Assignment-Lab

Student Name: Anupreet Kaur **UID: 22BCS50071**

Branch: BE-CSE **Section/Group: _NTPP_IOT-602-A**

Semester: 6th **Date of Performance: 03/03/2025**

Subject Name: AP Lab **Subject Code: 22CSP-351**

1. Aim: Problems

- ❖ Problem 1.2.1: Longest Substring Without Repeating Characters
- ❖ Problem 1.2.2: Reverse Linked List II
- ❖ Problem 1.2.2: Longest Increasing Subsequence II

2. Objective:

To find the length of the longest substring without repeating characters in a given string. To reverse a portion of a singly linked list between positions left and right.
To find the length of the longest strictly increasing subsequence in an integer array.

3. Code:

Longest Substring Without Repeating Characters:

```
import java.util.HashSet;

public class LongestSubstring {
    public static int lengthOfLongestSubstring(String s) {
        HashSet<Character> set = new HashSet<>();
        int maxLen = 0, left = 0;

        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left));
                left++;
            }
            set.add(s.charAt(right));
            maxLen = Math.max(maxLen, right - left + 1);
        }
        return maxLen;
    }

    public static void main(String[] args) {
        String s1 = "abcabcbb";
        System.out.println("Output: " + lengthOfLongestSubstring(s1));  // Output: 3

        String s2 = "bbbbb";
        System.out.println("Output: " + lengthOfLongestSubstring(s2));  // Output: 1
    }
}
```

Reverse Linked List II:

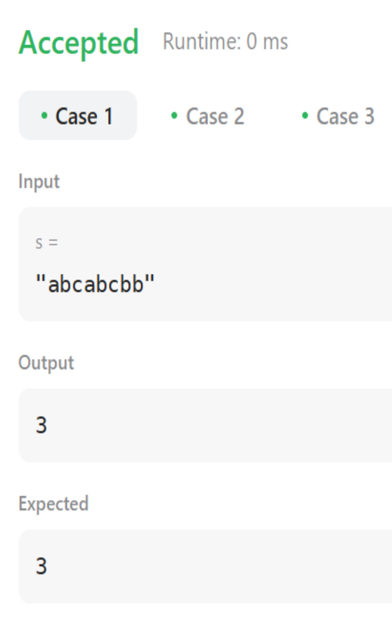
```
    class Solution {
        public ListNode reverseBetween(ListNode head, int left, int right) {
            // Edge case: if left == right, return the head as no reversal is needed
            if (left == right) {
                return head;
            }

            // Step 1: Create a dummy node to simplify edge cases like head being
    reversed
            ListNode dummy = new ListNode(0);
            dummy.next = head;

            // Step 2: Move the prev pointer to the node just before the "left" position
            ListNode prev = dummy;
            for (int i = 1; i < left; i++) {
                prev = prev.next;
            }

            // Step 3: Start the reversal from the "left" position
            ListNode curr = prev.next;  // Current node at "left"
            ListNode next = null;

            // Reverse the sublist between left and right
            for (int i = 0; i < right - left; i++) {
                next = curr.next;
                curr.next = next.next;
                next.next = prev.next;
                prev.next = next;
            }

            // Step 4: Return the new head, which is the next of dummy
            return dummy.next;
        }
```

```
}
```

Longest Increasing Path in a Matrix:

```
class Solution {
    public int lengthOfLIS(int[] nums, int k) {
        int maxNum = 0;
        for (int num : nums) {
            maxNum = Math.max(maxNum, num);
        }

        int[] segTree = new int[4 * (maxNum + 1)]; // Segment Tree

        int maxLen = 0;
        for (int num : nums) {
            int longestPrev = query(segTree, 0, maxNum, num - k, num - 1, 0);
            update(segTree, 0, maxNum, num, longestPrev + 1, 0);
            maxLen = Math.max(maxLen, longestPrev + 1);
        }

        return maxLen;
    }

    private int query(int[] segTree, int left, int right, int ql, int qr, int index) {
        if (ql > right || qr < left) return 0; // No overlap
        if (ql <= left && qr >= right) return segTree[index]; // Full overlap

        int mid = (left + right) / 2;
        return Math.max(
            query(segTree, left, mid, ql, qr, 2 * index + 1),
            query(segTree, mid + 1, right, ql, qr, 2 * index + 2)
        );
    }

    private void update(int[] segTree, int left, int right, int pos, int value, int index)
{
        if (left == right) {
            segTree[index] = value;
            return;
        }

        int mid = (left + right) / 2;
        if (pos <= mid) update(segTree, left, mid, pos, value, 2 * index + 1);
        else update(segTree, mid + 1, right, pos, value, 2 * index + 2);

        segTree[index] = Math.max(segTree[2 * index + 1], segTree[2 * index + 2]);
    }
}
```

6. Output:



Input

s =

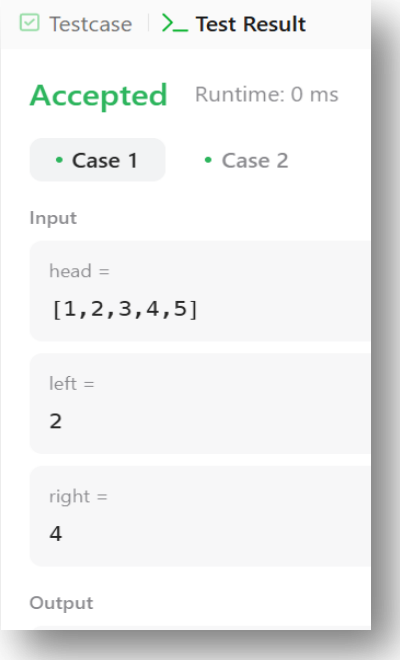
"abcabcbb"

Output

3

Expected

3



Input

head =

[1,2,3,4,5]

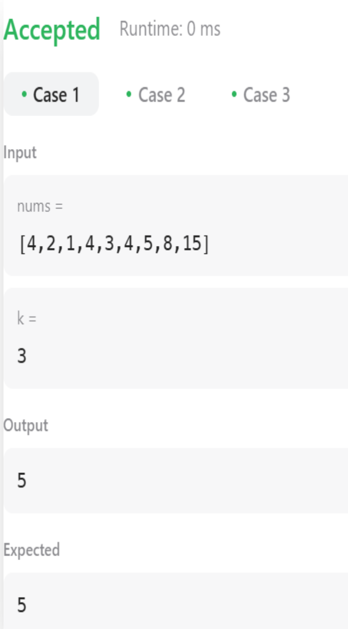
left =

2

right =

4

Output



Input

nums =

[4,2,1,4,3,4,5,8,15]

k =

3

Output

5

Expected

5

7. Learning Outcomes:

- Learned to apply sliding window technique for substring problems.
- Learned how to reverse a sublist in a singly linked list using pointer manipulation.
- Gained understanding of solving subsequence problems using dynamic programming.
- Learned to compare and store results using a `dp` array for optimal substructure problems.