AP Worksheet

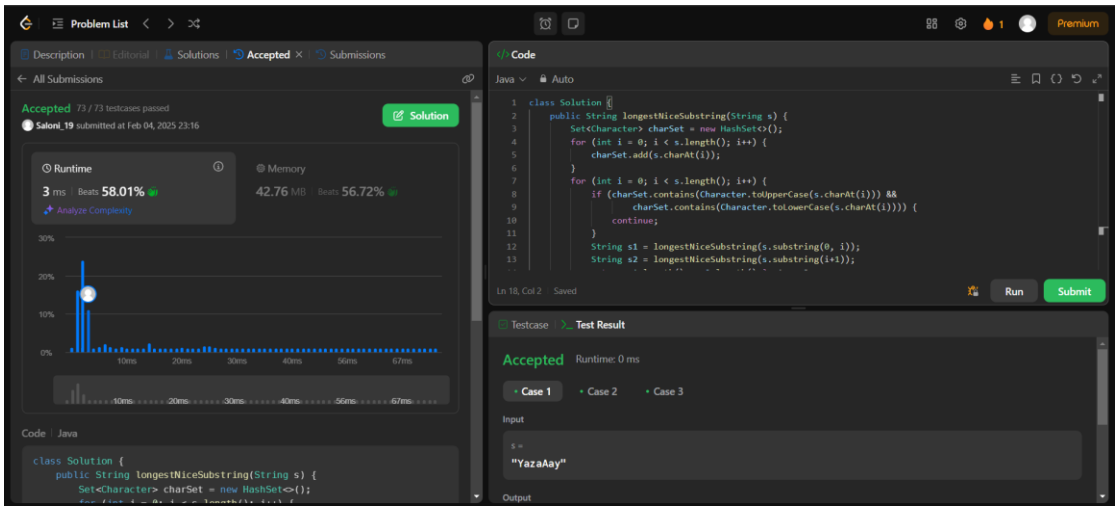
Saloni Gupta

22BCS16659

1763.Longest Nice Substring

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;
    }
}
```



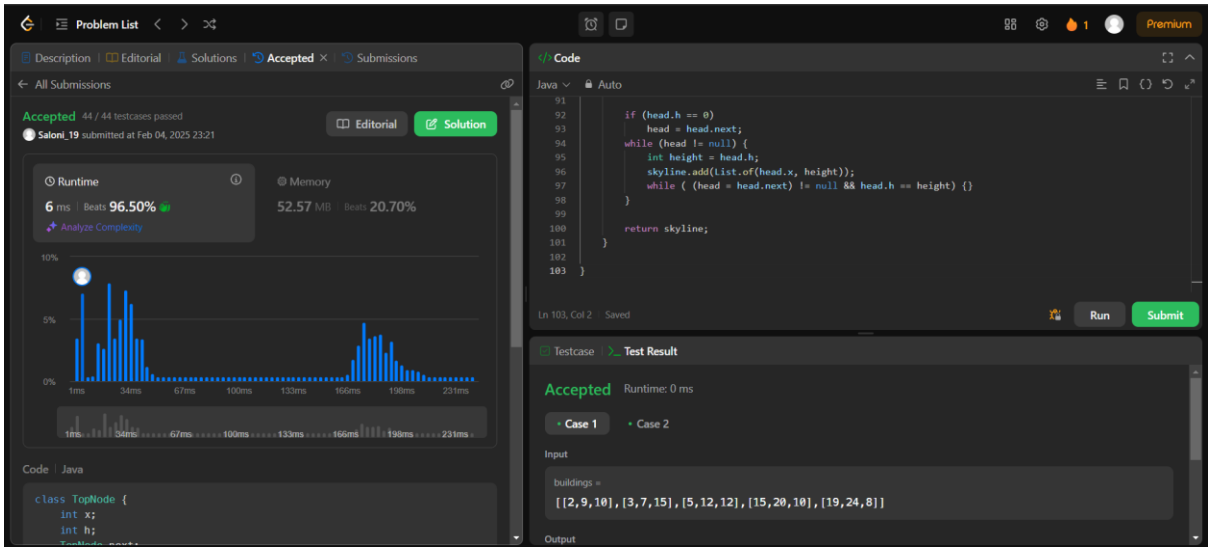
218.The Skyline Problem

Code:

```
class TopNode {
   int x;
   int h;
   TopNode next;
   TopNode() {
   }
   TopNode(int x, int h) {
      this.x = x;
      this.h = h;
   }

   void insert(TopNode n) {
      n.next = next;
      next = n;
   }

}
class Solution {
   static final int LEFT=0, RIGHT=1, HEIGHT=2;
   public List<List<Integer>> getSkyline(int[][] buildings) {
      TopNode head = new TopNode(0,0);
      head.insert(new TopNode(Integer.MAX_VALUE, 0));
      TopNode start = head;

      for (int i = 0; i<buildings.length; i++) {
         int[] b = buildings[i];
         int bL = buildings[i][LEFT];
         int bR = buildings[i][RIGHT];
         int bH = buildings[i][HEIGHT];
         //System.out.println(Arrays.toString(buildings[i]));
         while (bL >= start.next.x) { start = start.next; }
         //System.out.println(start.toString());
         for (TopNode t = start ; bR > t.x; t = t.next) {
            //System.out.println(head.toString());
            if (bH <= t.h) {
               continue;
            }
            TopNode stop = t;
            while (stop.next != null && stop.next.x < bR && stop.next.h <= bH ) {
               stop = stop.next;
            }
```

```
        if (bL <= t.x) {
            if (bR >= stop.next.x) {
                t.next = stop.next;
                t.h = bH;
            }
            else if (t == stop) {
                t.insert(new TopNode(bR,t.h));
                t.h = bH;
                break;
            }
            else {
                stop.x = bR;
                t.h = bH;
                t.next = stop;
                break;
            }
        }
        else {
            if (bR >= stop.next.x) {
                if (t == stop) {
                    t.insert(new TopNode(bL, bH));
                }
                else {
                    t.next = stop;
                    stop.x = bL;
                    stop.h = bH;
                }
                break;
            }
            else if (t == stop) {
                t.insert(new TopNode(bL,bH));
                t.next.insert(new TopNode(bR,t.h));
                break;
            }
            else {
                t.next = stop;
                t.insert(new TopNode(bL,bH));
                stop.x = bR;
                break;
            }
        }
        t = stop;
    }
}
List<List<Integer>> skyline = new ArrayList<>();
```

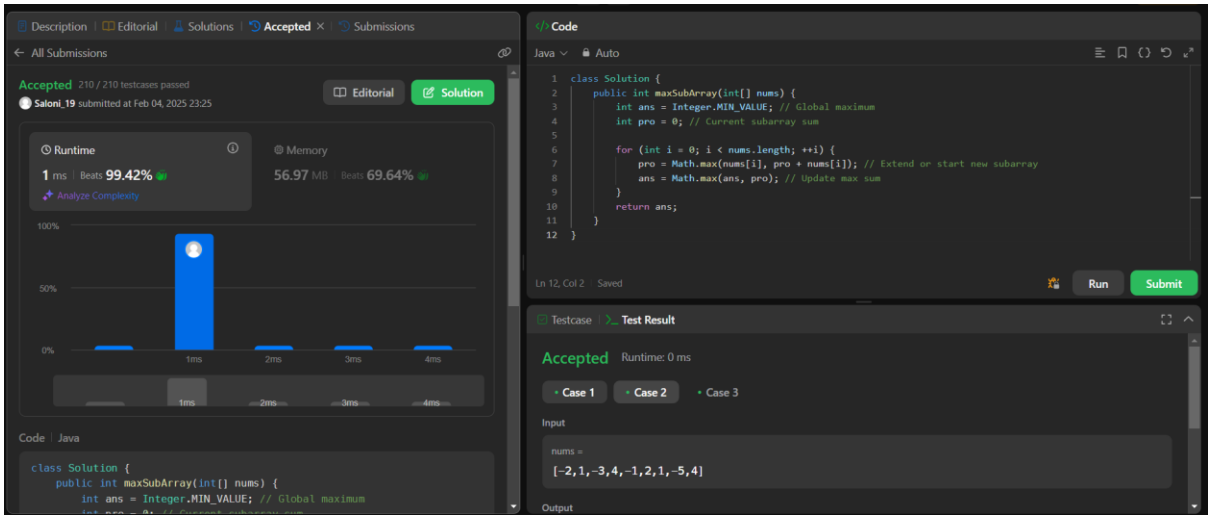
```
        if (head.h == 0)
            head = head.next;
        while (head != null) {
            int height = head.h;
            skyline.add(List.of(head.x, height));
            while ( (head = head.next) != null && head.h == height) {}
        }
        return skyline;
    }
}
```



53.Maximum Subarray

Code:

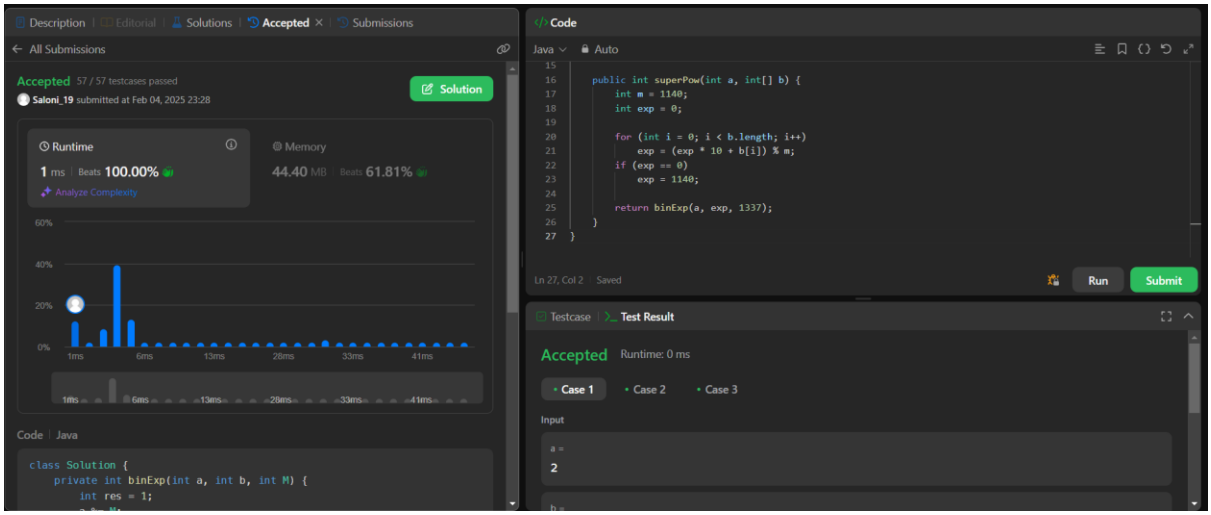
```
class Solution {
    public int maxSubArray(int[] nums) {
        int ans = Integer.MIN_VALUE; // Global maximum
        int pro = 0; // Current subarray sum

