Experiment 5

Student Name: Shubham Kumar Sharma **UID:** 22BET10353
Branch: IT **Section/Group:** 22BET_IOT-702/A
Semester: 6th **Date of Performance:** 20/02/25
Subject Name: Advance Programming-II **Subject Code:** 22ITP-367

Problem: 1: Merge Intervals

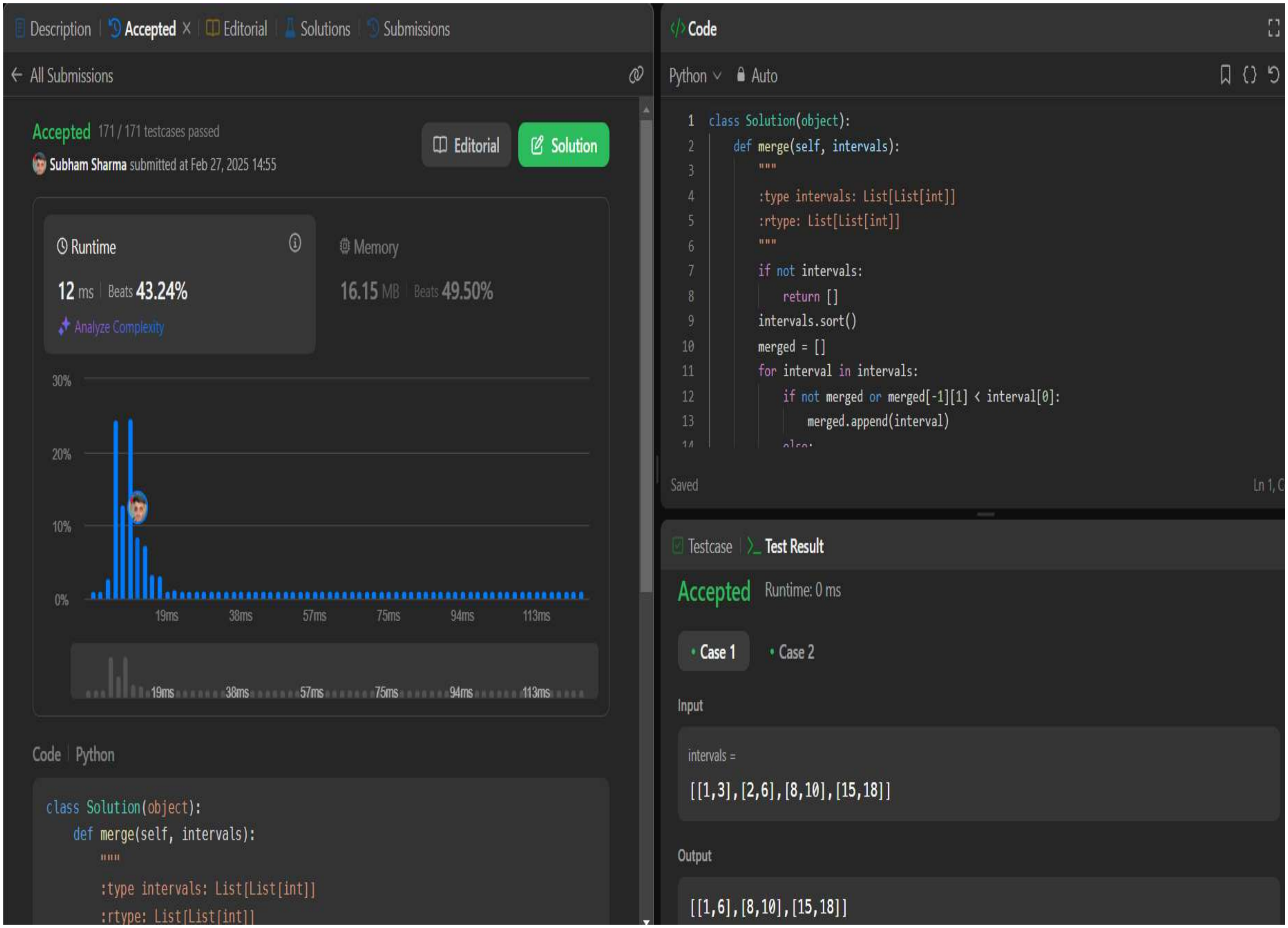
Problem Statement: Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

1. **Objective:** Find the merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input..

