Assignment - 4

Name: Aanchal Negi | **UID: 22BCS14969**

Branch: BE-CSE | **Section/Group: IOT-609/B**

Semester: 6th | **Date :18/03/25**

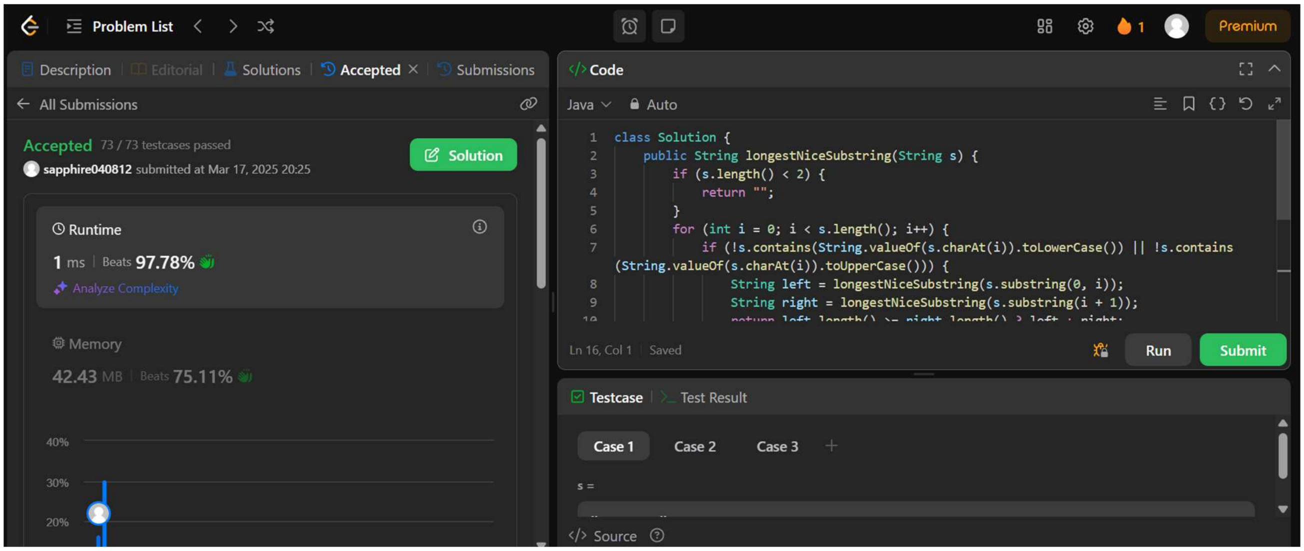
Subject Name: AP Lab-2 | **Subject Code: 22CSP-351**

➢ Longest Nice Substring

Code:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) {
            return "";
        }
        for (int i = 0; i < s.length(); i++) {
            if (!s.contains(String.valueOf(s.charAt(i)).toLowerCase()) ||
!s.contains(String.valueOf(s.charAt(i)).toUpperCase())) {
                String left = longestNiceSubstring(s.substring(0, i));
                String right = longestNiceSubstring(s.substring(i + 1));
                return left.length() >= right.length() ? left : right;
            }
        }
        return s;
    }
}
```

Output:



Accepted 73 / 73 testcases passed

sapphire040812 submitted at Mar 17, 2025 20:25

Runtime

1 ms | Beats 97.78%

Analyze Complexity

Memory

42.43 MB | Beats 75.11%

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) {
            return "";
        }
        for (int i = 0; i < s.length(); i++) {
            if (!s.contains(String.valueOf(s.charAt(i)).toLowerCase()) || !s.contains
(String.valueOf(s.charAt(i)).toUpperCase())) {
                String left = longestNiceSubstring(s.substring(0, i));
                String right = longestNiceSubstring(s.substring(i + 1));
```

