Experiment 5

Student Name: Sikander Singh Nanglu **UID: 22BET10031**
Branch: BE-IT **Section/Group: 22BET-IOT-701/A**
Semester: 6 **Date of Performance: 21/02/2025**
Subject Name:AP2 **Subject Code: 22ITP-351**

1. **Aim:** Implement the following problem:- Find Peak Element, Merge Intervals, Search in Rotated Sorted Array, Search a 2d matrix 2, Wiggle Sort 2, Kth smallest element in a sorted matrix, Median of Two Sorted Arrays.

2.Objective: To develop efficient algorithms for searching, merging, and sorting in complex data structures, improving problem-solving skills in array and matrix manipulation. The implementation focuses on optimizing time and space complexity while handling edge cases effectively.

3.Implementation/Code:

(A)Find Peak Element

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

(B)Merge Intervals

```
class Solution {
public:
vector<vector<int>> merge(vector<vector<int>>& intervals) {
if (intervals.empty()) return {};
sort(intervals.begin(), intervals.end());
```

```
vector<vector<int>> merged;
for (const auto& interval : intervals) {
if (merged.empty() || merged.back()[1] < interval[0]) {
merged.push_back(interval);
} else {
merged.back()[1] = max(merged.back()[1], interval[1]);
}
}
return merged;
}
};
```

(C)Search in rotated sorted array

```
class Solution {
public:
int search(vector<int>& nums, int target) {
int left = 0, right = nums.size() - 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;
}
};
```

(D)Search a 2D Matrix II

```
class Solution {
public:
bool searchMatrix(vector<vector<int>>& matrix, int target) {
int n=matrix.size();
int m =matrix[0].size();
for(int i=0;i<n;i++){
for(int j=0;j<m;j++){
if(matrix[i][j]==target)
return true;
}
}
return false;
}
};
```

(E)Wiggle Sort 2

```
class Solution {
public:
void wiggleSort(vector<int>& nums) {
int n = nums.size();
vector<int> sortedNums = nums;
sort(sortedNums.begin(), sortedNums.end());
int mid = (n - 1) / 2, end = n - 1;
for (int i = 0; i < n; i++) {
nums[i] = (i % 2 == 0) ? sortedNums[mid--] : sortedNums[end--];
}
}
};
```

(F)Kth smallest element in a sorted matrix.

```
class Solution {
public:
int kthSmallest(vector<vector<int>>& matrix, int k) {
priority_queue<int> maxHeap;
int n = matrix.size();
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
```

```
maxHeap.push(matrix[i][j]);
if (maxHeap.size() > k) {
maxHeap.pop();
}
}
}
return maxHeap.top();
}
};
```

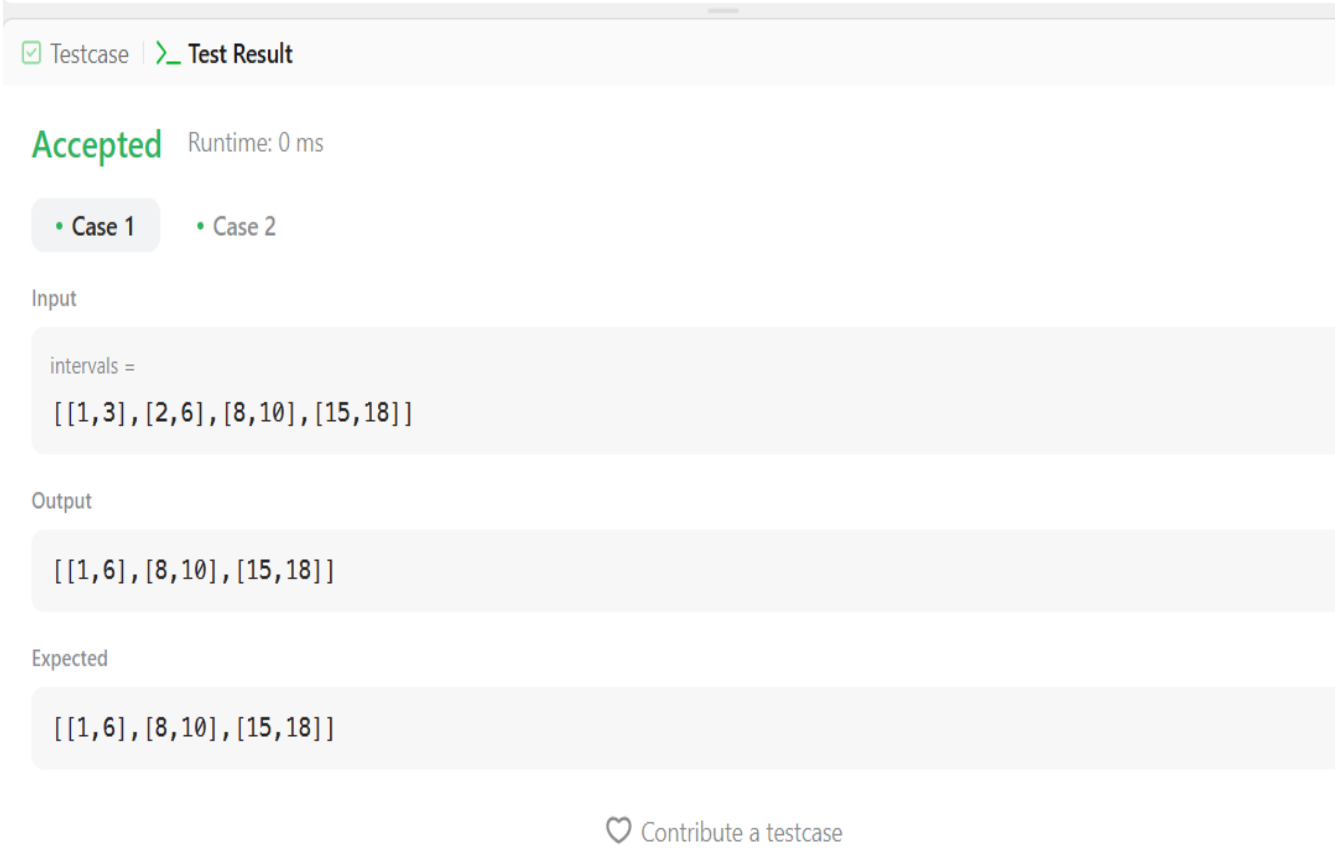
(G)Median of two sorted arrays

```
class Solution {
public:
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
int n1 = nums1.size(), n2 = nums2.size();
if (n1 > n2) return findMedianSortedArrays(nums2, nums1);
int low = 0, high = n1, midCount = (n1 + n2 + 1) / 2;
while (low <= high) {
int cut1 = (low + high) / 2;
int cut2 = midCount - cut1;
int left1 = (cut1 == 0) ? INT_MIN : nums1[cut1 - 1];
int left2 = (cut2 == 0) ? INT_MIN : nums2[cut2 - 1];
int right1 = (cut1 == n1) ? INT_MAX : nums1[cut1];
int right2 = (cut2 == n2) ? INT_MAX : nums2[cut2];
if (left1 <= right2 && left2 <= right1) {
if ((n1 + n2) % 2 == 0) {
return (max(left1, left2) + min(right1, right2)) / 2.0;
} else {
return max(left1, left2);
}
} else if (left1 > right2) {
high = cut1 - 1;
} else {
low = cut1 + 1;
}
}
return 0.0;
}
};
```

