DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 4

Student Name: Armaan Preet Singh **UID: 22BCS10433**

Branch: BE-CSE **Section/Group: 22BCS_IOT-606/B**

Semester: 6th **Date of Performance:13/03/25**

Subject Name: Advanced programming Lab II **Subject Code: 22CSP-351**

PROBLEM 1:

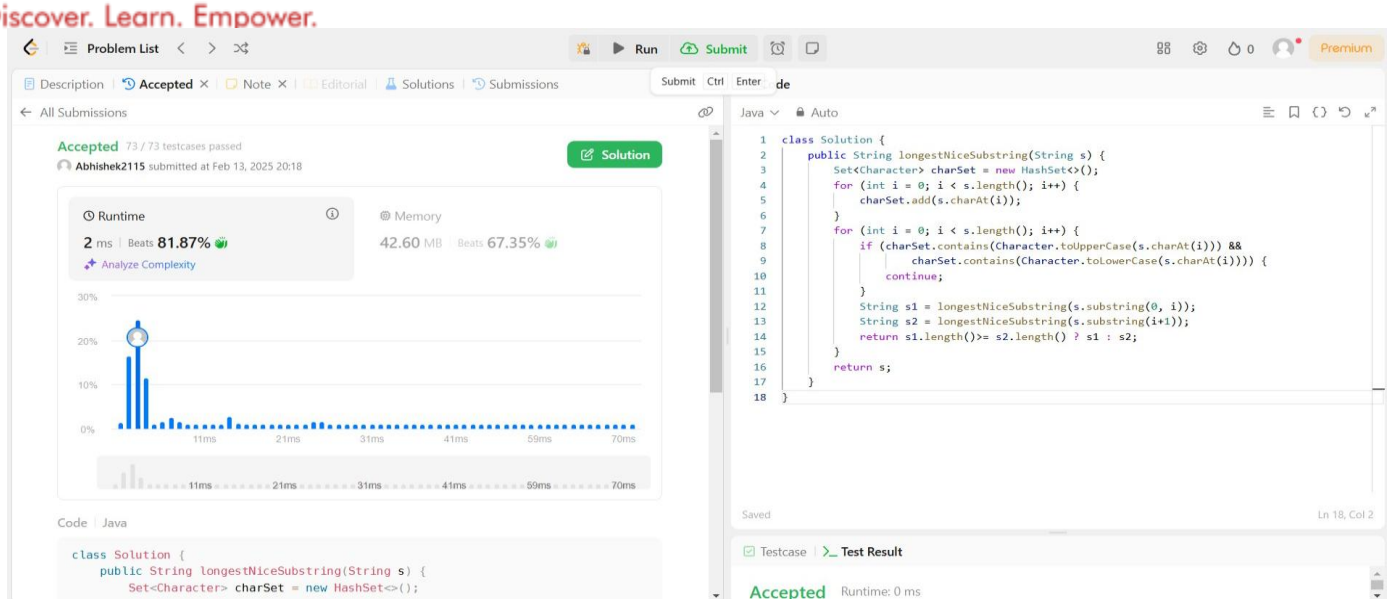
1. **Aim:** Longest Nice Substring

2. **Objective**:A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

3. **Code:**

```
class Solution {
    public String longestNiceSubstring(String s) {
        Set<Character> charSet = new HashSet<>();
        for (int i = 0; i < s.length(); i++) {
            charSet.add(s.charAt(i));
        }
        for (int i = 0; i < s.length(); i++) {
            if (charSet.contains(Character.toUpperCase(s.charAt(i))) &&
                    charSet.contains(Character.toLowerCase(s.charAt(i)))) {
                continue;
            }
            String s1 = longestNiceSubstring(s.substring(0, i));
            String s2 = longestNiceSubstring(s.substring(i+1));
            return s1.length()>= s2.length() ? s1 : s2;
        }
        return s;
    }
}
```

4. **Output:**



5. **Time complexity: O(n log n)** **Space Complexity: O(n)**

PROBLEM 2:

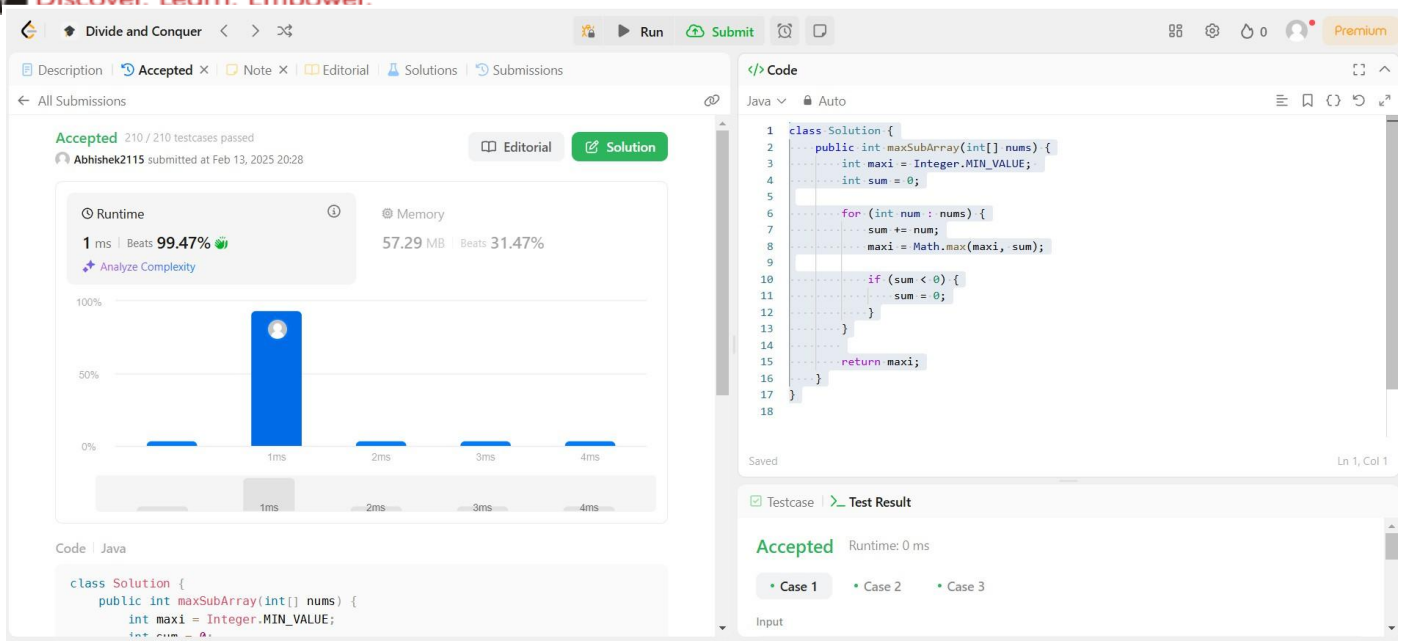
1. **Aim:Maximum Subarray**

2. **Objective: Given an integer array nums, find the subarray find the largest sum and return its sum**

3. **Code:**

```
class Solution {
   public int maxSubArray(int[] nums) {
      int maxi = Integer.MIN_VALUE;
      int sum = 0;

      for (int num : nums) {
         sum += num;
         maxi = Math.max(maxi, sum);

         if (sum < 0) {
            sum = 0;
         }
      }

      return maxi;
   }
}
```