2. **Code:**

```
class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: List[List[int]]
        """
        if not intervals:
            return []
        intervals.sort()
        merged = []
        for interval in intervals:
            if not merged or merged[-1][1] < interval[0]:
                merged.append(interval)
            else:
                merged[-1][1] = max(merged[-1][1], interval[1])

        return merged
```

3. Result:



Accepted 171 / 171 testcases passed

Subham Sharma submitted at Feb 27, 2025 14:55

Runtime 12 ms | Beats 43.24%

Memory 16.15 MB | Beats 49.50%

Code | Python

```
class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: List[List[int]]
```

Code Python

```
class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[List[int]]
        :rtype: List[List[int]]
        """
        if not intervals:
            return []
        intervals.sort()
        merged = []
        for interval in intervals:
            if not merged or merged[-1][1] < interval[0]:
                merged.append(interval)
            else:
```

Accepted Runtime: 0 ms

Input

```
intervals =
[[1,3],[2,6],[8,10],[15,18]]
```

Output

```
[[1,6],[8,10],[15,18]]
```

Problem 2: Search in Rotated Sorted Array

Problem Statement: There is an integer array nums sorted in ascending order (with distinct values).

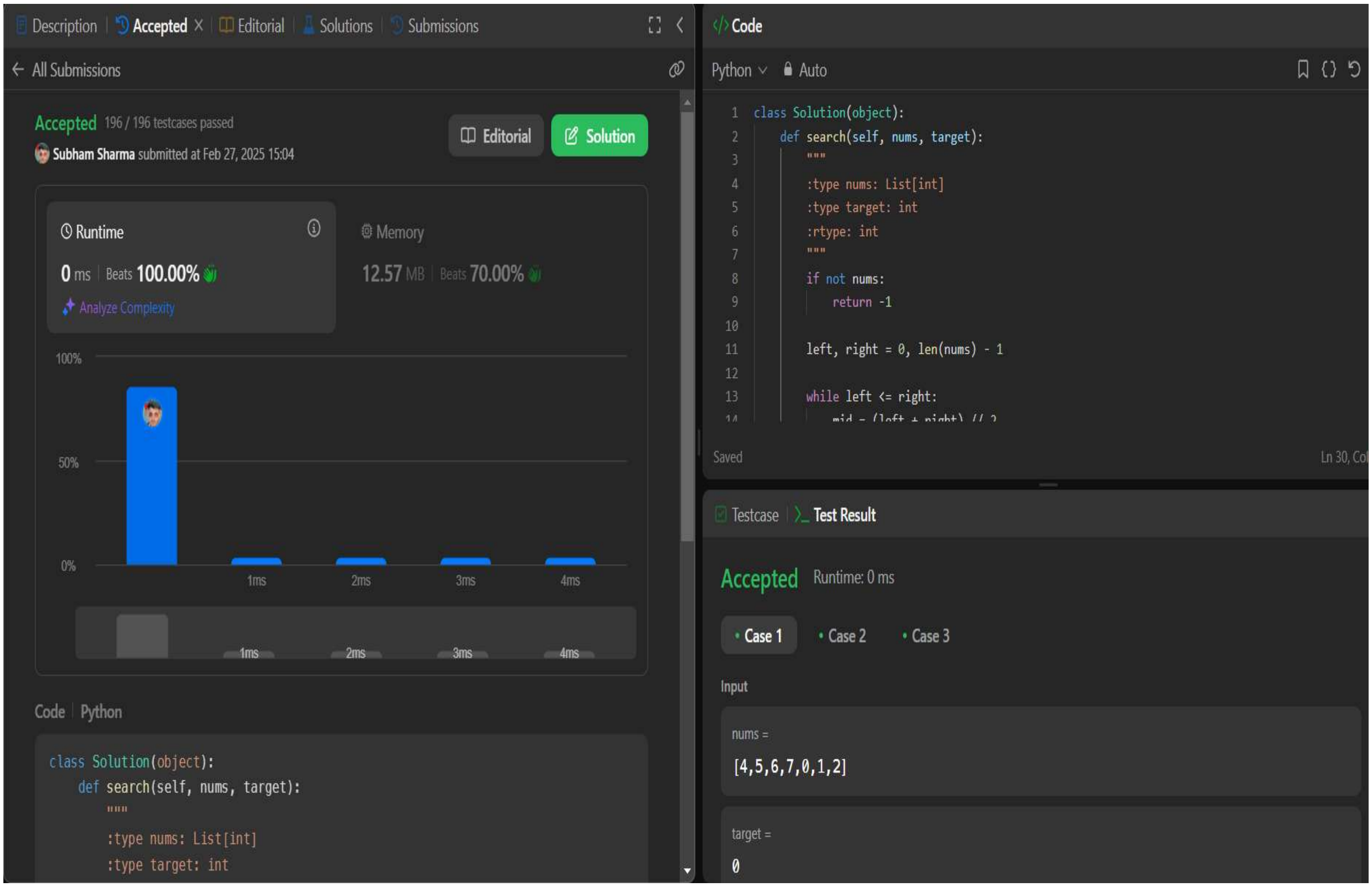
Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