        for (int i = 0; i < nums.length; ++i) {
            pro = Math.max(nums[i], pro + nums[i]); // Extend or start new subarray
            ans = Math.max(ans, pro); // Update max sum
        }
        return ans;
    }
}
```



372.Super Pow

Code:

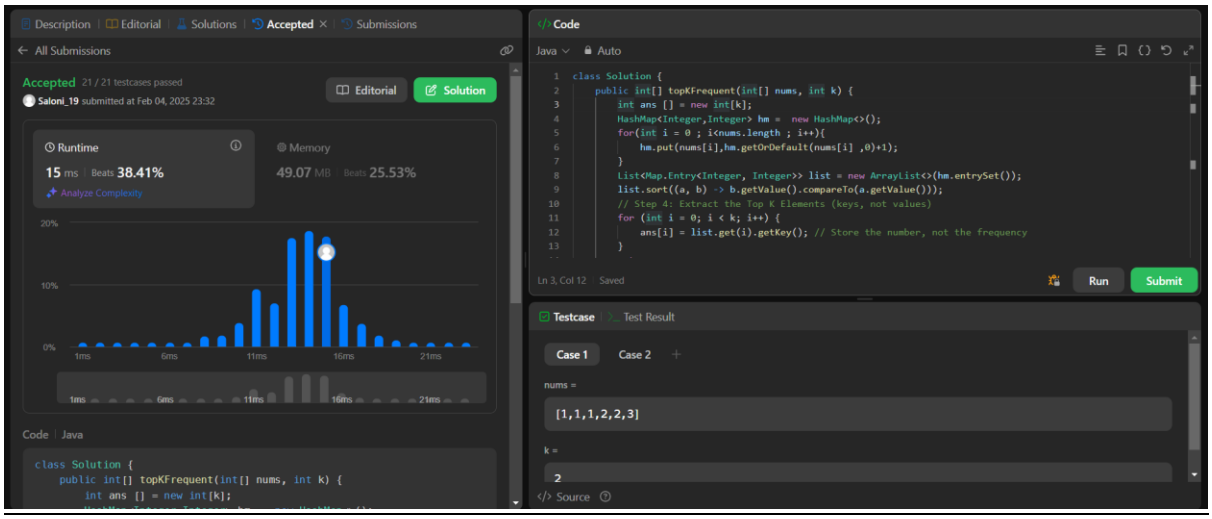
```
class Solution {
    private int binExp(int a, int b, int M) {
        int res = 1;
        a %= M;