Ln 16, Col 1 | Saved

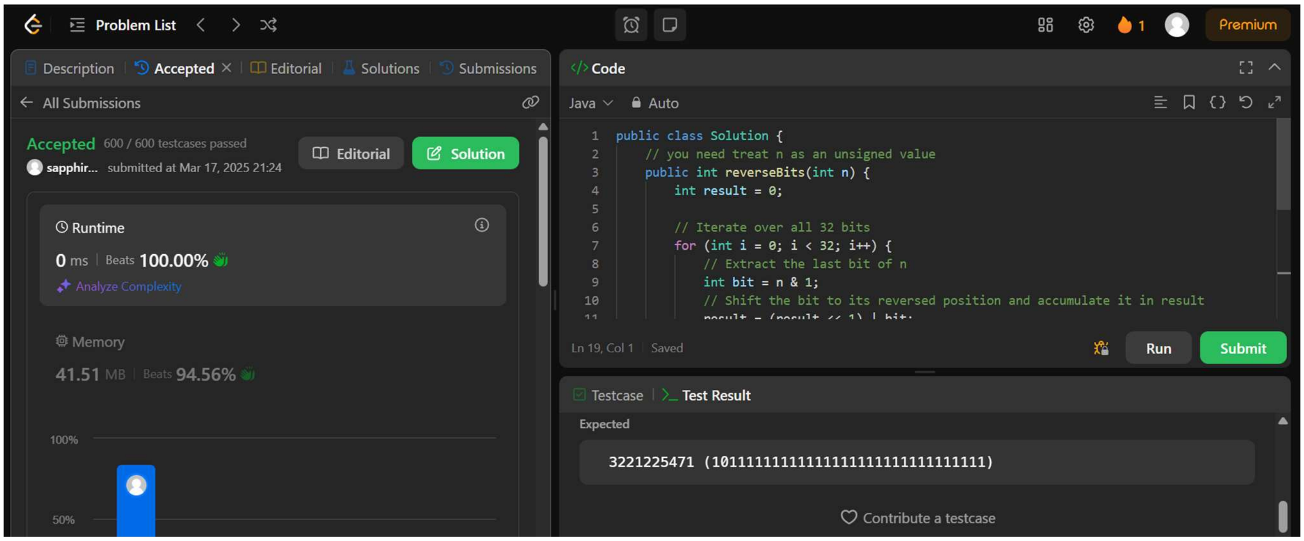
s =

➢ Reverse Bits

Code:

```
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {
        int result = 0;

        // Iterate over all 32 bits
        for (int i = 0; i < 32; i++) {
            // Extract the last bit of n
            int bit = n & 1;
            // Shift the bit to its reversed position and accumulate it in result
            result = (result << 1) | bit;
            // Right shift n to process the next bit
            n >>= 1;
        }

        return result;
    }
}
```

Output:



Accepted 600 / 600 testcases passed
sapphir... submitted at Mar 17, 2025 21:24

Runtime
0 ms | Beats 100.00%

Memory
41.51 MB | Beats 94.56%

```
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {
        int result = 0;

        // Iterate over all 32 bits
        for (int i = 0; i < 32; i++) {
            // Extract the last bit of n
            int bit = n & 1;
            // Shift the bit to its reversed position and accumulate it in result
            result = (result << 1) | bit;
```

Ln 19, Col 1 | Saved

Expected

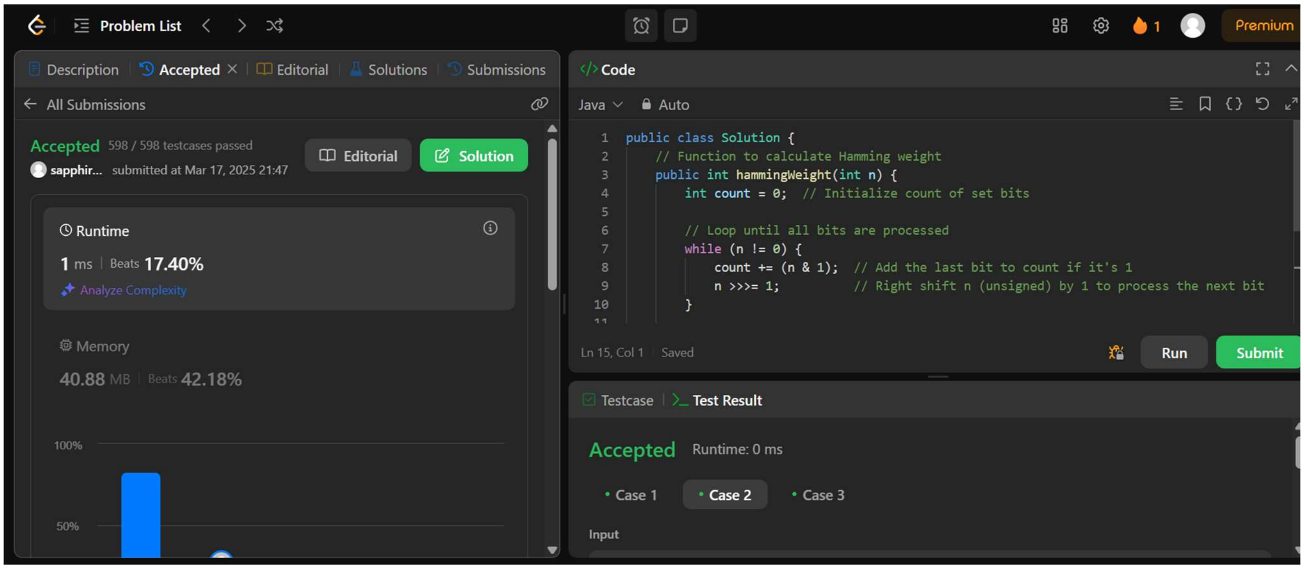
```
3221225471 (10111111111111111111111111111111)
```