1. **Objective:** Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.
   You must write an algorithm with O(log n) runtime complexity.

2. Code:

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums:
            return -1

        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2

            if nums[mid] == target:
                return mid

            if nums[left] <= nums[mid]:
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] < target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1

        return -1
```

3. Result:



Accepted 196 / 196 testcases passed

Subham Sharma submitted at Feb 27, 2025 15:04

Runtime 0 ms | Beats 100.00%

Memory 12.57 MB | Beats 70.00%

Code | Python

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums:
            return -1

        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2
```

Accepted Runtime: 0 ms

Input

nums =

[4,5,6,7,0,1,2]

target =

0

Problem 3: K th smallest element in a sorted matrix

Problem Statement: Given an n x n matrix where each of the rows and columns is sorted in ascending order, return the kth smallest element in the matrix.

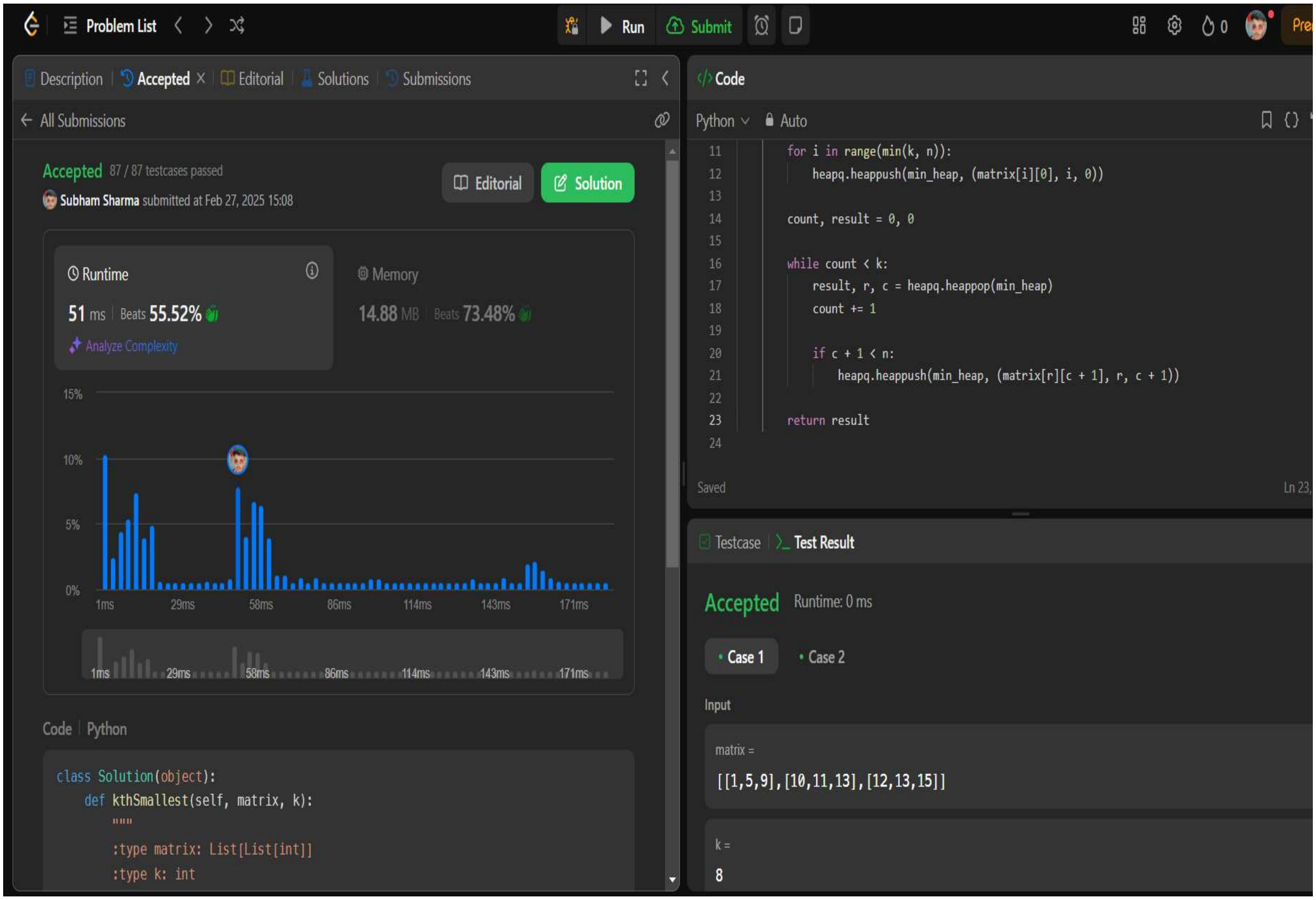
1. Objective: that it is the (k)th smallest element in the sorted order, not the kth distinct element.

