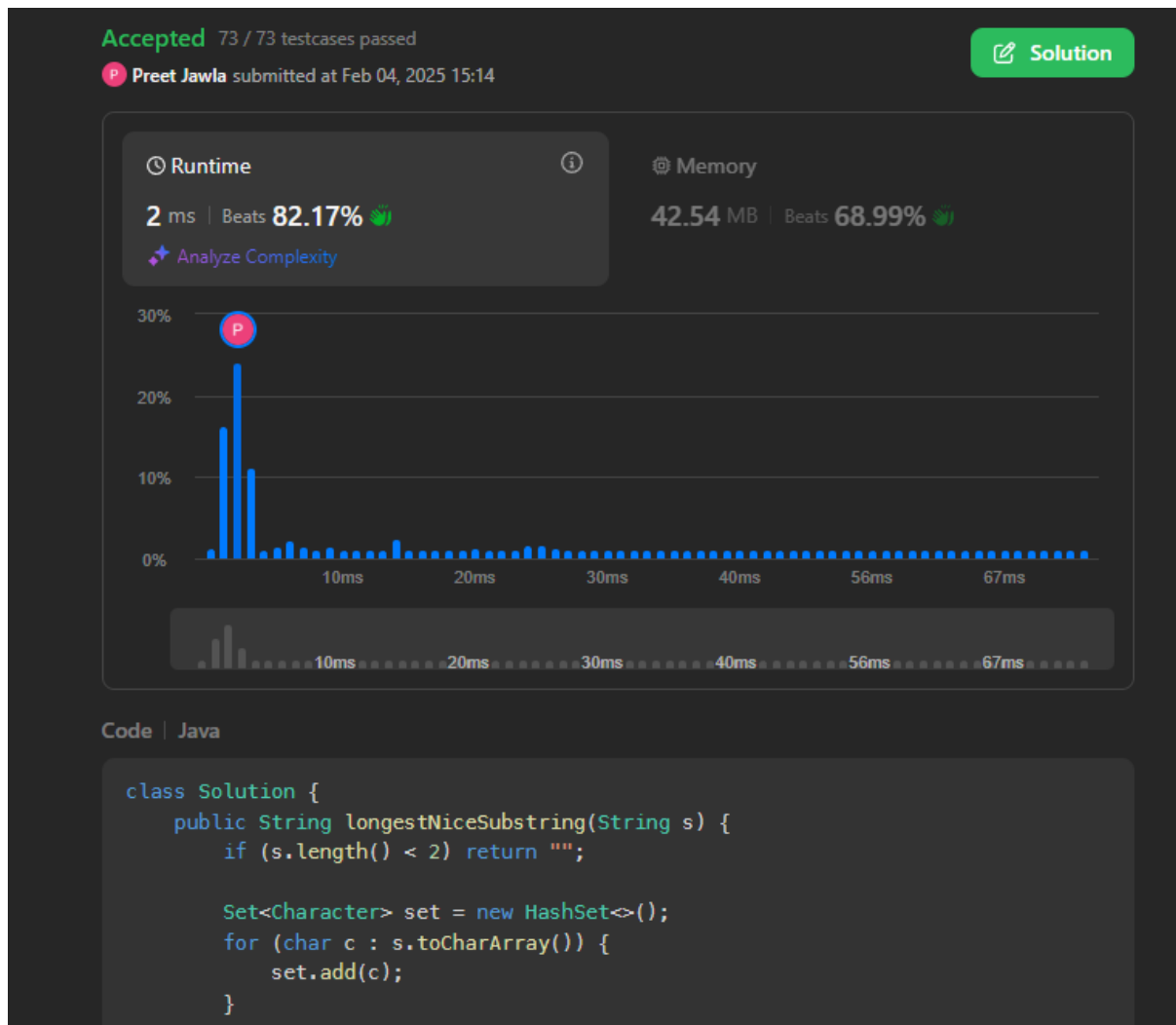


1763. [Longest Nice Substring](#)

Code:

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

SS:



1763. [Longest Nice Substring](#)

Code: public class Solution {

public int reverseBits(int n) {

int result = 0;

for (int i = 0; i < 32; i++) {

// Shift result left to make space for the new bit

result <<= 1;

// Get the last bit of n and add it to result

result |= (n & 1);