➢ Number of 1 Bits

Code:

```
public class Solution {
    // Function to calculate Hamming weight
    public int hammingWeight(int n) {
        int count = 0;  // Initialize count of set bits

        // Loop until all bits are processed
        while (n != 0) {
            count += (n & 1);  // Add the last bit to count if it's 1
            n >>>= 1;          // Right shift n (unsigned) by 1 to process the next bit
        }

        return count;
    }
}
```

Output:



Accepted 598 / 598 testcases passed

sapphir... submitted at Mar 17, 2025 21:47

Runtime

1 ms | Beats 17.40%

Analyze Complexity

Memory

40.88 MB | Beats 42.18%

```
public class Solution {
    // Function to calculate Hamming weight
    public int hammingWeight(int n) {
        int count = 0;  // Initialize count of set bits

        // Loop until all bits are processed
        while (n != 0) {
            count += (n & 1);  // Add the last bit to count if it's 1
            n >>>= 1;          // Right shift n (unsigned) by 1 to process the next bit
        }
```

Ln 15, Col 1 | Saved

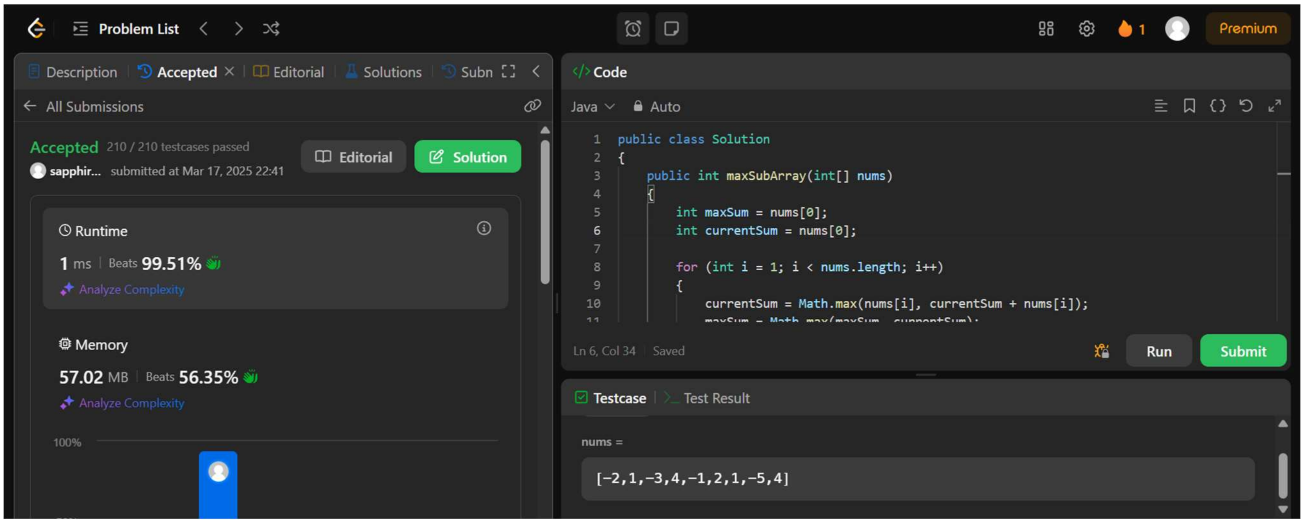
Accepted Runtime: 0 ms

➢ Maximum Subarray

Code:

```
public class Solution
{
   public int maxSubArray(int[] nums)
   {
      int maxSum = nums[0];
      int currentSum = nums[0];

      for (int i = 1; i < nums.length; i++)
      {
         currentSum = Math.max(nums[i], currentSum + nums[i]);
         maxSum = Math.max(maxSum, currentSum);
      }
      return maxSum;
   }
}
```

Output:



Accepted 210 / 210 testcases passed

sapphir... submitted at Mar 17, 2025 22:41

Runtime

1 ms | Beats 99.51%

Memory

57.02 MB | Beats 56.35%

```
public class Solution
{
    public int maxSubArray(int[] nums)
    {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++)
        {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
```