4. **Output:**



5. **Time Complexity: O (n)** **Space Complexity:O(1)**

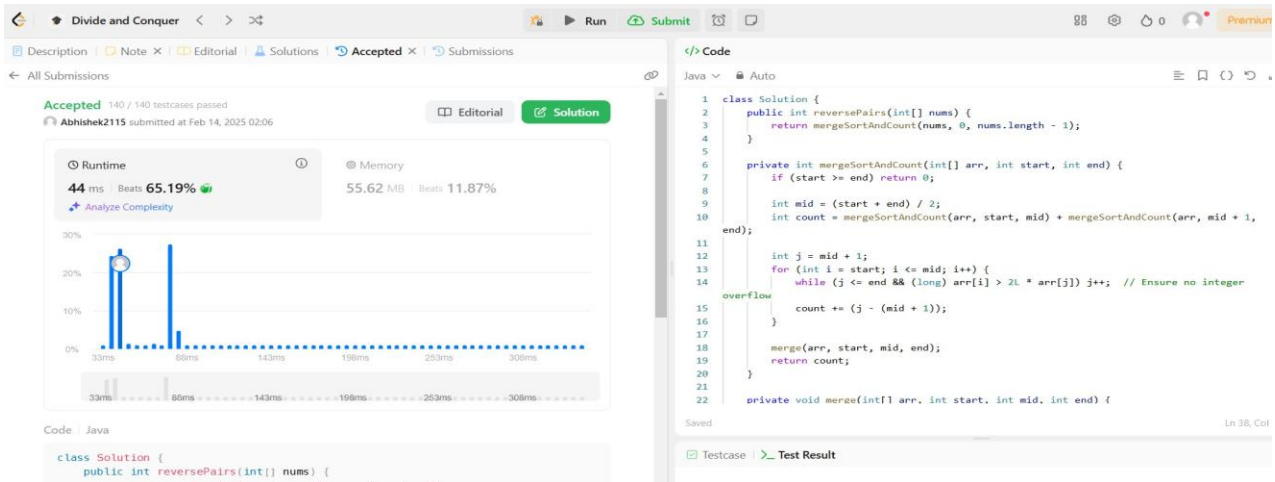
PROBLEM 3:

1. **Aim:** Reverse Pairs
2. **Objective: Given an integer array nums, return the number of reverse pairs in the array.**
3. **Code:**

```
class Solution {
    public int reversePairs(int[] nums) {
        return mergeSortAndCount(nums, 0, nums.length - 1);
    }

    private int mergeSortAndCount(int[] arr, int start, int end) {
        if (start >= end) return 0;

        int mid = (start + end) / 2;
        int count = mergeSortAndCount(arr, start, mid) + mergeSortAndCount(arr, mid + 1, end);

        int j = mid + 1;
        for (int i = start; i <= mid; i++) {
```

```
            while (j <= end && (long) arr[i] > 2L * arr[j]) j++;  // Ensure no integer overflow
            count += (j - (mid + 1));
        }

        merge(arr, start, mid, end);
        return count;
    }

    private void merge(int[] arr, int start, int mid, int end) {
        int n1 = mid - start + 1, n2 = end - mid;
        int[] left = new int[n1], right = new int[n2];

        for (int i = 0; i < n1; i++) left[i] = arr[start + i];
        for (int j = 0; j < n2; j++) right[j] = arr[mid + 1 + j];

        int i = 0, j = 0, k = start;
        while (i < n1 && j < n2) {
            if (left[i] <= right[j]) arr[k++] = left[i++];
            else arr[k++] = right[j++];
        }
        while (i < n1) arr[k++] = left[i++];
        while (j < n2) arr[k++] = right[j++];
    }
}
```

4. **Output:**



5. **Time complexity: O(N LOG N) Space complexity: O(n)**

PROBLEM 4:

1. **Aim:** Longest increasing subsequence II.

2. **Objective:** You are given an integer array nums and an integer k.

