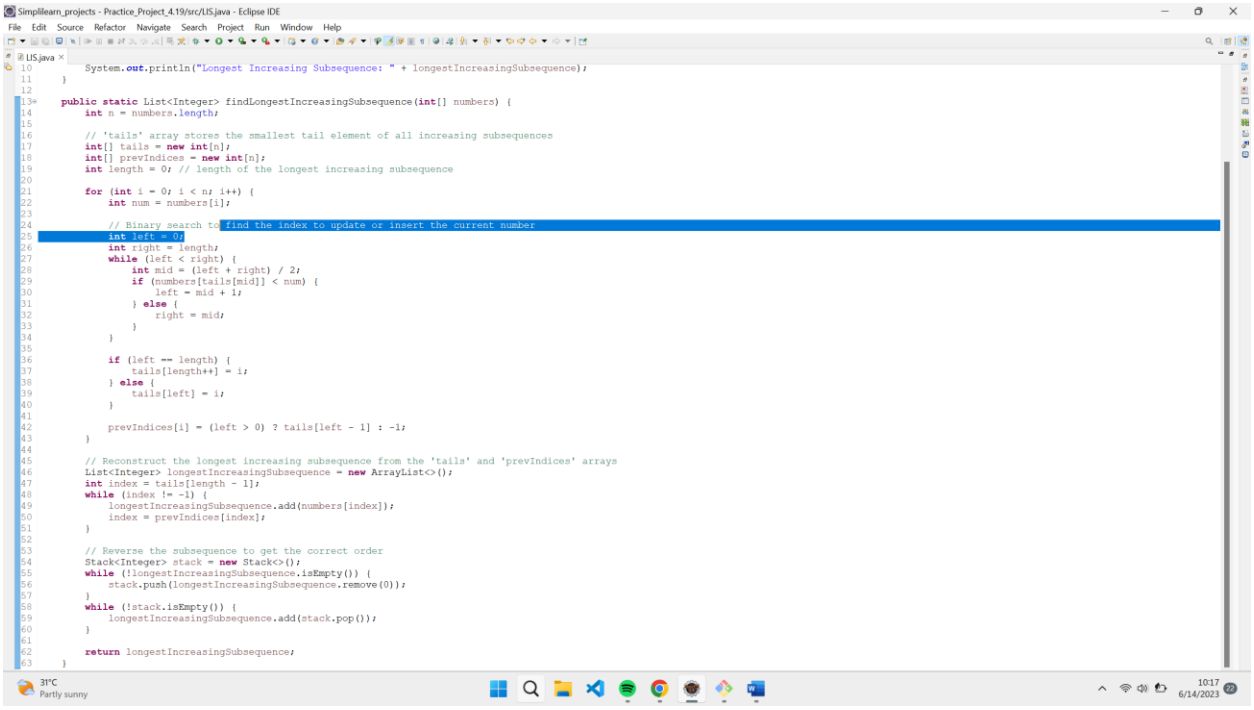



```java
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class LIS {
    public static void main(String[] args) {
        int[] numbers = {34, 54, 12, 64, 78, 29, 83, 46, 51, 15, 79, 39, 58, 24, 68, 33, 44, 56, 21, 42, 61, 73, 35, 93, 59, 27, 90, 49, 67, 14, 97, 71, 87, 26, 63, 19, 82, 53, 89, 16, 40, 75, 22, 65, 9
        List<Integer> longestIncreasingSubsequence = findLongestIncreasingSubsequence(numbers);

        System.out.println("Longest Increasing Subsequence: " + longestIncreasingSubsequence);
    }

    public static List<Integer> findLongestIncreasingSubsequence(int[] numbers) {
        int n = numbers.length;

        // 'tails' array stores the smallest tail element of all increasing subsequences
        int[] tails = new int[n];
        int[] prevIndices = new int[n];
        int length = 0; // length of the longest increasing subsequence

        for (int i = 0; i < n; i++) {
            int num = numbers[i];

            // Binary search to find the index to update or insert the current number
            int left = 0;
            int right = length;
            while (left < right) {
                int mid = (left + right) / 2;
                if (numbers[tails[mid]] < num) {
                    left = mid + 1;
                } else {
                    right = mid;
                }
            }

            if (left == length) {
                tails[length++] = i;
            } else {
                tails[left] = i;
            }

            prevIndices[i] = (left > 0) ? tails[left - 1] : -1;
        }

        // Reconstruct the longest increasing subsequence from the 'tails' and 'prevIndices' arrays
        List<Integer> longestIncreasingSubsequence = new ArrayList<>();
        int index = tails[length - 1];
        while (index != -1) {
            longestIncreasingSubsequence.add(numbers[index]);
            index = prevIndices[index];
        }

        // Reverse the subsequence to get the correct order
        Stack<Integer> stack = new Stack<>();
        while (!longestIncreasingSubsequence.isEmpty()) {
```




```java
        System.out.println("Longest Increasing Subsequence: " + longestIncreasingSubsequence);
    }

    public static List<Integer> findLongestIncreasingSubsequence(int[] numbers) {
        int n = numbers.length;

        // 'tails' array stores the smallest tail element of all increasing subsequences
        int[] tails = new int[n];
        int[] prevIndices = new int[n];
        int length = 0; // length of the longest increasing subsequence

        for (int i = 0; i < n; i++) {
            int num = numbers[i];

            // Binary search to find the index to update or insert the current number
            int left = 0;
            int right = length;
            while (left < right) {
                int mid = (left + right) / 2;
                if (numbers[tails[mid]] < num) {
                    left = mid + 1;
                } else {
                    right = mid;
                }
            }

            if (left == length) {
                tails[length++] = i;
            } else {
                tails[left] = i;
            }

            prevIndices[i] = (left > 0) ? tails[left - 1] : -1;
        }

        // Reconstruct the longest increasing subsequence from the 'tails' and 'prevIndices' arrays
        List<Integer> longestIncreasingSubsequence = new ArrayList<>();
        int index = tails[length - 1];
        while (index != -1) {
            longestIncreasingSubsequence.add(numbers[index]);
            index = prevIndices[index];
        }

        // Reverse the subsequence to get the correct order
        Stack<Integer> stack = new Stack<>();
        while (!longestIncreasingSubsequence.isEmpty()) {
            stack.push(longestIncreasingSubsequence.remove(0));
        }
        while (!stack.isEmpty()) {
            longestIncreasingSubsequence.add(stack.pop());
        }

        return longestIncreasingSubsequence;
    }
```