You must find a solution with a memory complexity better than O(n2).

2. Code:

```
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        n = len(matrix)
        min_heap = []

        for i in range(min(k, n)):
            heapq.heappush(min_heap, (matrix[i][0], i, 0))

        count, result = 0, 0

        while count < k:
            result, r, c = heapq.heappop(min_heap)
            count += 1

            if c + 1 < n:
                heapq.heappush(min_heap, (matrix[r][c +
1], r, c + 1))

        return result
```

3. Result:



Problem List
Run
Submit
Pre
Description | Accepted × | Editorial | Solutions | Submissions
All Submissions

Accepted 87 / 87 testcases passed

Subham Sharma submitted at Feb 27, 2025 15:08

```
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
```

```
        for i in range(min(k, n)):
            heapq.heappush(min_heap, (matrix[i][0], i, 0))

        count, result = 0, 0

        while count < k:
            result, r, c = heapq.heappop(min_heap)
            count += 1

            if c + 1 < n:
                heapq.heappush(min_heap, (matrix[r][c + 1], r, c + 1))

        return result
```

Accepted Runtime: 0 ms

Input

matrix =
[[1,5,9],[10,11,13],[12,13,15]]

k =
8

Problem 4: Search a 2D matrix II

Problem Statement:

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix.

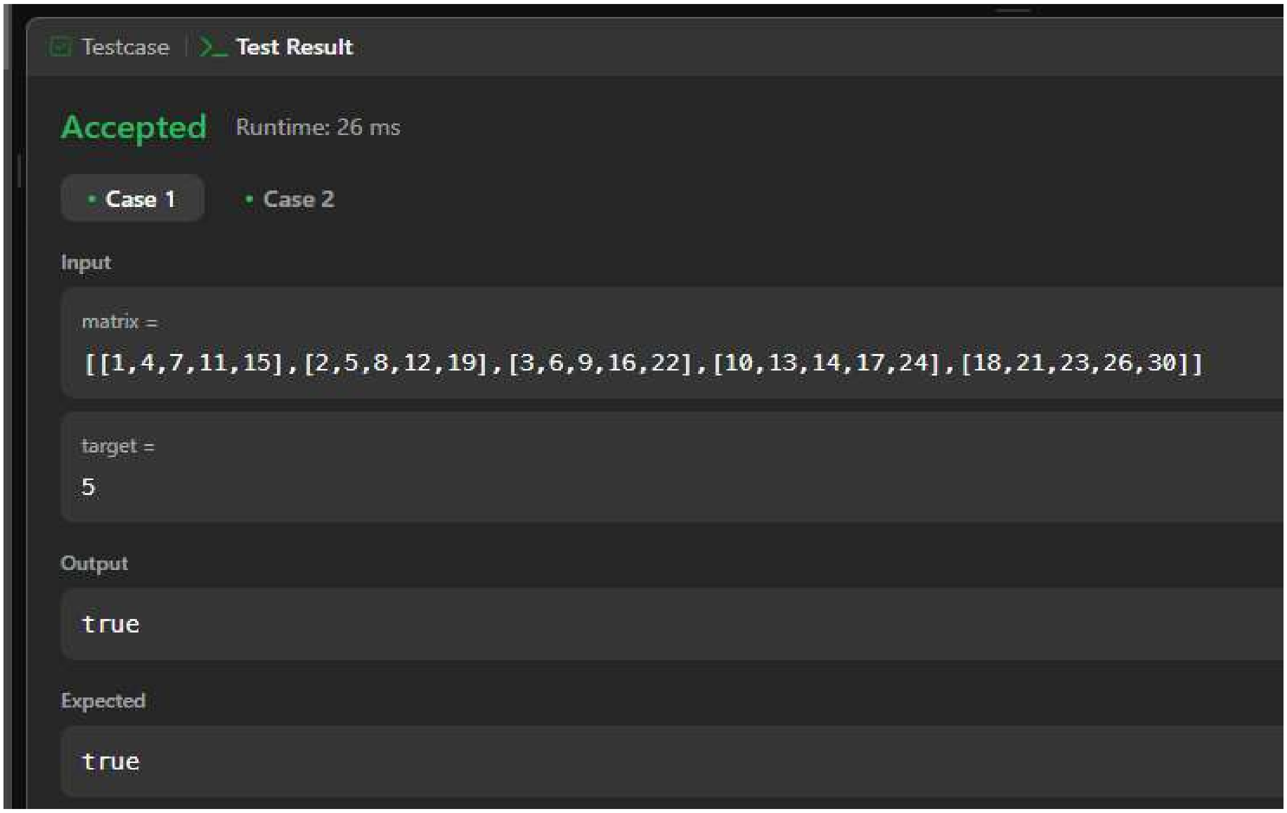
1. **Objective:** This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

2. Code:

```
class Solution(object):
    def searchMatrix(self, matrix, target):
        if not matrix or not matrix[0]:
            return False

        rows, cols = len(matrix), len(matrix[0])
        left, right = 0, rows * cols - 1

        while left <= right:
            mid = (left + right) // 2
            mid_value = matrix[mid // cols][mid % cols]

            if mid_value == target:
                return True
            elif mid_value < target:
                left = mid + 1
            else:
                right = mid - 1

        return False
```

3.Result:



Testcase | >_ Test Result

Accepted Runtime: 26 ms

• Case 1 • Case 2

Input

matrix =
```
[[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]]
```

target =
```
5
```

Output
```
true
```

Expected
```
true
```

Problem 5:Finnd Peak Elements

Problem Statement: A peak element is an element that is strictly greater than its neighbors.

