

Student Name: Chirag

Branch: BE-CSE

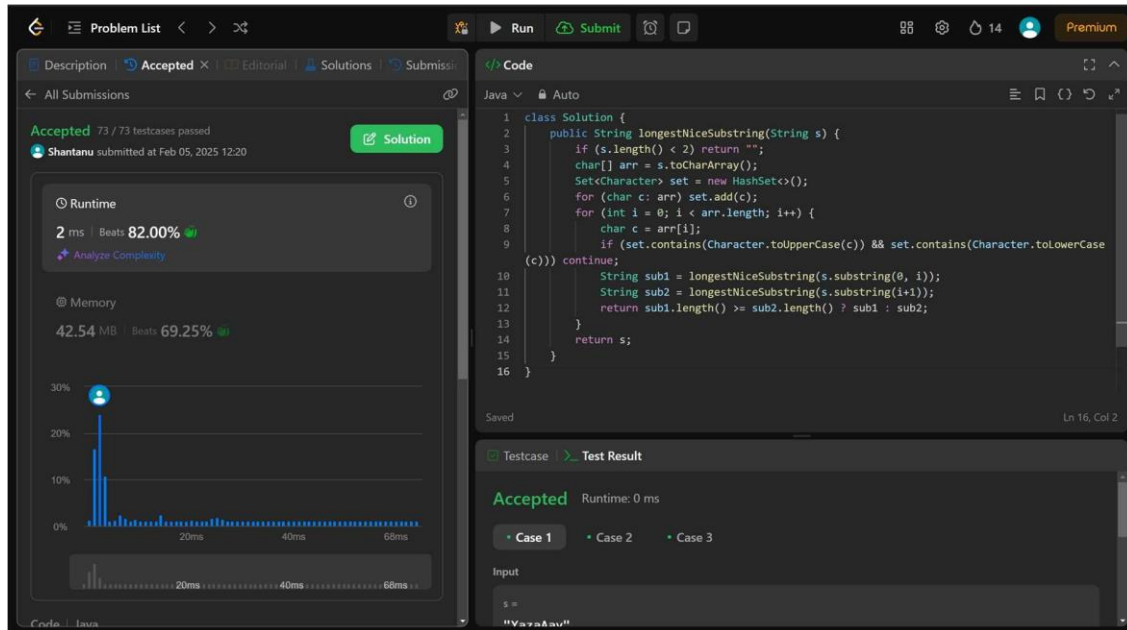
UID: 22BCS15116

Section: FL_603/A

Q1. 1763. Longest Nice Substring

Solution:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";
        char[] arr = s.toCharArray();
        Set<Character> set = new HashSet<>();
        for (char c: arr) set.add(c);
        for (int i = 0; i < arr.length; i++) {
            char c = arr[i];
            if (set.contains(Character.toUpperCase(c)) && set.contains(Character.toLowerCase(c))) continue;
            String sub1 = longestNiceSubstring(s.substring(0, i));
            String sub2 = longestNiceSubstring(s.substring(i+1));
            return sub1.length() >= sub2.length() ? sub1 : sub2;
        }
        return s;
    }
}
```



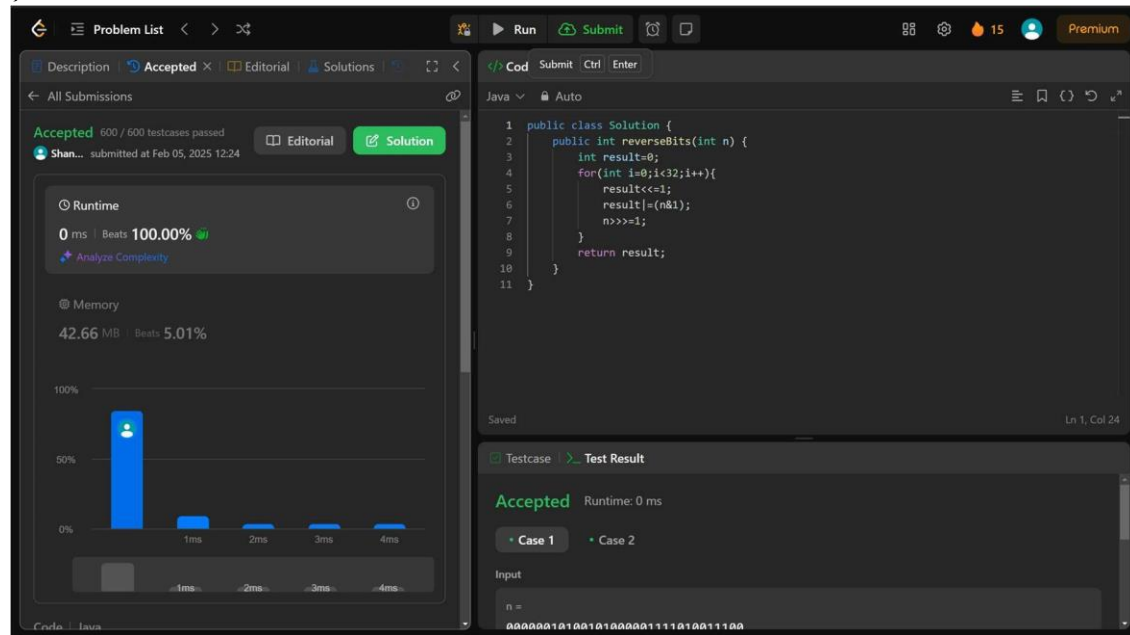
Q2. 190. Reverse Bits