// Shift n right to process the next bit

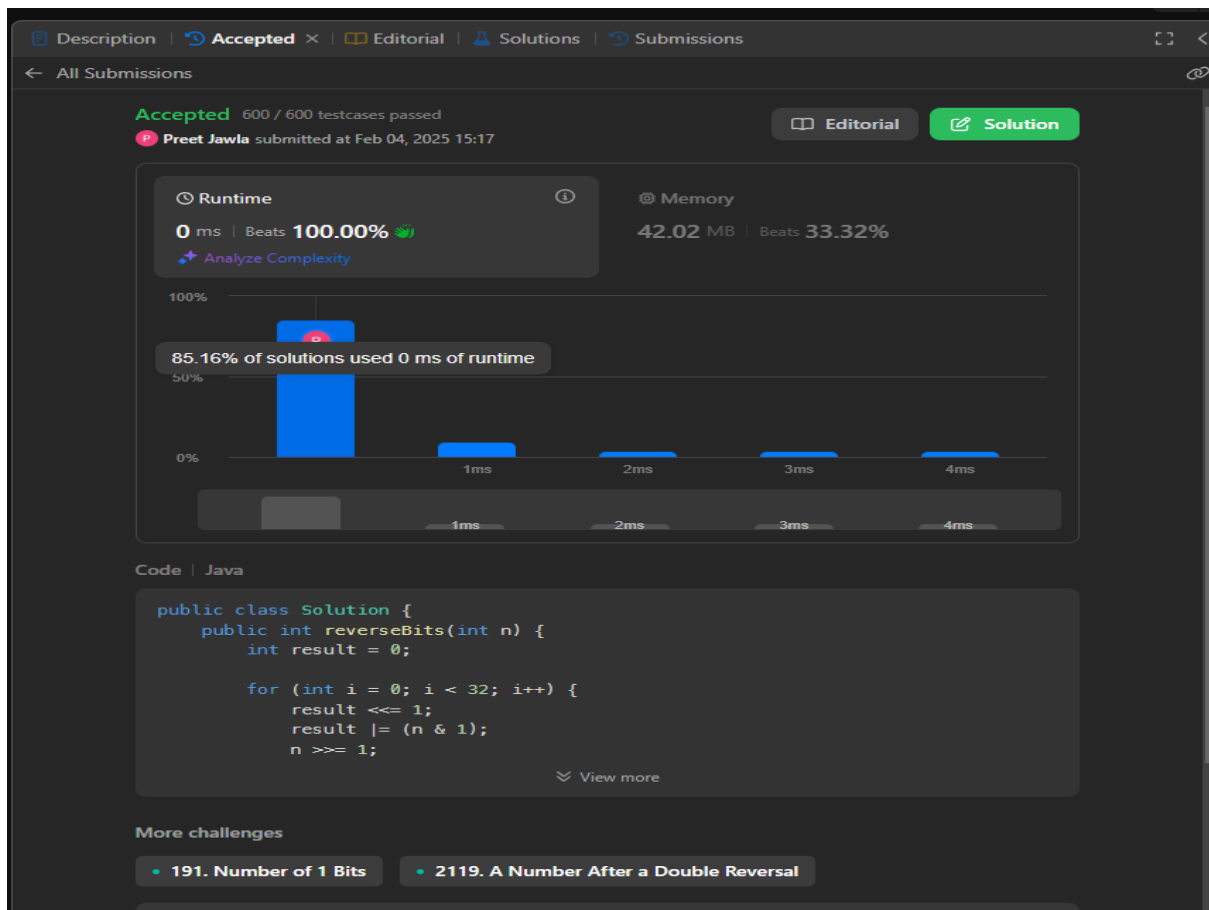
n >>= 1;

```

    }

    return result;
}
}
Ss:

```



190. Reverse Bits

Code:

```

public class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>= 1;
        }
        return count;
    }
}

```

```
}  
  
}  
  
SS:
```

The screenshot shows a LeetCode solution for the 'Number of 1 Bits' problem. The interface is in dark mode. On the left, the 'Problem List' tab is active, showing the problem name 'Number of 1 Bits' and its status 'Accepted'. Below this, a 'Runtime' section shows '0 ms' and '100.00%' efficiency. A 'Memory' section shows '41.02 MB' and '22.42%' efficiency. A bar chart shows the runtime performance across different test cases. The 'Code' section displays the following Java code:

```
public class Solution {  
    public int hammingWeight(int n) {  
        int count = 0;  
        while (n != 0) {  
            count += (n & 1);  
            n >>= 1;  
        }  
        return count;  
    }  
}
```

On the right, the 'Test Result' tab is active, showing the test case 'Case 1' with input 'n = 11' and output '3'. The status is 'Accepted'.

191. [Number of 1 Bits](#)

```
Code: public class Solution {  
    public int hammingWeight(int n) {  
        int count = 0;  
        while (n != 0) {  
            count += (n & 1);  
            n >>= 1;  
        }  
        return count;  
    }  
}
```

Ss:

The screenshot displays the LeetCode interface for the problem "Number of 1-bits". The submission is marked as "Accepted" with 998/998 test cases passed. The runtime is 0 ms and memory is 41.02 MB. The code editor shows a Java solution for the "Number of 1-bits" problem, which uses a while loop to count the number of 1-bits in a given integer n. The test result panel shows "Accepted" with a runtime of 0 ms and memory of 41.02 MB. The code is as follows:

```
public class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>= 1;
        }
        return count;
    }
}
```

53. [Maximum Subarray](#)

Code: public class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0], 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;

}

}

The screenshot displays the LeetCode interface for the 'Maximum Subarray' problem. The top navigation bar includes links for 'Problem List', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'. The main content area shows the problem description, a performance graph, and a code editor. The performance graph indicates a runtime of 1 ms and memory usage of 57.21 MB. The code editor contains a Java solution for the 'Maximum Subarray' problem. The test result section shows the input array [-2, 1, -3, 4, -1, 2, 1, -5, 4] and the output 6.

```

public class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0], 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;
    }
}

```

240. Search a 2D Matrix II

Code: public class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

int row = 0, col = matrix[0].length - 1;

while (row < matrix.length && col >= 0) {

if (matrix[row][col] == target) return true;

else if (matrix[row][col] > target) col--;

else row++;

}

return false;

}

}

SS:

The screenshot displays the LeetCode submission page for the problem "Search a 2D Matrix". The submission is accepted, with a runtime of 5 ms and a memory usage of 46.27 MB. The Java code is as follows:

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

The test result shows the input matrix as `[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]` and the target as `5`. The output is `true`, which matches the expected result.