Ln 6, Col 34 | Saved

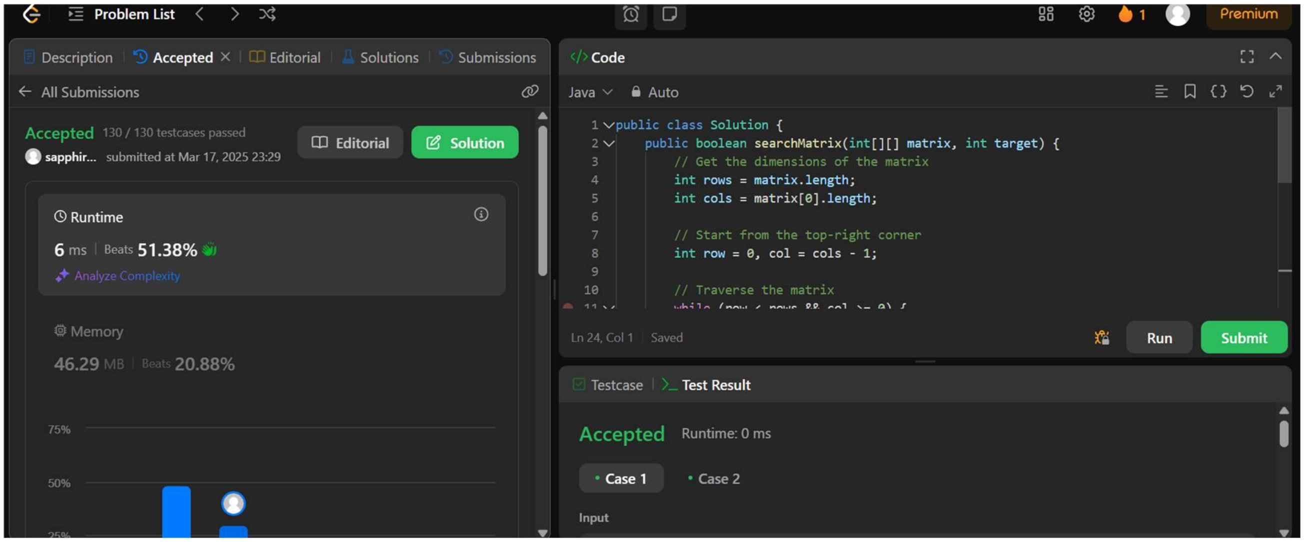
nums =

```
[-2,1,-3,4,-1,2,1,-5,4]
```

➢ Search a 2D Matrix II

Code:

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        // Get the dimensions of the matrix
        int rows = matrix.length;
        int cols = matrix[0].length;

        // Start from the top-right corner
        int row = 0, col = cols - 1;

        // Traverse the matrix
        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true; // Target found
            } else if (matrix[row][col] > target) {
                col--; // Move left
            } else {
                row++; // Move down
            }
        }

        return false; // Target not found
    }
}
```



Accepted 130 / 130 testcases passed

sapphir... submitted at Mar 17, 2025 23:29

Runtime

6 ms | Beats 51.38%

Analyze Complexity

Memory

46.29 MB | Beats 20.88%

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        // Get the dimensions of the matrix
        int rows = matrix.length;
        int cols = matrix[0].length;

        // Start from the top-right corner
        int row = 0, col = cols - 1;

        // Traverse the matrix
        while (row < rows && col >= 0) {
```