```
public class Solution {
    public int reverseBits(int n) {
        int result=0;
```

```

for(int i=0;i<32;i++){
    result<<=1;
    result|=(n&1);
    n>>=1;
}
return result;
}
}

```

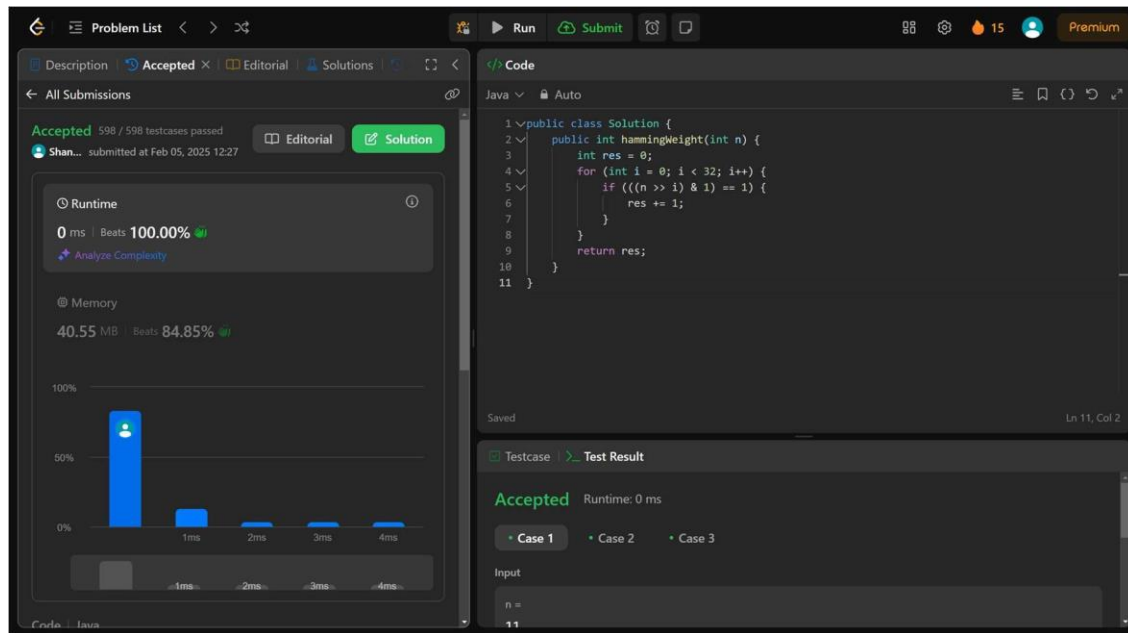


Q3. 191. Number of 1 Bits

```

public class Solution {
    public int hammingWeight(int n) {
        int res = 0;
        for (int i = 0; i < 32; i++) {
            if (((n >> i) & 1) == 1) {
                res += 1;
            }
        }
        return res;
    }
}

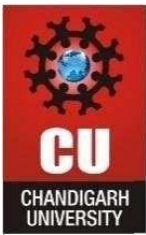
```



Q4. 53. Maximum Subarray

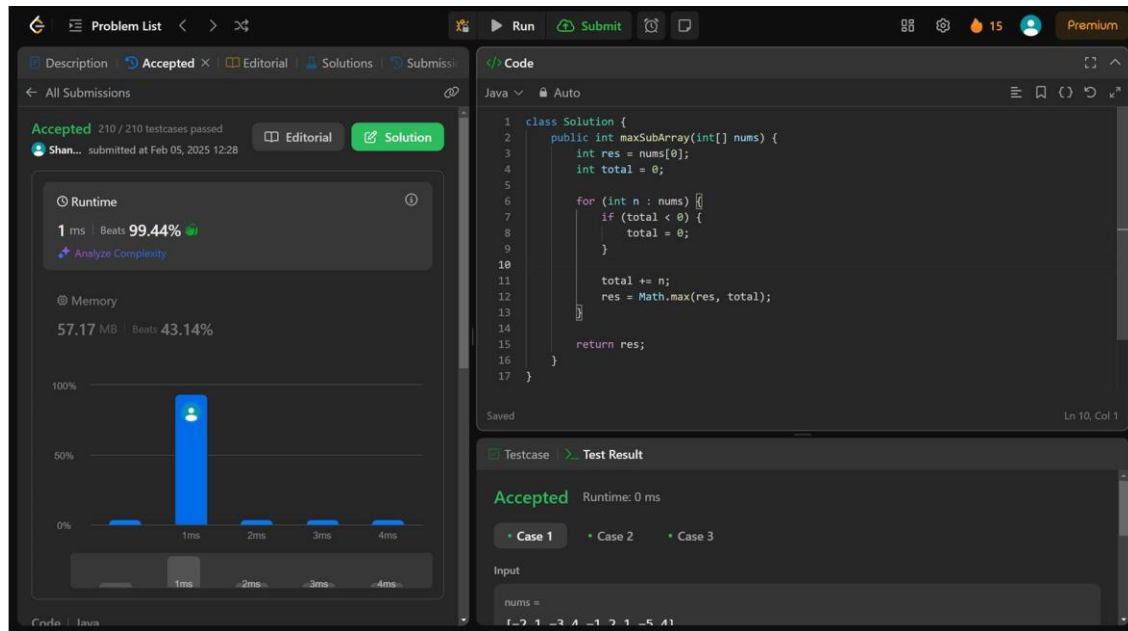
```