372. Super Pow

Code: public class Solution {

private static final int MOD = 1337;

public int superPow(int a, int[] b) {

a %= MOD;

int result = 1;

for (int digit : b) {

result = powMod(result, 10) * powMod(a, digit) % MOD;

}

return result;

}

private int powMod(int x, int y) {

int res = 1;

while (y > 0) {

if (y % 2 == 1) res = res * x % MOD;

```

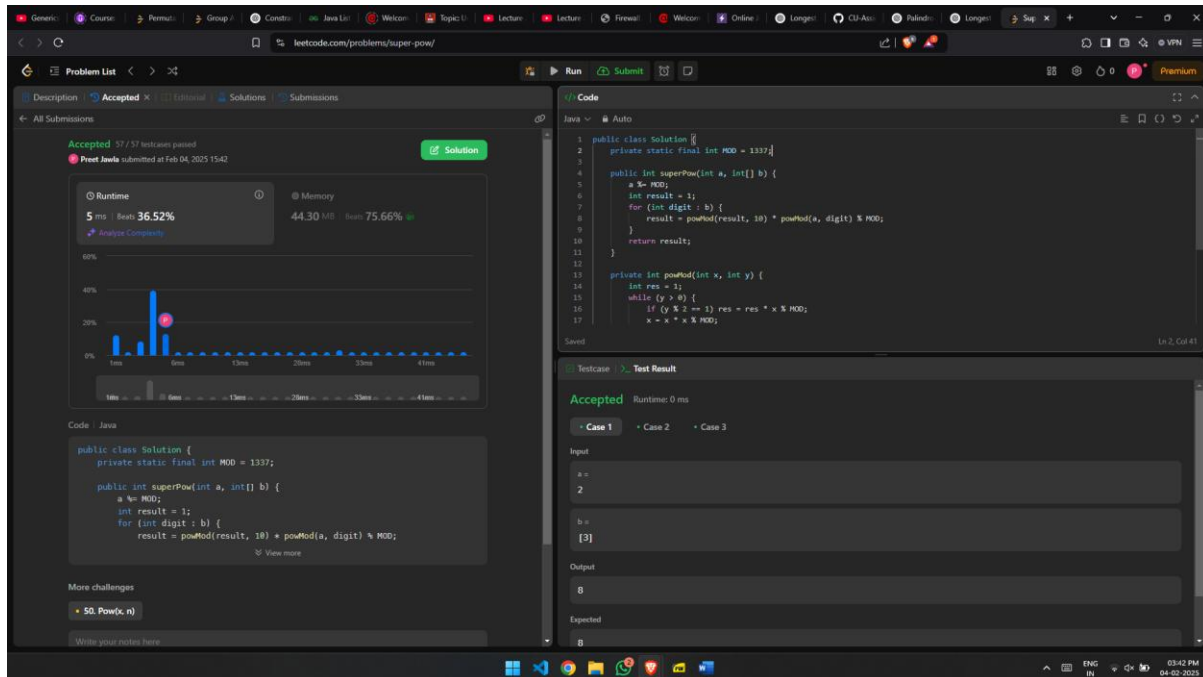
        x = x * x % MOD;

        y /= 2;
    }

    return res;
}
}

```

SS:



932. Beautiful Array

Code: import java.util.*;

```

public class Solution {

    public int[] beautifulArray(int n) {

        List<Integer> res = new ArrayList<>();

        res.add(1);

        while (res.size() < n) {

            List<Integer> next = new ArrayList<>();

            for (int num : res) if (num * 2 - 1 <= n) next.add(num * 2 - 1);

```



```

        for (int num : res) if (num * 2 <= n) next.add(num * 2);

        res = next;
    }

    return res.stream().mapToInt(i -> i).toArray();
}
}

```

The screenshot shows the LeetCode editor for the 'Beautiful Array' problem. The code is written in Java and is accepted. The runtime is 5 ms, and the memory usage is 42.74 MB. The test result shows Case 1 and Case 2 both passing.

```

public class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> res = new ArrayList<>();
        res.add(1);

        while (res.size() < n) {
            List<Integer> next = new ArrayList<>();
            for (int num : res) if (num * 2 - 1 <= n) next.add(num * 2 - 1);
            for (int num : res) if (num * 2 <= n) next.add(num * 2);
            res = next;
        }

        return res.stream().mapToInt(i -> i).toArray();
    }
}

```

Testcase: Case 1, Case 2

Input: n = 4

Output: [1, 3, 2, 4]

Expected: [2, 1, 4, 3]

218. The Skyline Problem

Code: import java.util.*;

```

public class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> events = new ArrayList<>();

        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]});
            events.add(new int[]{b[1], b[2]});
        }
    }
}