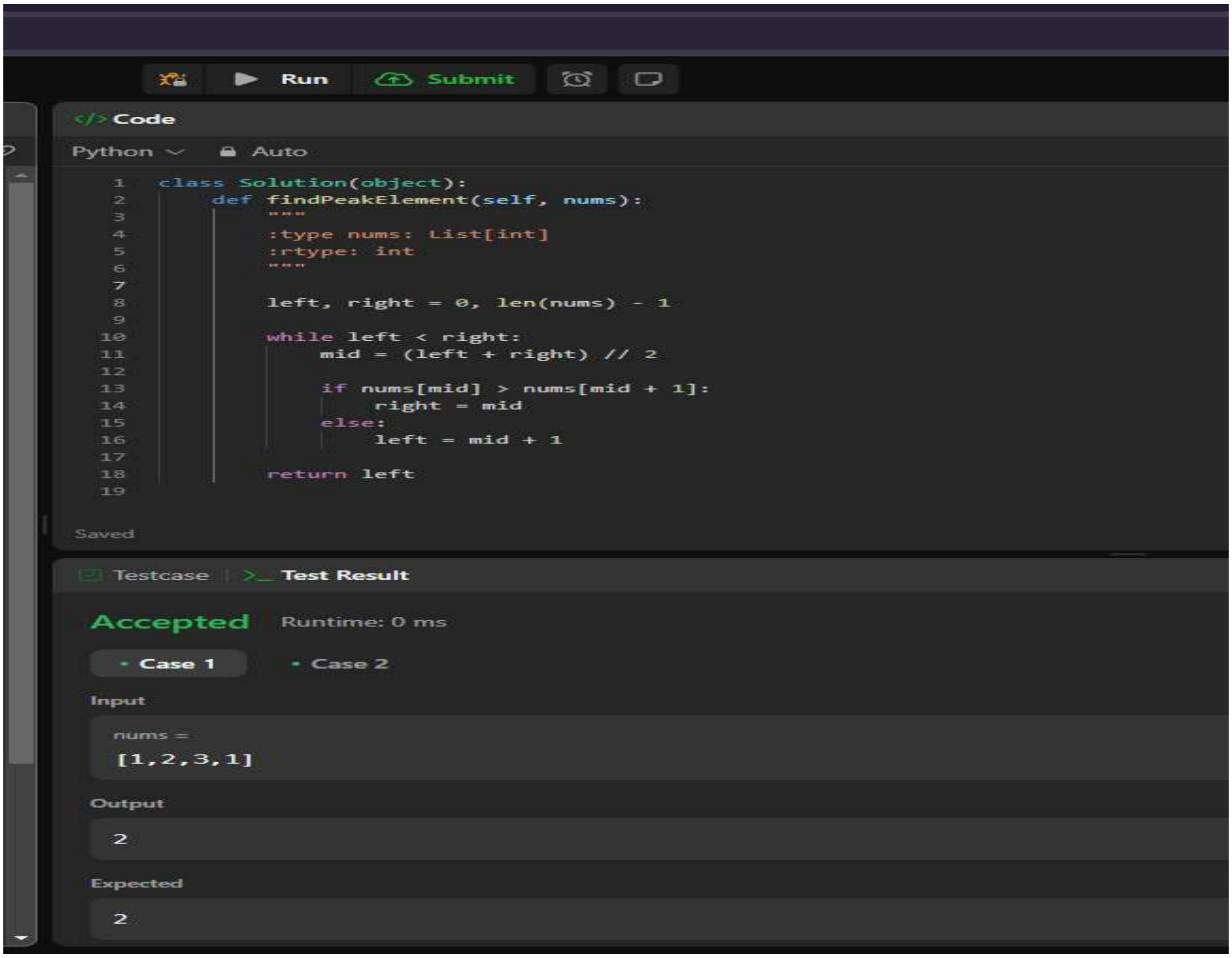
1. **Objective:** Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.
   You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.
   You must write an algorithm that runs in O(log n) time.

2. **Code:**

```
  class Solution(object):
def findPeakElement(self, nums):
   """
   :type nums: List[int]
   :rtype: int
   """

   left, right = 0, len(nums) – 1
```

```
while left < right:
          mid = (left + right) // 2

          if nums[mid] > nums[mid + 1]:
              right = mid
          else:
              left = mid + 1

      return left
```

3. Result:



```
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        left, right = 0, len(nums) - 1

        while left < right:
            mid = (left + right) // 2

            if nums[mid] > nums[mid + 1]:
                right = mid
            else:
                left = mid + 1

        return left
```

Accepted Runtime: 0 ms

Input

```
nums =
[1,2,3,1]
```

Output

```
2
```

Expected

```
2
```

Problem 6: Median of Two Sorted Arrays

Problem Statement: An array nums of length n is beautiful if:

nums is a permutation of the integers in the range [1, n].