class Solution {
    public int maxSubArray(int[] nums) {
        int res = nums[0];
        int total = 0;
        for (int n : nums) {
            if (total < 0) {
                total = 0;
            }
            total += n;
            res = Math.max(res, total);
        }
        return res;
    }
}
```



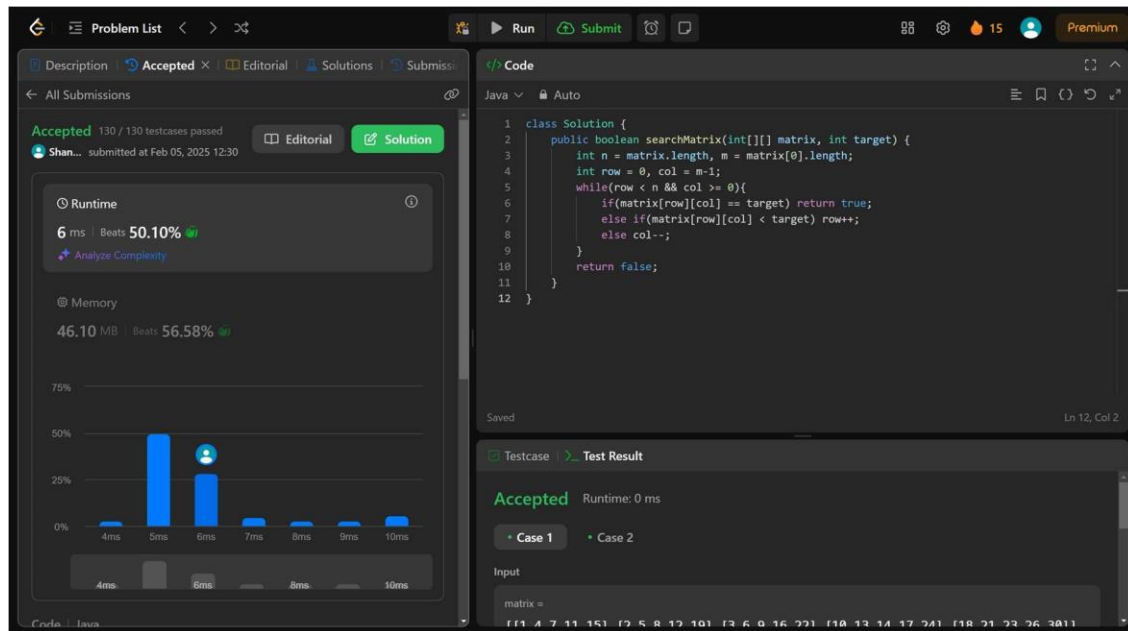
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.



Q5. 240. Search a 2D Matrix II

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int n = matrix.length, m = matrix[0].length;
        int row = 0, col = m-1;
        while(row < n && col >= 0){
            if(matrix[row][col] == target) return true;
            else if(matrix[row][col] < target) row++;
            else col--;
        }
        return false;
    }
}
```