```

```

events.sort((a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));

List<List<Integer>> result = new ArrayList<>();

TreeMap<Integer, Integer> heights = new TreeMap<>(Collections.reverseOrder());

heights.put(0, 1);

int prevHeight = 0;

for (int[] event : events) {
    if (event[1] < 0) heights.put(-event[1], heights.getOrDefault(-event[1], 0) + 1);
    else {
        if (heights.get(event[1]) == 1) heights.remove(event[1]);
        else heights.put(event[1], heights.get(event[1]) - 1);
    }

    int currentHeight = heights.firstKey();
    if (currentHeight != prevHeight) {
        result.add(Arrays.asList(event[0], currentHeight));
        prevHeight = currentHeight;
    }
}

return result;
}

```



```

    }

    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++];

    System.arraycopy(temp, 0, nums, left, temp.length);
}

```

The screenshot shows a LeetCode submission for the 'Reverse Pairs' problem. The submission is accepted with a runtime of 41 ms and a memory usage of 55.51 MB. The code is in Java and implements a merge sort algorithm to count reverse pairs. The test result shows the input [1, 3, 2, 3, 1] and the output 2.

SS:

88. Merge Sorted Array

Code: public class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

int i = m - 1, j = n - 1, k = m + n - 1;

while (i >= 0 && j >= 0) {

if (nums1[i] > nums2[j]) {

nums1[k--] = nums1[i--];

} else {

nums1[k--] = nums2[j--];

}

}

while (j >= 0) {

nums1[k--] = nums2[j--];

}

}

}

The screenshot shows a LeetCode submission for the 'Merge Sorted Array' problem. The submission is accepted, showing a runtime of 0 ms and memory usage of 42.47 MB. The code is in Java and implements a merge sort algorithm. The test results show the input arrays [1, 2, 3, 0, 0, 0] and [2, 5, 6] being merged into [1, 2, 3, 5, 6, 6].

75. Sort Colors

Code: public class Solution {

public void sortColors(int[] nums) {

int low = 0, mid = 0, high = nums.length - 1;

while (mid <= high) {

if (nums[mid] == 0) {

swap(nums, low++, mid++);

} else if (nums[mid] == 1) {

mid++;

} else {

swap(nums, mid, high--);

}

}

}

private void swap(int[] nums, int i, int j) {

```

int temp = nums[i];

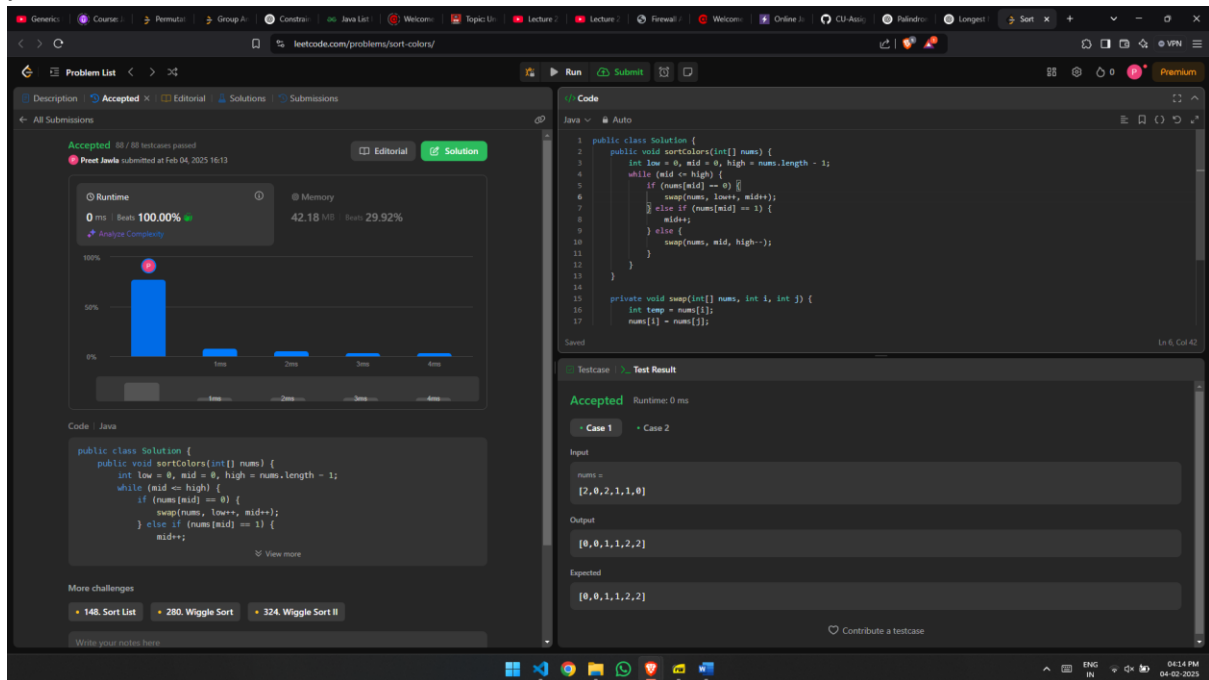
nums[i] = nums[j];

nums[j] = temp;

}

}

```



SS:

347. Top K Frequent Elements

Code: public class Solution {

```

    public int[] topKFrequent(int[] nums, int k) {

```

```

        Map<Integer, Integer> freqMap = new HashMap<>();

```

```

        for (int num : nums) {

```

```

            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);

```

```

        }

```

```

        PriorityQueue<Map.Entry<Integer, Integer>> pq = new PriorityQueue<>((a, b) -> a.getValue() - b.getValue());

```

```

        for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {

```

[illegible]

56. Merge Intervals

```
Code: public class Solution {
    public int[][] merge(int[][] intervals) {
```



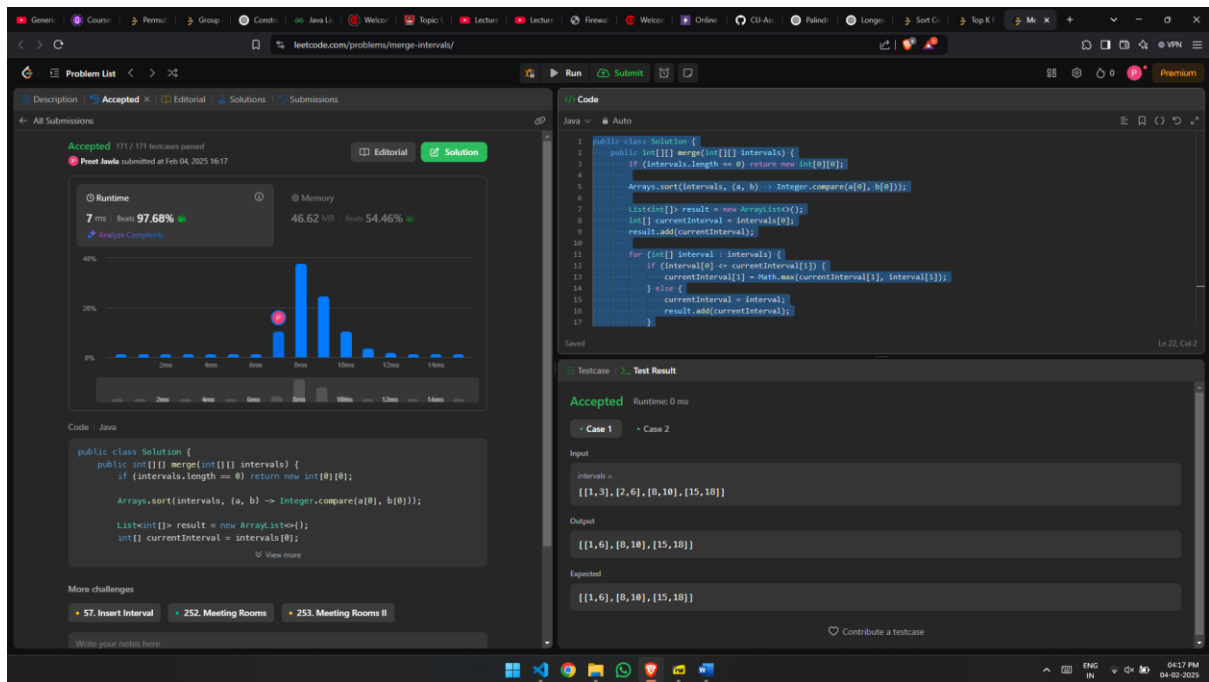
```
if (intervals.length == 0) return new int[0][0];

Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

List<int[]> result = new ArrayList<>();
int[] currentInterval = intervals[0];
result.add(currentInterval);

for (int[] interval : intervals) {
    if (interval[0] <= currentInterval[1]) {
        currentInterval[1] = Math.max(currentInterval[1], interval[1]);
    } else {
        currentInterval = interval;
        result.add(currentInterval);
    }
}

return result.toArray(new int[result.size()][]);
}
```



SS:

33. Search in Rotated Sorted Array

Code: class Solution {

public int search(int[] nums, int target) {

int left = 0, right = nums.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) return mid;

if (nums[left] <= nums[mid]) {

if (nums[left] <= target && target < nums[mid]) right = mid - 1;

else left = mid + 1;

} else {

if (nums[mid] < target && target <= nums[right]) left = mid + 1;

else right = mid - 1;