        while (b > 0) {
            if ((b & 1) != 0)
                res = (res * a) % M;
            a = (a * a) % M;
            b >>= 1;
        }

        return res;
    }

    public int superPow(int a, int[] b) {
        int m = 1140;
        int exp = 0;

        for (int i = 0; i < b.length; i++)
            exp = (exp * 10 + b[i]) % m;
        if (exp == 0)
            exp = 1140;

        return binExp(a, exp, 1337);
    }
}
```



347. Top K Frequent Elements

Code:

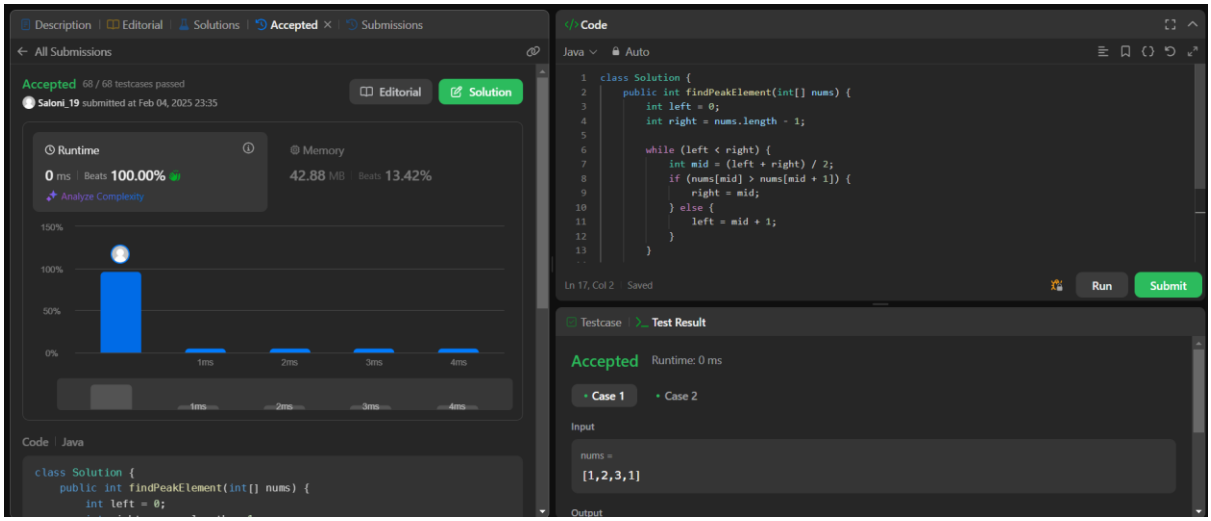
```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        int ans [] = new int[k];
        HashMap<Integer,Integer> hm =  new HashMap<>();
        for(int i = 0 ; i<nums.length ; i++){
            hm.put(nums[i],hm.getOrDefault(nums[i] ,0)+1);
        }
        List<Map.Entry<Integer, Integer>> list = new ArrayList<>(hm.entrySet());
        list.sort((a, b) -> b.getValue().compareTo(a.getValue()));
        // Step 4: Extract the Top K Elements (keys, not values)
        for (int i = 0; i < k; i++) {
            ans[i] = list.get(i).getKey(); // Store the number, not the frequency
        }
        return ans;
    }
}
```



162. Find Peak Element

Code:

```
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;
    }
}
```

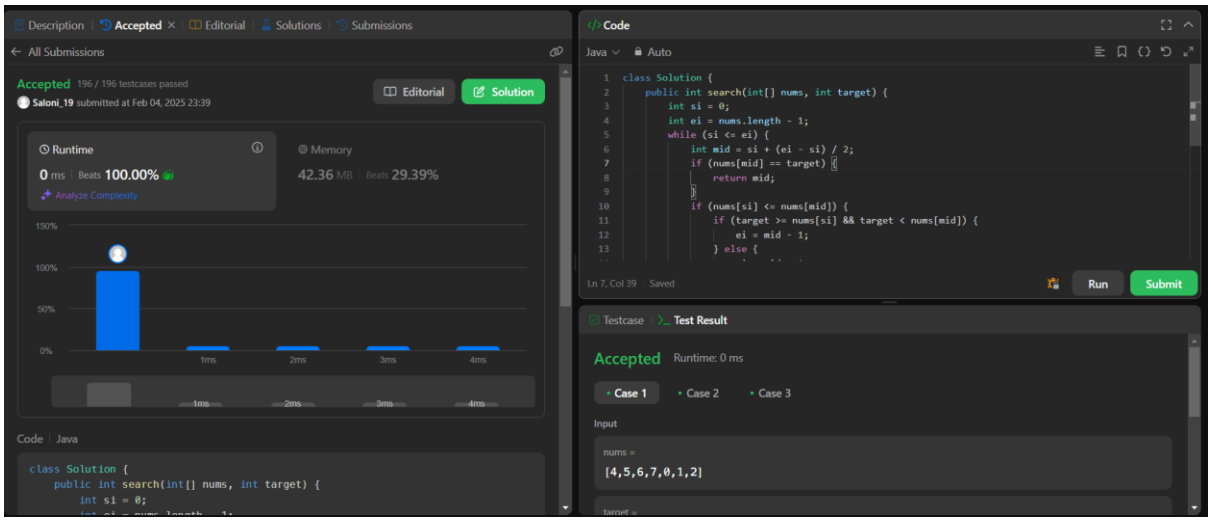


33. Search in Rotated Sorted Array

Code:

```
class Solution {
    public int search(int[] nums, int target) {
        int si = 0;
        int ei = nums.length - 1;
        while (si <= ei) {
            int mid = si + (ei - si) / 2;
            if (nums[mid] == target) {
                return mid;
            }
            if (nums[si] <= nums[mid]) {
                if (target >= nums[si] && target < nums[mid]) {
```

```
            ei = mid - 1;
        } else {
            si = mid + 1;
        }
      }
      else {
        if (target > nums[mid] && target <= nums[ei]) {
            si = mid + 1;
        } else {
            ei = mid - 1;
        }
      }
    }
    return -1;
  }
}
```

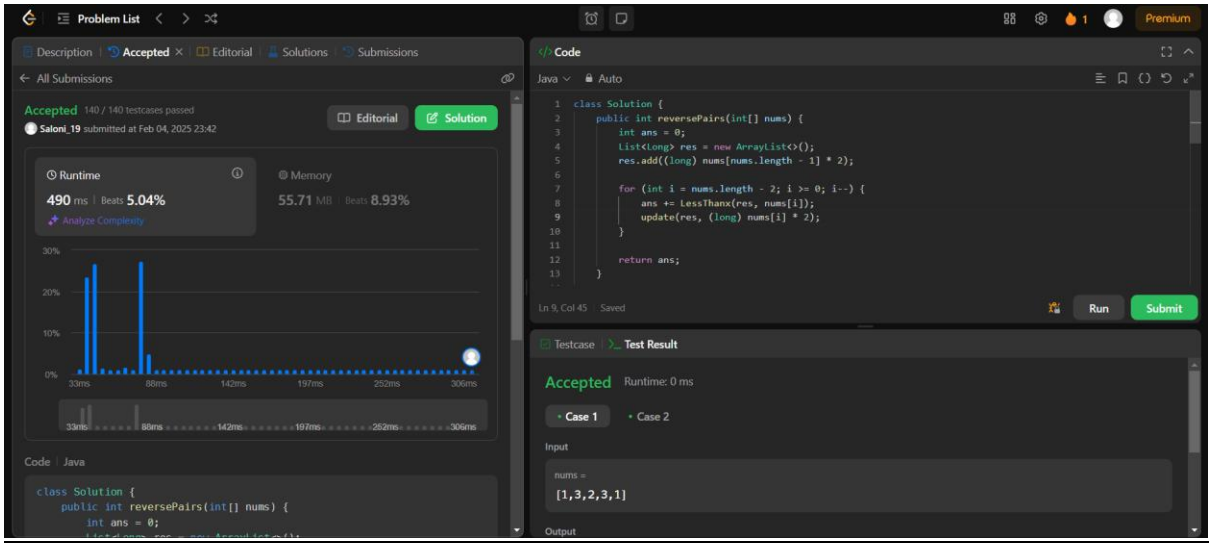


493. Reverse Pairs

Code:

```
class Solution {
  public int reversePairs(int[] nums) {
    int ans = 0;
    List<Long> res = new ArrayList<>();
    res.add((long) nums[nums.length - 1] * 2);

    for (int i = nums.length - 2; i >= 0; i--) {
      ans += LessThanx(res, nums[i]);
      update(res, (long) nums[i] * 2);
    }

    return ans;
  }
```

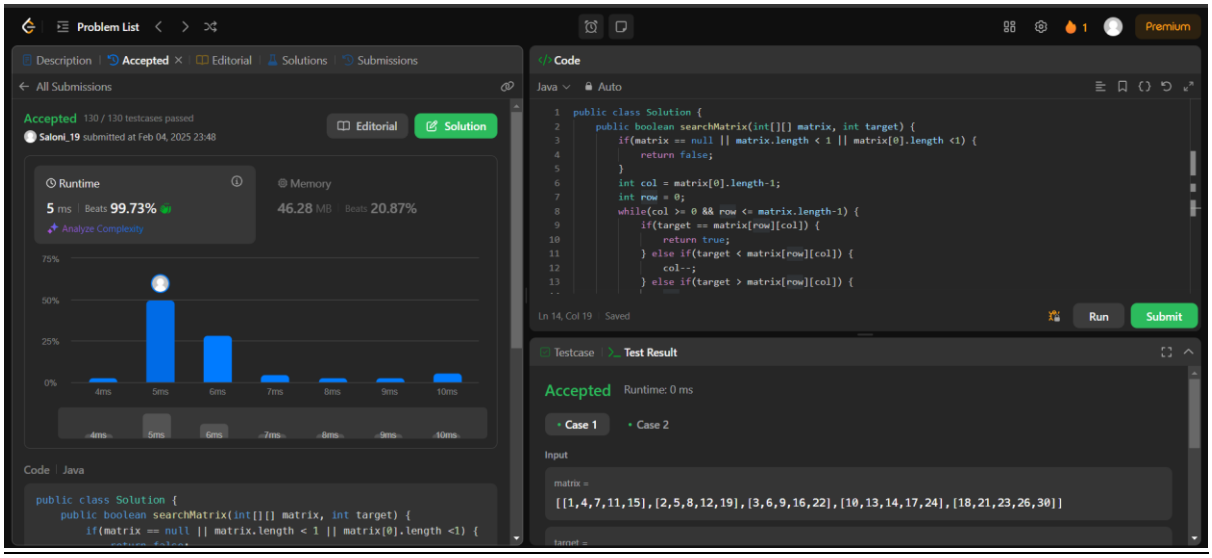
```
    private int LessThanx(List<Long> res, long val) {
        if (res.get(0) >= val) {
            return 0;
        }

        if (res.get(res.size() - 1) < val) {
            return res.size();
        }

        int lo = 0, hi = res.size() - 1;

        while (lo < hi) {
            int mid = (lo + hi) / 2;

            if (res.get(mid) < val) {
                lo = mid + 1;
            } else {
                hi = mid;
            }
        }