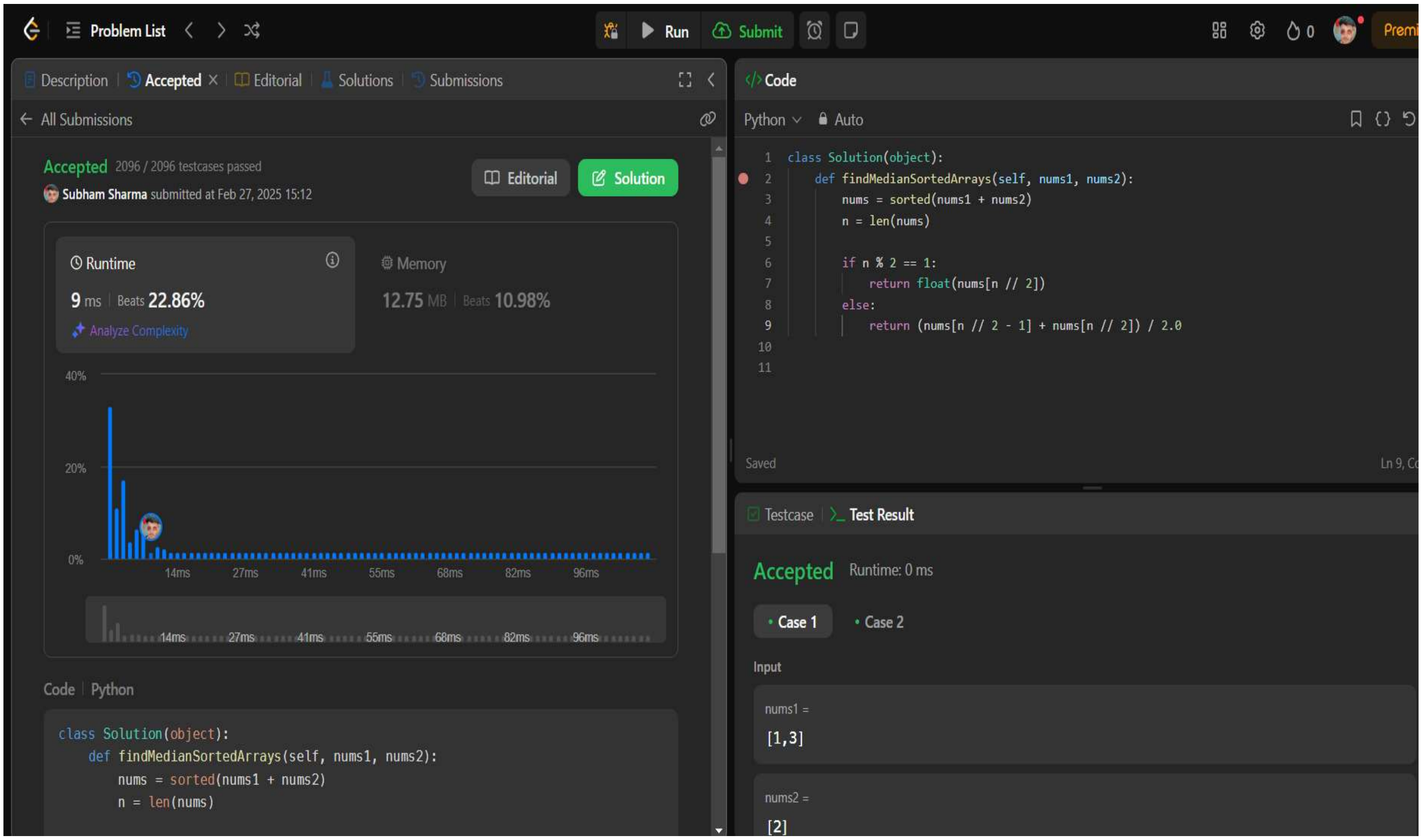
For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

1. **Objective:** Given the integer n, return any beautiful array nums of length n. There will be at least one valid answer for the given n.

2. **Code:**

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        nums = sorted(nums1 + nums2)
        n = len(nums)

        if n % 2 == 1:
            return float(nums[n // 2])
        else:
            return (nums[n // 2 - 1] + nums[n // 2]) / 2.0
```

3. Result:



Accepted 2096 / 2096 testcases passed

Subham Sharma submitted at Feb 27, 2025 15:12

Runtime

9 ms | Beats 22.86%

Memory

12.75 MB | Beats 10.98%

Code | Python

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        nums = sorted(nums1 + nums2)
        n = len(nums)
```

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        nums = sorted(nums1 + nums2)
        n = len(nums)

        if n % 2 == 1:
            return float(nums[n // 2])
        else:
            return (nums[n // 2 - 1] + nums[n // 2]) / 2.0
```

Accepted Runtime: 0 ms

Input

nums1 =

[1,3]

nums2 =

[2]

Learning Outcomes:

1. Understanding Merging and Sorting : Learn how to merge two sorted arrays and apply sorting techniques to maintain order efficiently.
2. Median Calculation : Gain insights into how to calculate the median for both even and odd-length lists by using index manipulation.
3. Time Complexity Awareness : Understand the impact of sorting (O(N log N)) and how to optimize solutions using binary search or two-pointer techniques.
4. Handling Edge Cases : Learn to handle edge cases like empty arrays, single-element arrays, and duplicate values while computing the median.
5. Application of Mathematical Logic : Improve problem-solving skills by using mathematical formulas for index calculations in ordered lists.