}

}

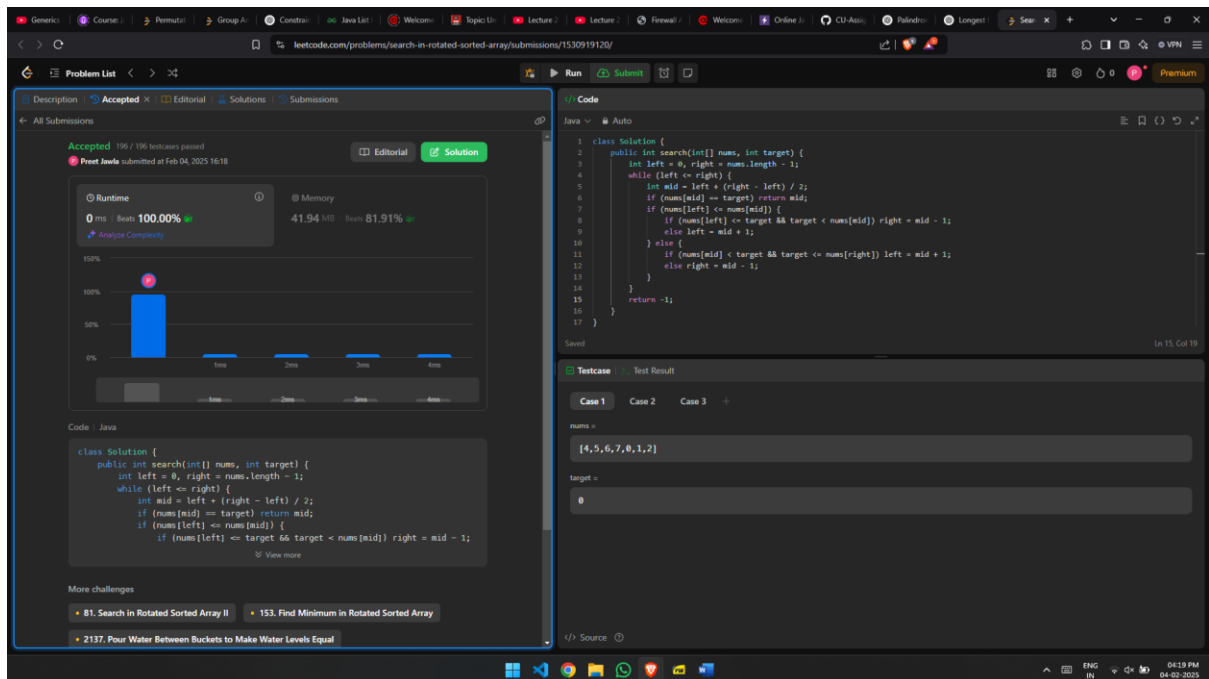
return -1;

```

}

}

```



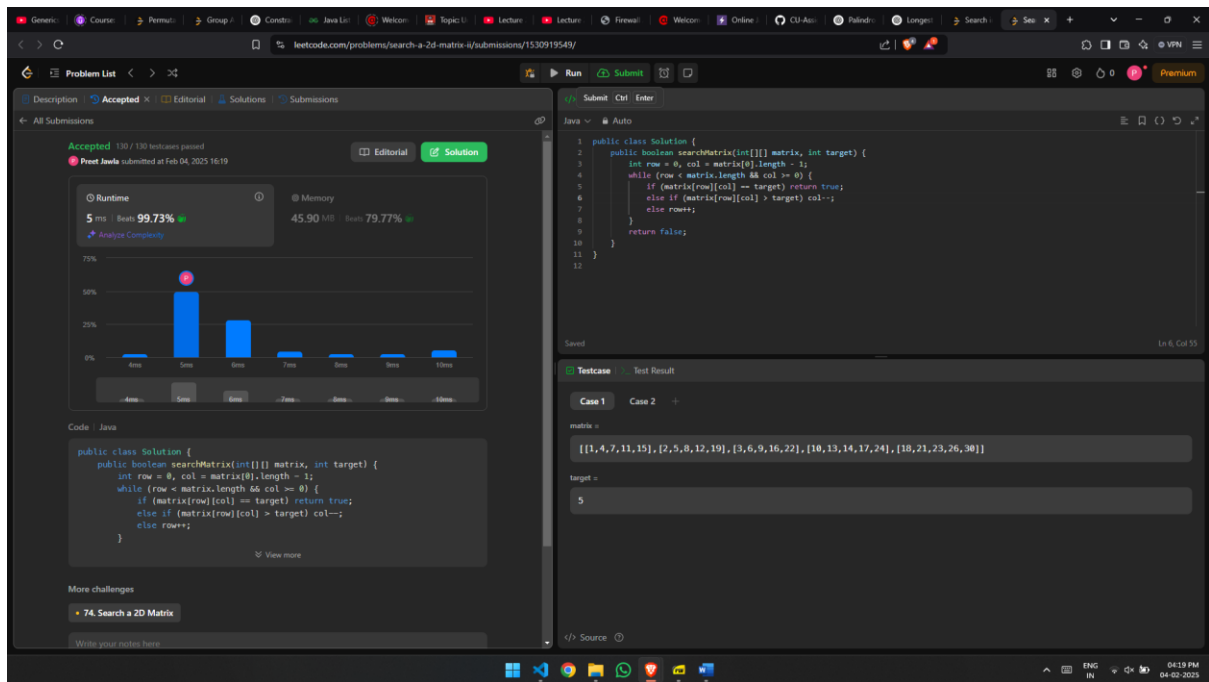
SS:

240. Search a 2D Matrix II

```

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

```



378. Kth Smallest Element in a Sorted Matrix

Code: `import java.util.*;`

```
public class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        PriorityQueue<int[]> minHeap = new PriorityQueue<>((a, b) -> a[0] - b[0]);

        // Push the first element of each row into the heap
        for (int i = 0; i < n; i++) {
            minHeap.offer(new int[]{matrix[i][0], i, 0});
        }

        int count = 0;
        while (!minHeap.isEmpty()) {
            int[] current = minHeap.poll();
            count++;
        }
    }
}
```

```

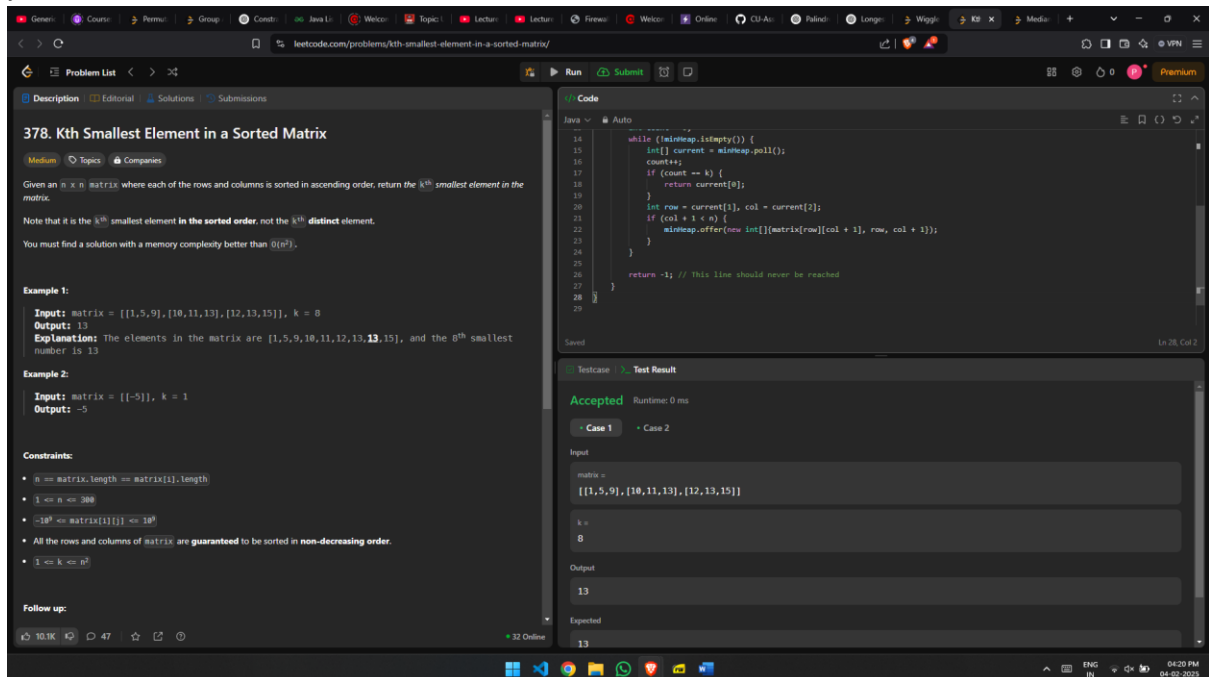
        if (count == k) {
            return current[0];
        }

        int row = current[1], col = current[2];

        if (col + 1 < n) {
            minHeap.offer(new int[]{matrix[row][col + 1], row, col + 1});
        }
    }

    return -1; // This line should never be reached
}
}

```



SS:

4. Median of Two Sorted Arrays

Code: class Solution {

```

    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        // Ensure nums1 is the smaller array for binary search efficiency

```

```
if (nums1.length > nums2.length) {  
    return findMedianSortedArrays(nums2, nums1);  
}  
  
int m = nums1.length;  
int n = nums2.length;  
int totalLeft = (m + n + 1) / 2; // Number of elements in the left  
partition  
  
int left = 0, right = m; // Binary search range for nums1  
  
while (left <= right) {  
    int partition1 = (left + right) / 2; // Partition point in nums1  
    int partition2 = totalLeft - partition1; // Partition point in  
nums2  
  
    // Edge cases where partition is at array boundaries  
    int maxLeft1 = (partition1 == 0) ? Integer.MIN_VALUE :  
nums1[partition1 - 1];  
    int minRight1 = (partition1 == m) ? Integer.MAX_VALUE :  
nums1[partition1];  
    int maxLeft2 = (partition2 == 0) ? Integer.MIN_VALUE :  
nums2[partition2 - 1];  
    int minRight2 = (partition2 == n) ? Integer.MAX_VALUE :  
nums2[partition2];
```

```

// Check if the partition is correct
if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {
    // If total length is odd, return max of left partition
    if ((m + n) % 2 == 1) {
        return Math.max(maxLeft1, maxLeft2);
    }
    // If total length is even, return average of middle two
elements
    return (Math.max(maxLeft1, maxLeft2) +
Math.min(minRight1, minRight2)) / 2.0;
} else if (maxLeft1 > minRight2) {
    // Move partition1 to the left
    right = partition1 - 1;
} else {
    // Move partition1 to the right
    left = partition1 + 1;
}
}

throw new IllegalArgumentException("Input arrays are not
sorted.");
}
}