Ln 24, Col 1 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2

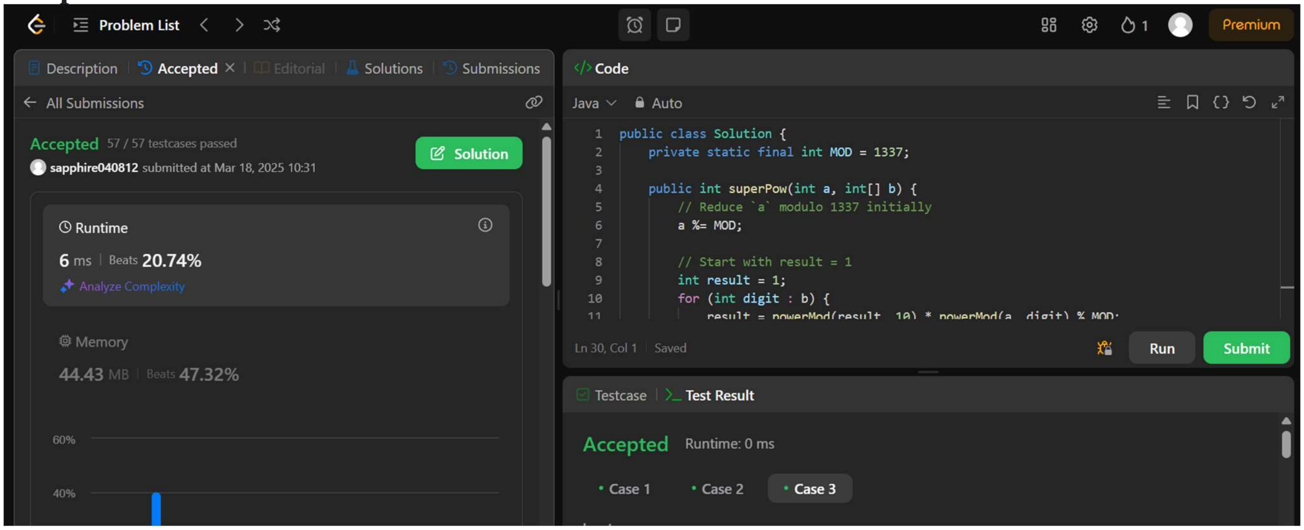
Input

➢ Super Pow

Code:

```
public class Solution {
    private static final int MOD = 1337;

    public int superPow(int a, int[] b) {
        // Reduce `a` modulo 1337 initially
        a %= MOD;

        // Start with result = 1
        int result = 1;
        for (int digit : b) {
            result = powerMod(result, 10) * powerMod(a, digit) % MOD;
        }

        return result;
    }

    // Helper method to calculate (base^exp) % MOD
    private int powerMod(int base, int exp) {
        int result = 1;
        while (exp > 0) {
            if (exp % 2 == 1) {
                result = result * base % MOD;
            }
            base = base * base % MOD;
            exp /= 2;
        }
        return result;
    }
}
```

Output:



Accepted 57 / 57 testcases passed

sapphire040812 submitted at Mar 18, 2025 10:31

Runtime

6 ms | Beats 20.74%

Analyze Complexity

Memory

44.43 MB | Beats 47.32%

```
public class Solution {
    private static final int MOD = 1337;

    public int superPow(int a, int[] b) {
        // Reduce `a` modulo 1337 initially
        a %= MOD;

        // Start with result = 1
        int result = 1;
        for (int digit : b) {
            result = powerMod(result, 10) * powerMod(a, digit) % MOD;
```

Ln 30, Col 1 | Saved

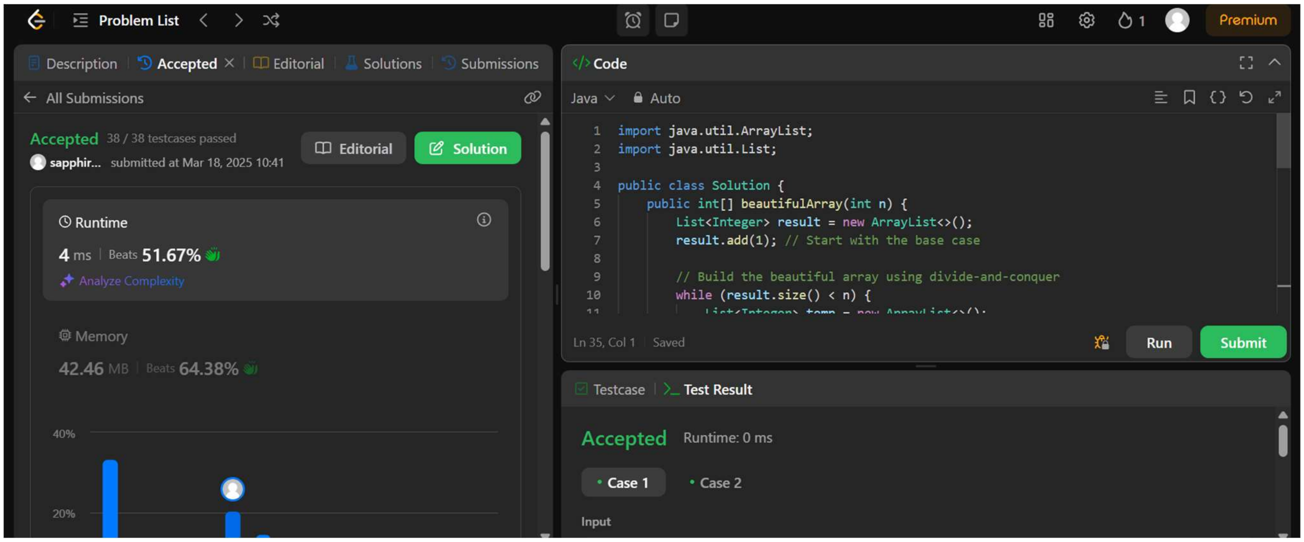
Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

➢ Beautiful Array

Code:

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

public class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = new ArrayList<>();
        result.add(1); // Start with the base case

        // Build the beautiful array using divide-and-conquer
        while (result.size() < n) {
            List<Integer> temp = new ArrayList<>();
            // Add odd numbers to preserve the beautiful property
            for (int num : result) {
                if (num * 2 - 1 <= n) {
                    temp.add(num * 2 - 1);
                }
            }
            // Add even numbers to preserve the beautiful property
            for (int num : result) {
                if (num * 2 <= n) {
                    temp.add(num * 2);
                }
            }
            result = temp;
        }