Q6. 372. Super Pow

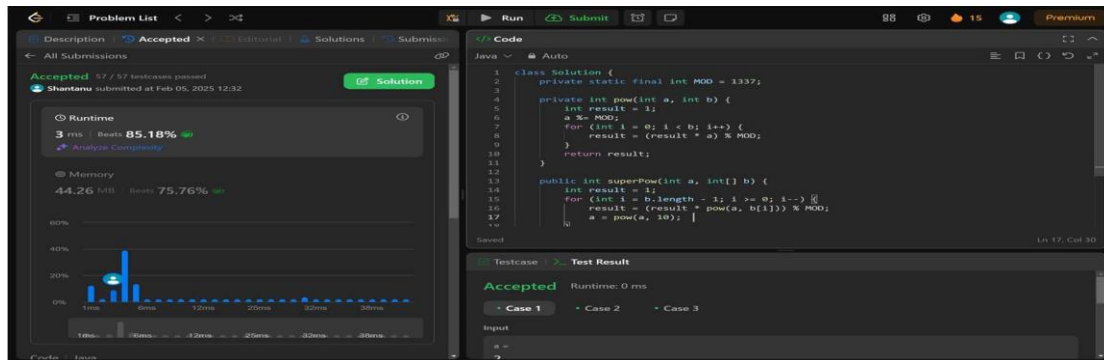
```

class Solution {
    private static final int MOD = 1337;

    private int pow(int a, int b) {
        int result = 1;
        a %= MOD; // Taking mod to prevent overflow
        for (int i = 0; i < b; i++) {
            result = (result * a) % MOD;
        }
        return result;
    }

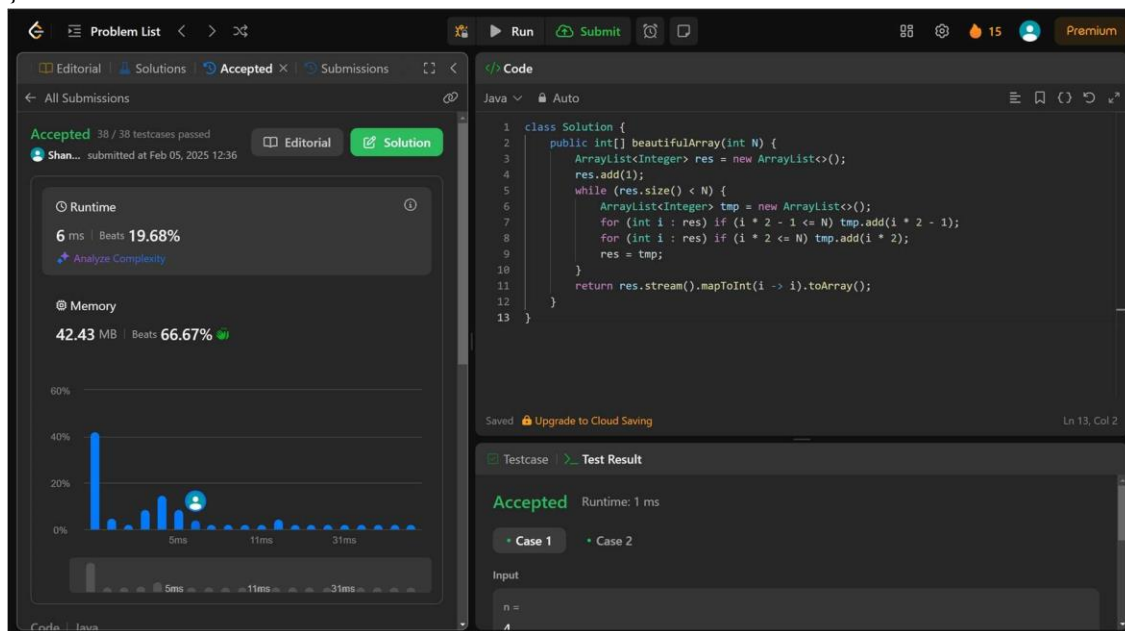
    public int superPow(int a, int[] b) {
        int result = 1;
        for (int i = b.length - 1; i >= 0; i--) {
            result = (result * pow(a, b[i])) % MOD;
            a = pow(a, 10); // Power up for the next iteration
        }
        return result;
    }
}

```

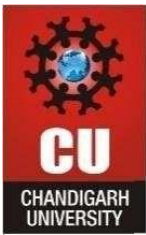


Q7. 932. Beautiful Array

```
class Solution {
    public int[] beautifulArray(int N) {
        ArrayList<Integer> res = new ArrayList<>();
        res.add(1);
        while (res.size() < N) {
            ArrayList<Integer> tmp = new ArrayList<>();
            for (int i : res) if (i * 2 - 1 <= N) tmp.add(i * 2 - 1);
            for (int i : res) if (i * 2 <= N) tmp.add(i * 2);
            res = tmp;
        }
        return res.stream().mapToInt(i -> i).toArray();
    }
}
```



Q8. 493. Reverse Pairs



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

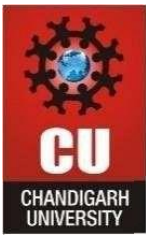
```
class Solution {
    public void merge(int[] arr, int low, int mid, int high) {
        ArrayList<Integer> temp = new ArrayList<>();
        int left = low;
        int right = mid+1;

        while(left <= mid && right <= high) {
            if(arr[left] <= arr[right]) {
                temp.add(arr[left++]);
            } else {
                temp.add(arr[right++]);
            }
        }
        while(left <= mid) temp.add(arr[left++]);
        while(right <= high) temp.add(arr[right++]);
        for(int i=low; i<=high; i++) {
            arr[i] = temp.get(i-low);
        }
    }

    public int countPairs(int[] arr, int low, int mid, int high) {
        int right = mid + 1;
        int cnt = 0;
        for(int i=low; i<=mid; i++) {
            while(right <= high && (long) arr[i] > 2L * arr[right])
                right++;
            cnt += (right - (mid + 1));
        }
        return cnt;
    }

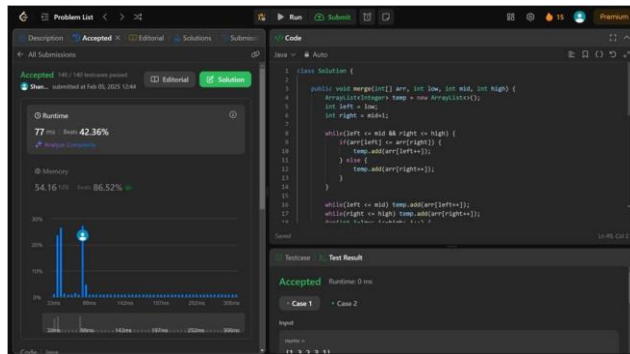
    public int mergeSort(int[] arr, int low, int high) {
        int cnt = 0;
        if(low >= high) return cnt;
        int mid = (low + high) / 2;
        cnt += mergeSort(arr, low, mid);
        cnt += mergeSort(arr, mid+1, high);
        cnt += countPairs(arr, low, mid, high);
        merge(arr, low, mid, high);
        return cnt;
    }

    public int reversePairs(int[] nums) {
        int n = nums.length;
        return mergeSort(nums, 0, n-1);
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.



Q9. 2407. Longest Increasing Subsequence II