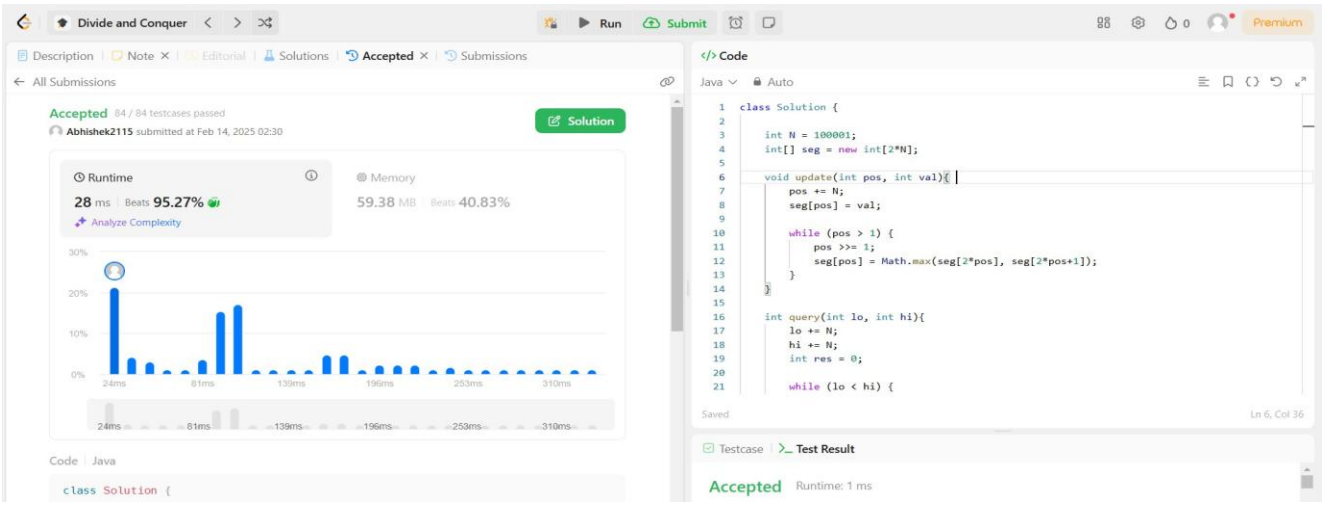
   Find the longest subsequence of nums that meets the following requirements:
   The subsequence is strictly increasing and
   The difference between adjacent elements in the subsequence is at most k.
   Return the length of the longest subsequence that meets the requirements.

3. **Code:**

```
class Solution {

    int N = 100001;
    int[] seg = new int[2*N];

    void update(int pos, int val){
        pos += N;
        seg[pos] = val;

        while (pos > 1) {
            pos >>= 1;
            seg[pos] = Math.max(seg[2*pos], seg[2*pos+1]);
        }
    }

    int query(int lo, int hi){
        lo += N;
        hi += N;
        int res = 0;

        while (lo < hi) {
            if ((lo & 1)==1) {
                res = Math.max(res, seg[lo++]);
            }
```

```
            if ((hi & 1)==1) {
                res = Math.max(res, seg[--hi]);
            }
            lo >>= 1;
            hi >>= 1;
        }
        return res;
    }

    public int lengthOfLIS(int[] A, int k) {
        int ans = 0;
        for (int i = 0; i < A.length; ++i){
            int l = Math.max(0, A[i]-k);
            int r = A[i];
            int res = query(l, r) + 1;
            ans = Math.max(res, ans);
            update(A[i], res);
        }
        return ans;
    }
}
```

4. Output:



5. **Time Complexity: O(N Log N)** **Space Complexity: O(n)**

6. **Learning Outcome:**

1. Recursion and Divide & Conquer: Used recursive approaches to solve problems like Longest Nice Substring and Reverse Pairs, breaking them into smaller subproblems.

2. Kadane's Algorithm:Implemented an O(N) solution for Maximum Subarray Sum, maintaining a running sum while discarding negative contributions.

3. Merge Sort for Counting Inversions: Used a modified merge sort (O(N log N)) to efficiently count reverse pairs while merging sorted subarrays.

4. Segment Tree for Range Queries: Implemented Segment Tree to efficiently compute Longest Increasing Subsequence with constraints in O(N log N).

5. Efficient Data Structures: Used HashSet for quick lookups in Longest Nice Substring, improving performance for character presence checks.

6. Handling Edge Cases and Overflow: Prevented integer overflow in Reverse Pairs using (long) arr[i] > 2L *arr[j], ensuring correct calculations.