        // Convert the result list to an array
        int[] beautifulArray = new int[n];
        for (int i = 0; i < n; i++) {
            beautifulArray[i] = result.get(i);
        }
        return beautifulArray;
    }
}
```



Accepted 38 / 38 testcases passed

sapphir... submitted at Mar 18, 2025 10:41

Runtime

4 ms | Beats 51.67%

Analyze Complexity

Memory

42.46 MB | Beats 64.38%

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

public class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = new ArrayList<>();
        result.add(1); // Start with the base case

        // Build the beautiful array using divide-and-conquer
        while (result.size() < n) {
            List<Integer> temp = new ArrayList<>();
```

Ln 35, Col 1 | Saved

Accepted Runtime: 0 ms

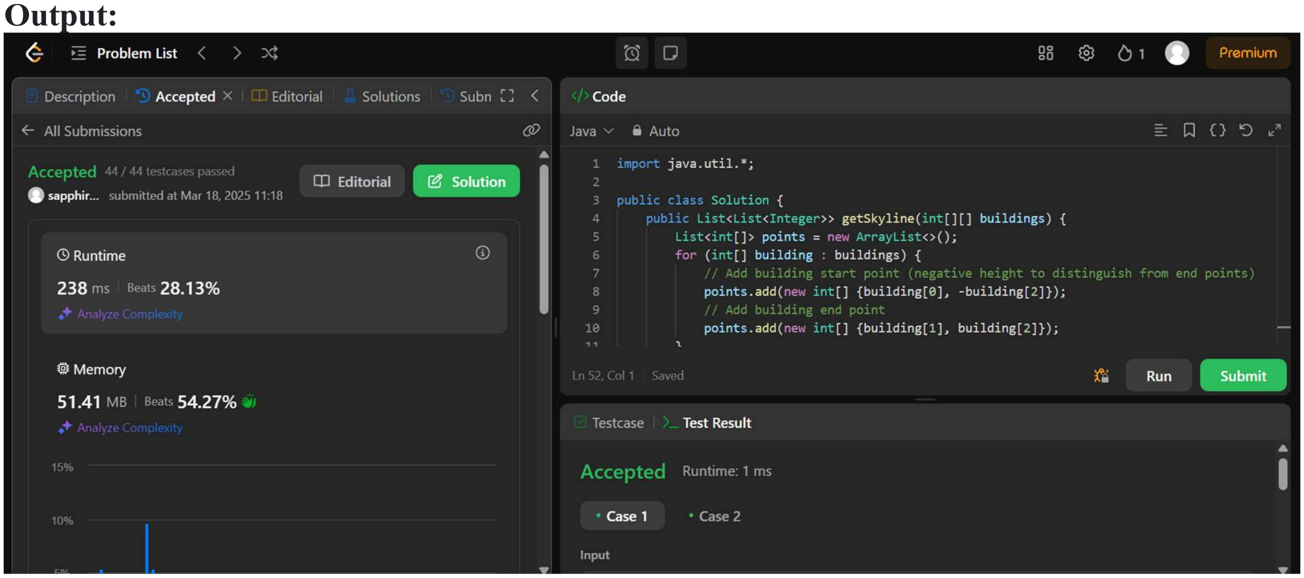
Input

➢ The Skyline Problem

Code:

```
import java.util.*;

public class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> points = new ArrayList<>();
        for (int[] building : buildings) {
            // Add building start point (negative height to distinguish from end points)
            points.add(new int[] {building[0], -building[2]});
            // Add building end point
            points.add(new int[] {building[1], building[2]});
        }

        // Sort points:
        // 1. By x-coordinate
        // 2. By height (start points before end points for same x)
        Collections.sort(points, (a, b) -> {
            if (a[0] != b[0]) return a[0] - b[0]; // Sort by x
            return a[1] - b[1]; // Start (-height) before end (+height)
        });

        // Use a max-heap to keep track of building heights
        PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
        maxHeap.add(0); // Initialize with ground level
        int prevMaxHeight = 0;

        List<List<Integer>> result = new ArrayList<>();
        for (int[] point : points) {
            int x = point[0];
            int height = point[1];

            if (height < 0) {
                // Start of a building: add its height
                maxHeap.add(-height);
            } else {
                // End of a building: remove its height
                maxHeap.remove(height);
            }

            // Get current max height
            int currMaxHeight = maxHeap.peek();

            // If max height changes, add a key point
            if (currMaxHeight != prevMaxHeight) {
                result.add(Arrays.asList(x, currMaxHeight));
                prevMaxHeight = currMaxHeight;
            }
        }
        return result;
    }
}
```

Output:



Accepted 44 / 44 testcases passed

sapphir... submitted at Mar 18, 2025 11:18

Runtime

238 ms | Beats 28.13%

Memory

51.41 MB | Beats 54.27%

```
import java.util.*;

public class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> points = new ArrayList<>();
        for (int[] building : buildings) {
            // Add building start point (negative height to distinguish from end points)
            points.add(new int[] {building[0], -building[2]});
            // Add building end point
            points.add(new int[] {building[1], building[2]});
        }
```

Ln 52, Col 1 | Saved

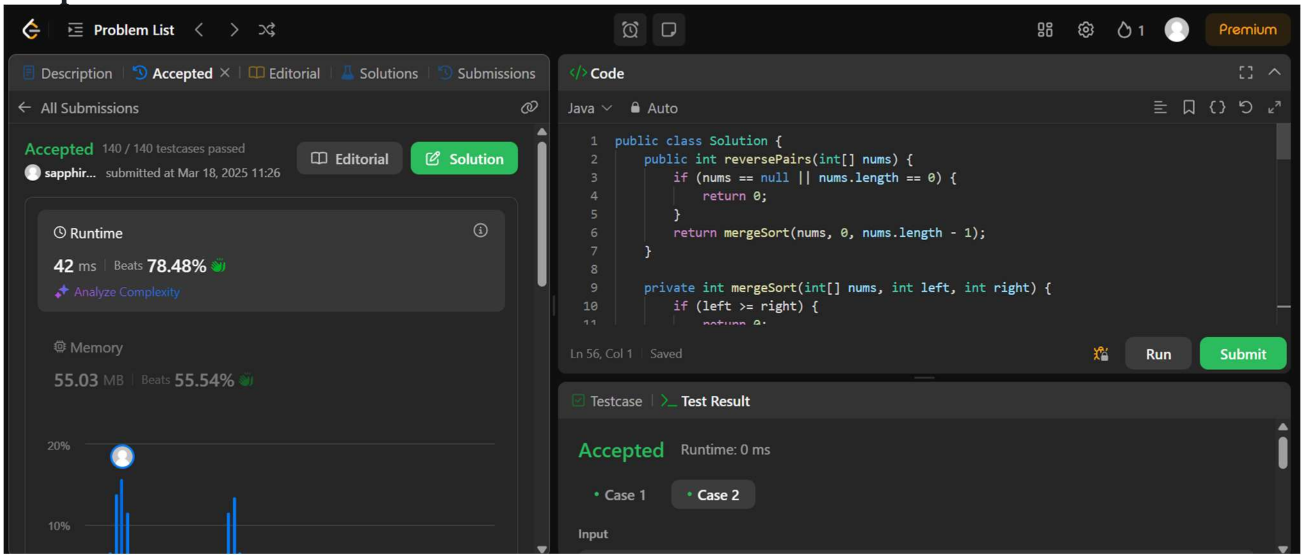
Accepted Runtime: 1 ms

➢ Reverse Pairs :

Code:

```
public class Solution {
    public int reversePairs(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        return mergeSort(nums, 0, nums.length - 1);
    }

    private int mergeSort(int[] nums, int left, int right) {
        if (left >= right) {
            return 0;
        }

        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

        // Count reverse pairs in the current range
        int j = mid + 1;
        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2L * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }
        // Merge the two halves
        merge(nums, left, mid, right);
        return count;
    }

    private void merge(int[] nums, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;

        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) {
                temp[k++] = nums[i++];
            } else {
                temp[k++] = nums[j++];
            }
        }

        while (i <= mid) {
            temp[k++] = nums[i++];
        }

        while (j <= right) {
            temp[k++] = nums[j++];
        }

        for (int p = 0; p < temp.length; p++) {
            nums[left + p] = temp[p];
        }
    }
}
```

Output:



Accepted 140 / 140 testcases passed
sapphir... submitted at Mar 18, 2025 11:26

Runtime
42 ms | Beats 78.48%
Analyze Complexity

Memory
55.03 MB | Beats 55.54%

```
public class Solution {
    public int reversePairs(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        return mergeSort(nums, 0, nums.length - 1);
    }

    private int mergeSort(int[] nums, int left, int right) {
        if (left >= right) {
            return 0;
```