```
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;

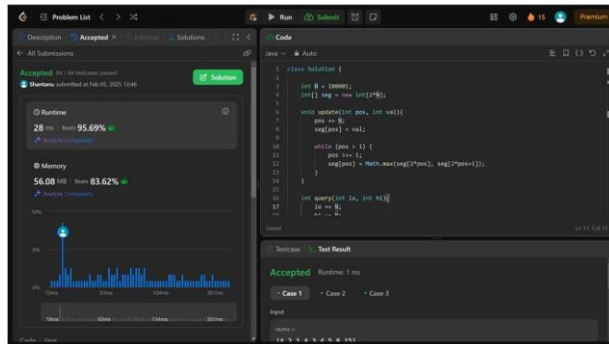
        while (lo < hi) {
            if ((lo & 1) == 1) {
                res = Math.max(res, seg[lo++]);
            }
            if ((hi & 1) == 1) {
                res = Math.max(res, seg[--hi]);
            }
            lo >>= 1;
            hi >>= 1;
        }
        return res;
    }
    public int 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;
}
}

```

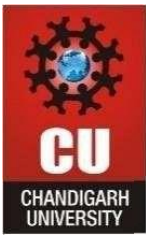


Q10. Merge Sorted Array

```

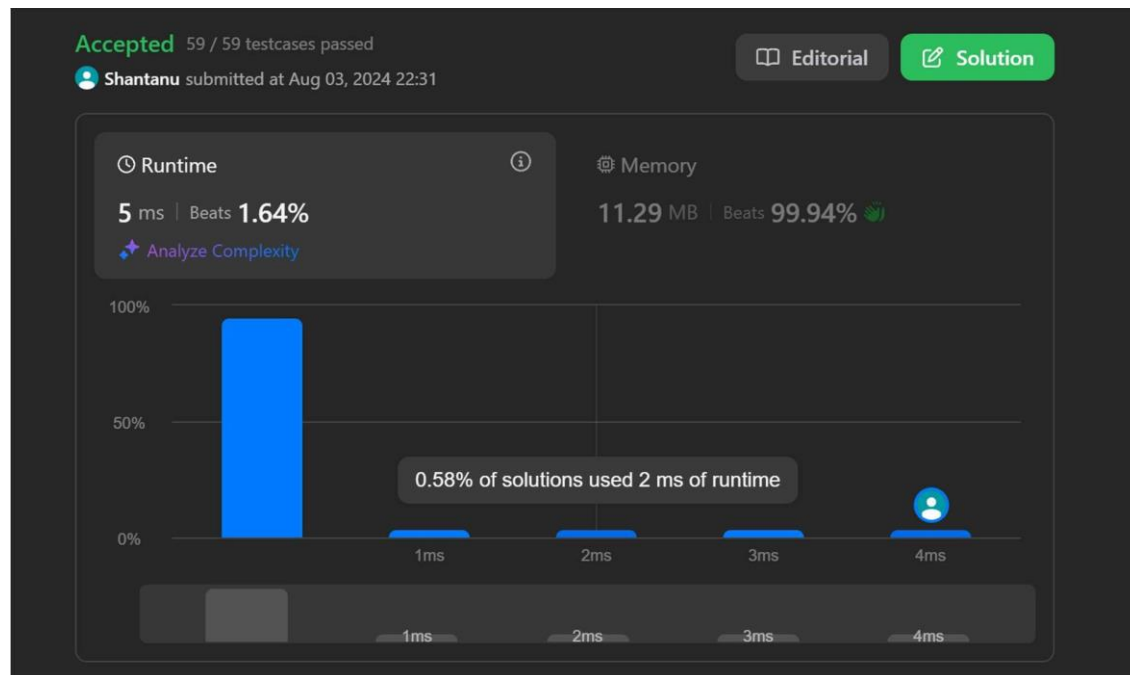
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m-1;
        int j = n-1;
        int k = (m+n-1);
        while (i>=0&& j>=0){
            if (nums1[i]>nums2[j]){
                nums1[k--]=nums1[i--];
            }
            else {
                nums1[k--]=nums2[j--];
            }
        }
        while (i>=0){
            nums1[k--]=nums1[i--];
        }
        while (j>=0){
            nums1[k--]=nums2[j--];
        }
    }
};

```



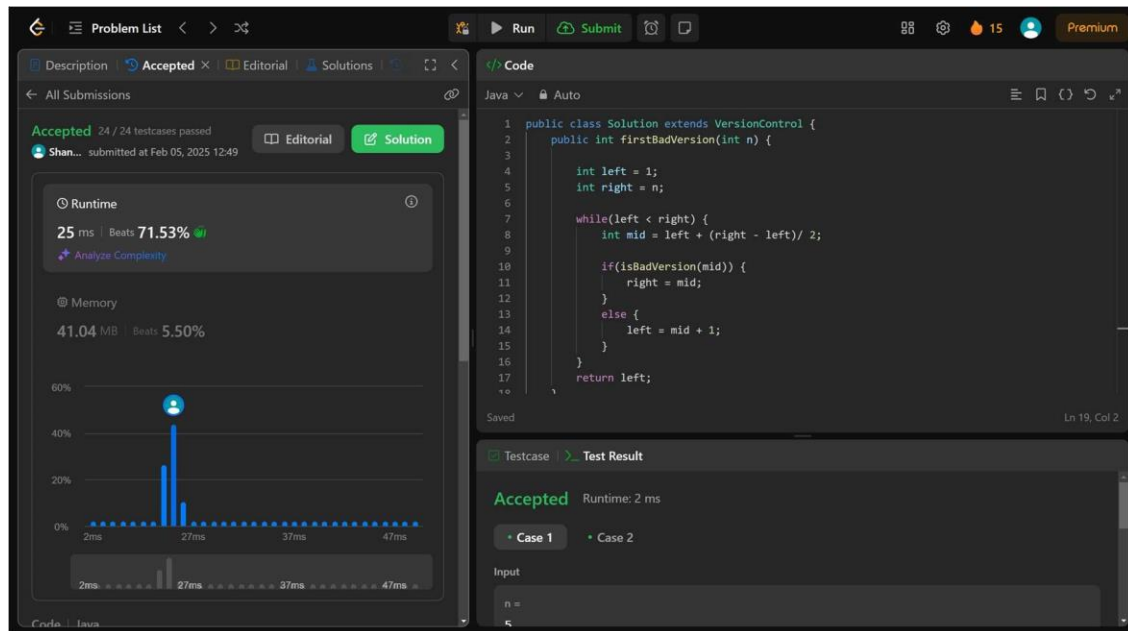
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.



Q11. 278. First Bad Version

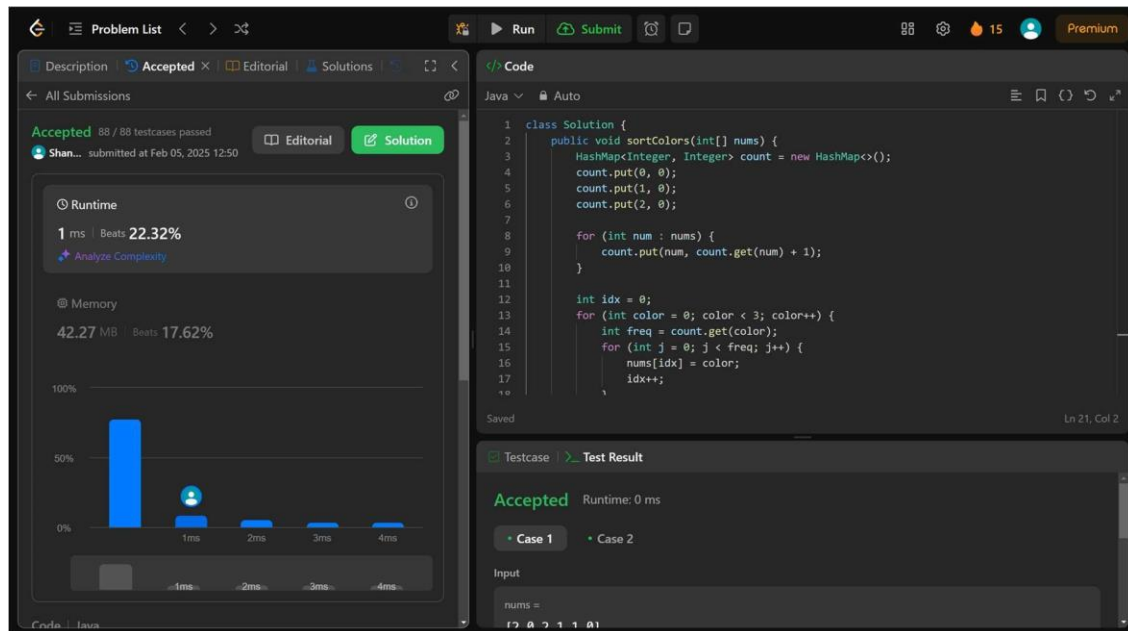
```
public class Solution extends VersionControl {  
    public int firstBadVersion(int n)  
    {  
        int left = 1;  
        int right = n;  
        while(left < right) {  