```

leetcode.com/problems/median-of-two-sorted-arrays/

Problem List

Accepted

Editorial

Solutions

Submissions

All Submissions

Runtime

1 ms

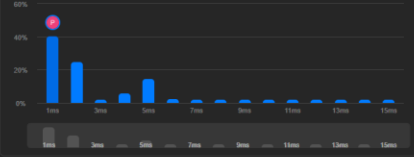
Beats: 100.00%

Analyze Complexity

Memory

46.14 MB

Beats: 59.63%



| Language | Runtime (ms) |
|--------------|--------------|
| Java | 1 |
| C++ | 25 |
| Python | 10 |
| JavaScript | 5 |
| Go | 10 |
| Rust | 10 |
| C# | 10 |
| PHP | 10 |
| Perl | 10 |
| Python 3 | 10 |
| Java 8 | 10 |
| Scala | 10 |
| Swift | 10 |
| Kotlin | 10 |
| Objective-C | 10 |
| VB.NET | 10 |
| F# | 10 |
| OCaml | 10 |
| Perl 6 | 10 |
| Crystal | 10 |
| Erlang | 10 |
| Julia | 10 |
| Fortran | 10 |
| Fortran 90 | 10 |
| Fortran 95 | 10 |
| Fortran 2003 | 10 |
| Fortran 2008 | 10 |
| Fortran 2013 | 10 |
| Fortran 2015 | 10 |
| Fortran 2018 | 10 |
| Fortran 2020 | 10 |
| Fortran 2022 | 10 |
| Fortran 2023 | 10 |
| Fortran 2024 | 10 |
| Fortran 2025 | 10 |
| Fortran 2026 | 10 |
| Fortran 2027 | 10 |
| Fortran 2028 | 10 |
| Fortran 2029 | 10 |
| Fortran 2030 | 10 |
| Fortran 2031 | 10 |
| Fortran 2032 | 10 |
| Fortran 2033 | 10 |
| Fortran 2034 | 10 |
| Fortran 2035 | 10 |
| Fortran 2036 | 10 |
| Fortran 2037 | 10 |
| Fortran 2038 | 10 |
| Fortran 2039 | 10 |
| Fortran 2040 | 10 |
| Fortran 2041 | 10 |
| Fortran 2042 | 10 |
| Fortran 2043 | 10 |
| Fortran 2044 | 10 |
| Fortran 2045 | 10 |
| Fortran 2046 | 10 |
| Fortran 2047 | 10 |
| Fortran 2048 | 10 |
| Fortran 2049 | 10 |
| Fortran 2050 | 10 |

Code | Java

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        // Ensure nums1 is the smaller array for binary search efficiency
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int m = nums1.length;
        int n = nums2.length;
        int totalLeft = (m + n + 1) / 2; // Number of elements in the left partition

        int left = 0, right = m; // Binary search range for nums1

        while (left < right) {
            int partition1 = (left + right) / 2; // Partition point in nums1
            int partition2 = totalLeft - partition1; // Partition point in nums2

            // Edge cases where partition is at array boundaries
            if (partition1 == 0) {
                // All elements from nums1 are in the left partition
                return Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
            }
            if (partition2 == 0) {
                // All elements from nums2 are in the left partition
                return Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
            }
            if (partition1 == m) {
                // All elements from nums1 are in the right partition
                return Math.min(nums1[partition1], nums2[partition2]);
            }
            if (partition2 == n) {
                // All elements from nums2 are in the right partition
                return Math.min(nums1[partition1], nums2[partition2]);
            }

            int maxLeft = Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
            int minRight = Math.min(nums1[partition1], nums2[partition2]);

            if (maxLeft < minRight) {
                left = partition1;
            } else {
                right = partition1;
            }
        }

        return Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
    }
}
```

View more

More challenges

2387. Median of a Row Wise Sorted Matrix

Write your notes here

Code

Auto

```
2 public double findMedianSortedArrays(int[] nums1, int[] nums2) {
3     // Ensure nums1 is the smaller array for binary search efficiency
4     if (nums1.length > nums2.length) {
5         return findMedianSortedArrays(nums2, nums1);
6     }
7
8     int m = nums1.length;
9     int n = nums2.length;
10    int totalLeft = (m + n + 1) / 2; // Number of elements in the left partition
11
12    int left = 0, right = m; // Binary search range for nums1
13
14    while (left < right) {
15        int partition1 = (left + right) / 2; // Partition point in nums1
16        int partition2 = totalLeft - partition1; // Partition point in nums2
17
18        // Edge cases where partition is at array boundaries
19        if (partition1 == 0) {
20            // All elements from nums1 are in the left partition
21            return Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
22        }
23        if (partition2 == 0) {
24            // All elements from nums2 are in the left partition
25            return Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
26        }
27        if (partition1 == m) {
28            // All elements from nums1 are in the right partition
29            return Math.min(nums1[partition1], nums2[partition2]);
30        }
31        if (partition2 == n) {
32            // All elements from nums2 are in the right partition
33            return Math.min(nums1[partition1], nums2[partition2]);
34        }
35
36        int maxLeft = Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
37        int minRight = Math.min(nums1[partition1], nums2[partition2]);
38
39        if (maxLeft < minRight) {
40            left = partition1;
41        } else {
42            right = partition1;
43        }
44    }
45
46    return Math.max(nums1[partition1 - 1], nums2[partition2 - 1]);
47 }
```

Ln 11, Col 1

Testcase

Test Result

Case 1

Case 2

+

nums1 =

[1,3]

nums2 =

[2]

</> Source

04:21 PM 04-02-2025