        return lo;
    }

    private void update(List<Long> res, long val) {
        int index = Collections.binarySearch(res, val);
        if (index < 0) {
            index = -(index + 1);
        }
        res.add(index, val);
    }
}
```



240. Search a 2D Matrix II

Code:

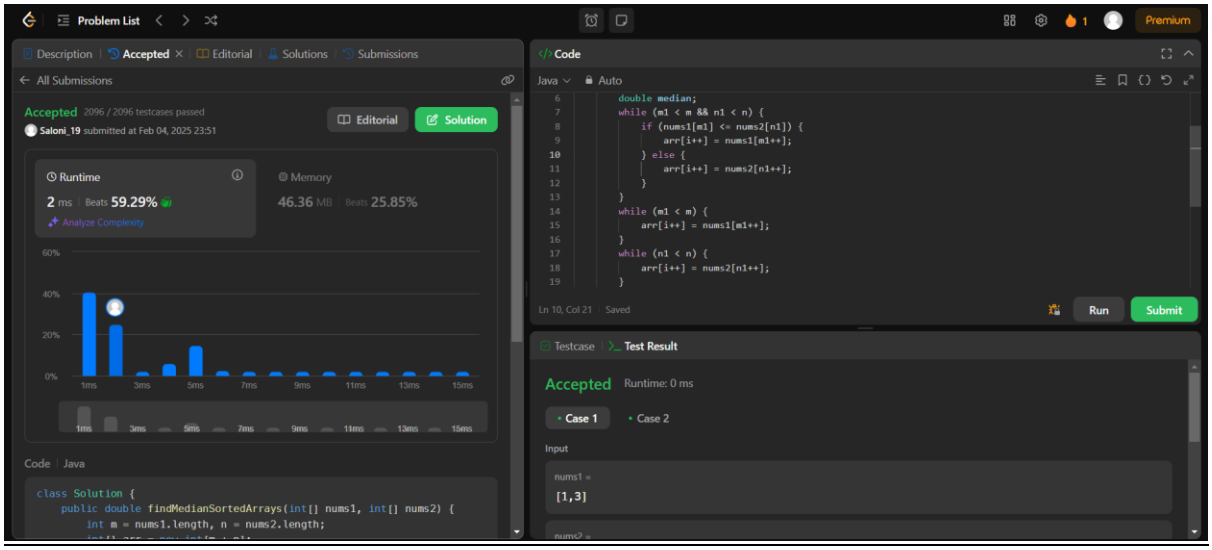
```
public class Solution {
   public boolean searchMatrix(int[][] matrix, int target) {
      if(matrix == null || matrix.length < 1 || matrix[0].length <1) {
         return false;
      }
      int col = matrix[0].length-1;
      int row = 0;
      while(col >= 0 && row <= matrix.length-1) {
    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



4. Median of Two Sorted Arrays

Code:

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int m = nums1.length, n = nums2.length;
        int[] arr = new int[m + n];
        int m1 = 0, n1 = 0, i = 0;
        double median;
        while (m1 < m && n1 < n) {
            if (nums1[m1] <= nums2[n1]) {
                arr[i++] = nums1[m1++];
            } else {
                arr[i++] = nums2[n1++];
            }
        }
        while (m1 < m) {
            arr[i++] = nums1[m1++];
        }
        while (n1 < n) {
            arr[i++] = nums2[n1++];
        }
        int len = m + n;
        if (len % 2 == 0) {
            median = (arr[len / 2 - 1] + arr[len / 2]) / 2.0;
        } else {
            median = arr[len / 2];
        }
        return median;
    }
}
```


Problem List
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 2096 / 2096 testcases passed
Saloni_19 submitted at Feb 04, 2025 23:51
Editorial
Solution
Runtime
2 ms | Beats 59.29%
Analyze Complexity
Memory
46.36 MB | Beats 25.85%
60%
40%
20%
0%
1ms
3ms
5ms
7ms
9ms
11ms
13ms
15ms
Code | Java
class Solution {
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
int m = nums1.length, n = nums2.length;
Code
Java
Auto
double median;
while (m1 < m && n1 < n) {
if (nums1[m1] <= nums2[n1]) {
arr[i++] = nums1[m1++];
} else {
arr[i++] = nums2[n1++];
}
}
while (m1 < m) {
arr[i++] = nums1[m1++];
}
while (n1 < n) {
arr[i++] = nums2[n1++];
}
Ln 10, Col 21 | Saved
Run
Submit
Testcase | Test Result
Accepted Runtime: 0 ms
Case 1
Case 2
Input
nums1 =
[1,3]