            int mid = left + (right - left) / 2;  
            if(isBadVersion(mid)) {  
                right = mid;  
            }  
            else {  
                left = mid + 1;  
            }  
        }  
        return left;  
    }  
}
```



Q12. 75. Sort Colors

```
class Solution {
    public void sortColors(int[] nums) {
        HashMap<Integer, Integer> count = new HashMap<>();
        count.put(0, 0);
        count.put(1, 0);
        count.put(2, 0);
        for (int num : nums) {
            count.put(num, count.get(num) + 1);
        }

        int idx = 0;
        for (int color = 0; color < 3; color++) {
            int freq = count.get(color);
            for (int j = 0; j < freq; j++) {
                nums[idx] = color;
                idx++;
            }
        }
    }
}
```



The screenshot shows a coding platform interface with a dark theme. On the left, the 'Runtime' tab is active, displaying performance metrics: 1 ms runtime and 22.32% beats, and 42.27 MB memory and 17.62% beats. A bar chart shows the distribution of runtime performance. The 'Code' tab on the right shows a Java solution for the 'Top K Frequent Elements' problem. The solution uses a HashMap to count frequencies and a PriorityQueue to find the top k elements. The 'Testcase' tab at the bottom shows the solution is 'Accepted' with a runtime of 0 ms for the provided input.