Ln 56, Col 1 | Saved

Accepted Runtime: 0 ms

Case 1
Case 2

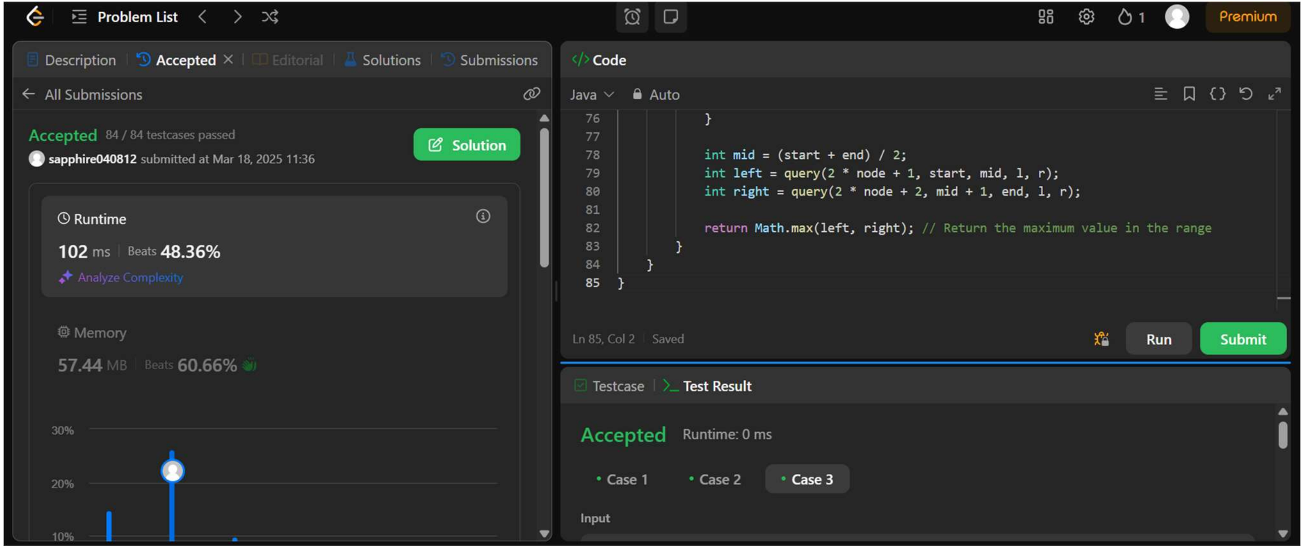
Input

➢ Longest Increasing Subsequence II

Code:

```
import java.util.*;

class Solution {
    public int lengthOfLIS(int[] nums, int k) {
        // Find the maximum number in nums to determine the size of the segment tree
        int maxNum = 0;
        for (int num : nums) {
            if (num > maxNum) {
                maxNum = num;
            }
        }

        // Initialize the segment tree
        SegmentTree st = new SegmentTree(maxNum);

        int result = 0;
        for (int num : nums) {
            // Query the maximum subsequence length in the range [num - k, num - 1]
            int prevMax = st.query(Math.max(1, num - k), num - 1);
            int currentLength = prevMax + 1;

            // Update the segment tree with the new subsequence length for the current number
            st.update(num, currentLength);

            // Update the result with the maximum subsequence length found so far
            result = Math.max(result, currentLength);
        }

        return result;
    }

    // Segment Tree implementation
    class SegmentTree {
        int[] tree;
        int size;

        public SegmentTree(int size) {
            this.size = size;
            this.tree = new int[4 * (size + 1)]; // Ensure enough space for the tree
        }

        // Update the value at a specific index
        public void update(int index, int value) {
            update(0, 0, size, index, value);
        }

        private void update(int node, int start, int end, int index, int value) {
            if (start == end) {
                tree[node] = Math.max(tree[node], value); // Ensure we take the maximum value
                return;
            }

            int mid = (start + end) / 2;
            if (index <= mid) {
                update(2 * node + 1, start, mid, index, value);
            } else {
                update(2 * node + 2, mid + 1, end, index, value);
            }

            // Update the current node with the maximum value from its children
            tree[node] = Math.max(tree[2 * node + 1], tree[2 * node + 2]);
        }

        // Query the maximum value in a range [l, r]
        public int query(int l, int r) {
            return query(0, 0, size, l, r);
        }

        private int query(int node, int start, int end, int l, int r) {
            if (r < start || end < l) {
                return 0; // Out of range
            }

            if (l <= start && end <= r) {
                return tree[node]; // Fully within range
            }

            int mid = (start + end) / 2;
            int left = query(2 * node + 1, start, mid, l, r);
            int right = query(2 * node + 2, mid + 1, end, l, r);

            return Math.max(left, right); // Return the maximum value in the range
        }
    }
}
```

Output:



Accepted 84 / 84 testcases passed

sapphire040812 submitted at Mar 18, 2025 11:36

Runtime

102 ms | Beats 48.36%

Analyze Complexity

Memory

57.44 MB | Beats 60.66%

```
            }

            int mid = (start + end) / 2;
            int left = query(2 * node + 1, start, mid, l, r);
            int right = query(2 * node + 2, mid + 1, end, l, r);

            return Math.max(left, right); // Return the maximum value in the range
        }
    }
}
```

Ln 85, Col 2 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input