```

1 class Solution {
2     public int[] topKFrequent(int[] nums) {
3         HashMap<Integer, Integer> count = new HashMap<>();
4         count.put(0, 0);
5         count.put(1, 0);
6         count.put(2, 0);
7
8         for (int num : nums) {
9             count.put(num, count.get(num) + 1);
10        }
11
12        int idx = 0;
13        for (int color = 0; color < 3; color++) {
14            int freq = count.get(color);
15            for (int j = 0; j < freq; j++) {
16                nums[idx] = color;
17                idx++;
18            }
19        }
20    }
21}

```

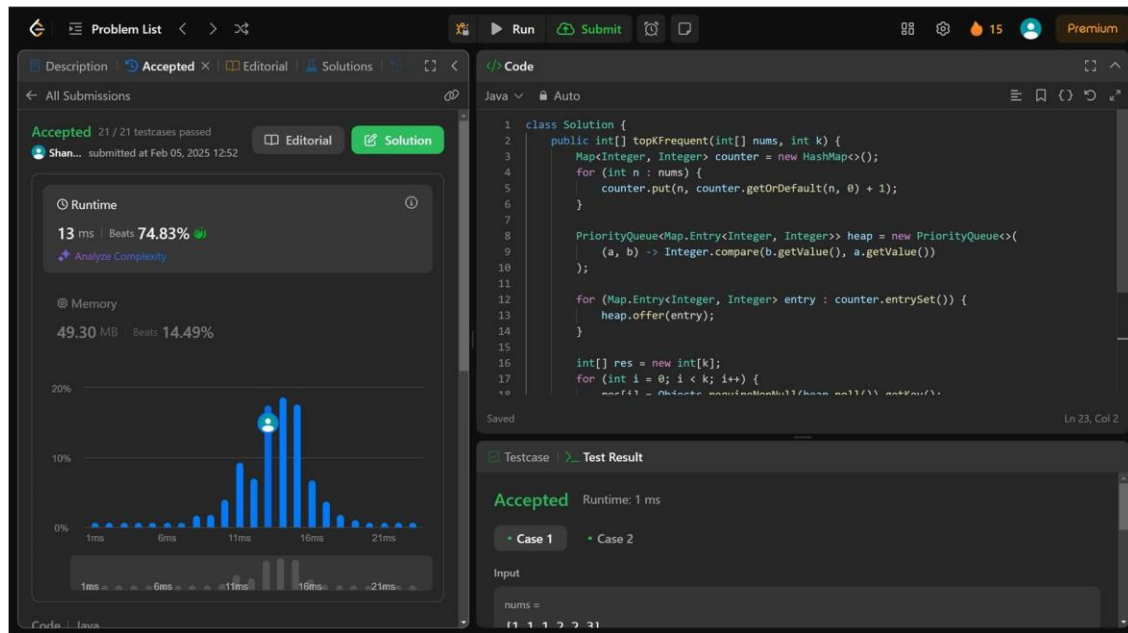
Q13. 347. Top K Frequent Elements

```

class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> counter = new HashMap<>();
        for (int n : nums) {
            counter.put(n, counter.getOrDefault(n, 0) + 1);
        }
        PriorityQueue<Map.Entry<Integer, Integer>> heap = new PriorityQueue<>(
            (a, b) -> Integer.compare(b.getValue(), a.getValue())
        );
        for (Map.Entry<Integer, Integer> entry : counter.entrySet()) {
            heap.offer(entry);
        }
        int[] res = new int[k];
        for (int i = 0; i < k; i++) {
            res[i] = Objects.requireNonNull(heap.poll()).getKey();
        }

        return res;
    }
}

```

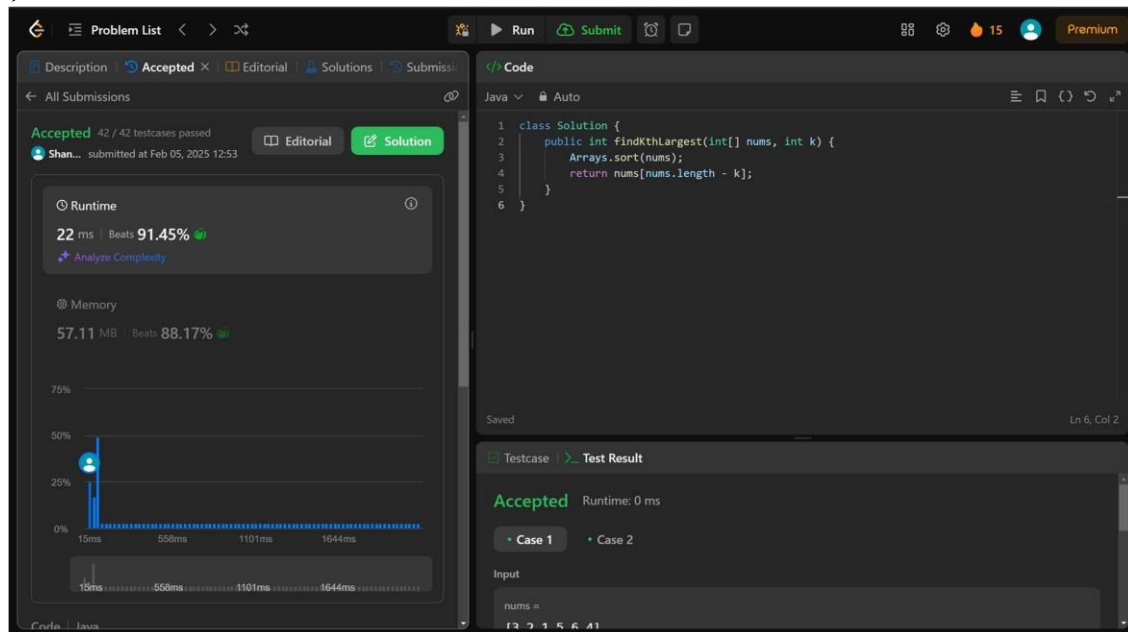


Q14. 215. Kth Largest Element in an Array

```

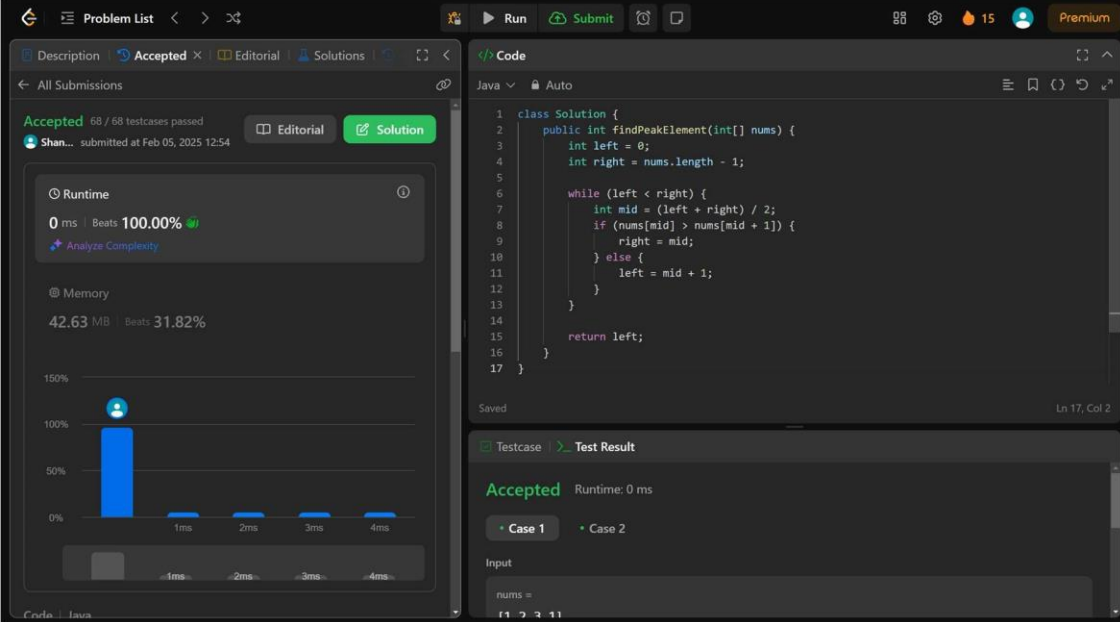
class Solution {
    public int findKthLargest(int[] nums, int k) {
        Arrays.sort(nums);
        return nums[nums.length - k];
    }
}

```



Q15. 162. Find Peak Element

```
class Solution {  
    public int findPeakElement(int[] nums) {  
        int left = 0;  
        int right = nums.length - 1;  
  
        while (left < right) {  
            int mid = (left + right) / 2;  
            if (nums[mid] > nums[mid + 1]) {  
                right = mid;  
            } else {  
                left = mid + 1;  
            }  
        }  
  
        return left;  
    }  
}
```



The screenshot displays a code editor interface for a problem titled "162. Find Peak Element". The code is written in Java and implements a binary search algorithm to find a peak element in an array. The code is as follows:

```
1 class Solution {  
2     public int findPeakElement(int[] nums) {  
3         int left = 0;  
4         int right = nums.length - 1;  
5  
6         while (left < right) {  
7             int mid = (left + right) / 2;  
8             if (nums[mid] > nums[mid + 1]) {  
9                 right = mid;  
10            } else {  
11                left = mid + 1;  
12            }  
13        }  
14  
15        return left;  
16    }  
17 }
```

The left sidebar shows the "Runtime" and "Memory" performance metrics. The "Runtime" section indicates "0 ms" and "Beats 100.00%", with a link to "Analyze Complexity". The "Memory" section shows "42.63 MB" and "Beats 31.82%". Below these, a bar chart compares the user's performance with other submissions, showing a blue bar at 100% for runtime and a grey bar at approximately 32% for memory.

The right sidebar shows the "Testcase" and "Test Result" sections. The "Test Result" section indicates "Accepted" with a "Runtime: 0 ms". Below this, there are two tabs for "Case 1" and "Case 2". The "Input" section shows the input array: `nums = [1, 2, 3, 2, 1]`.