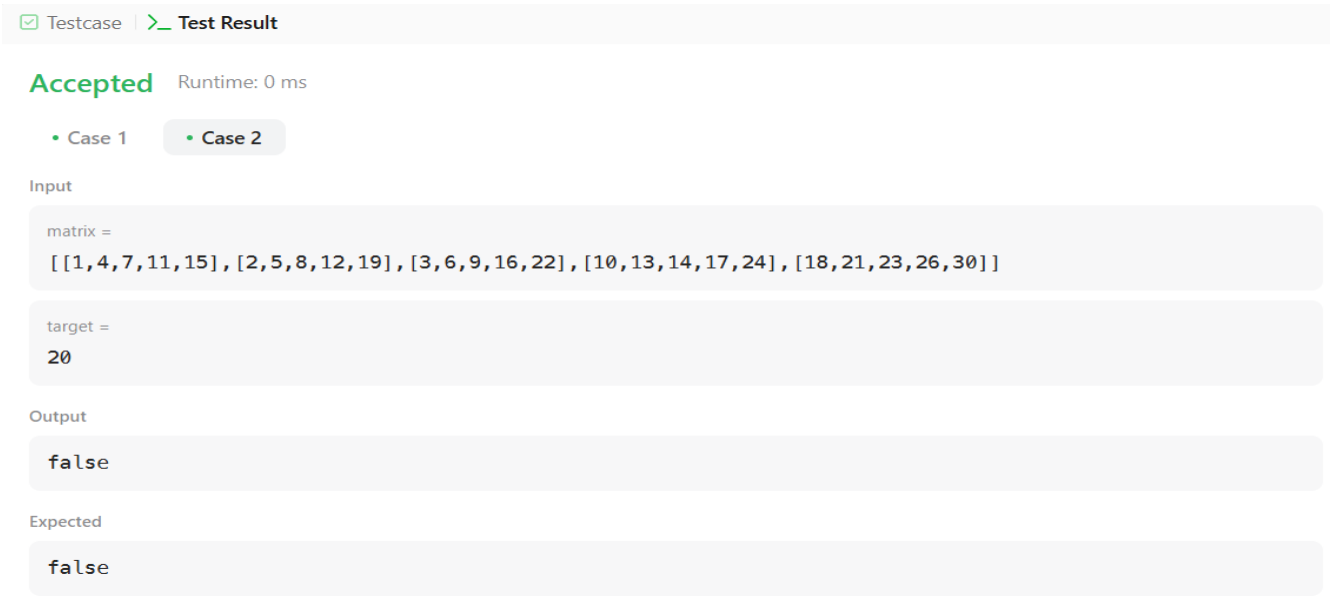
4.Output:

(A) Find Peak Element

(B)Merge Intervals

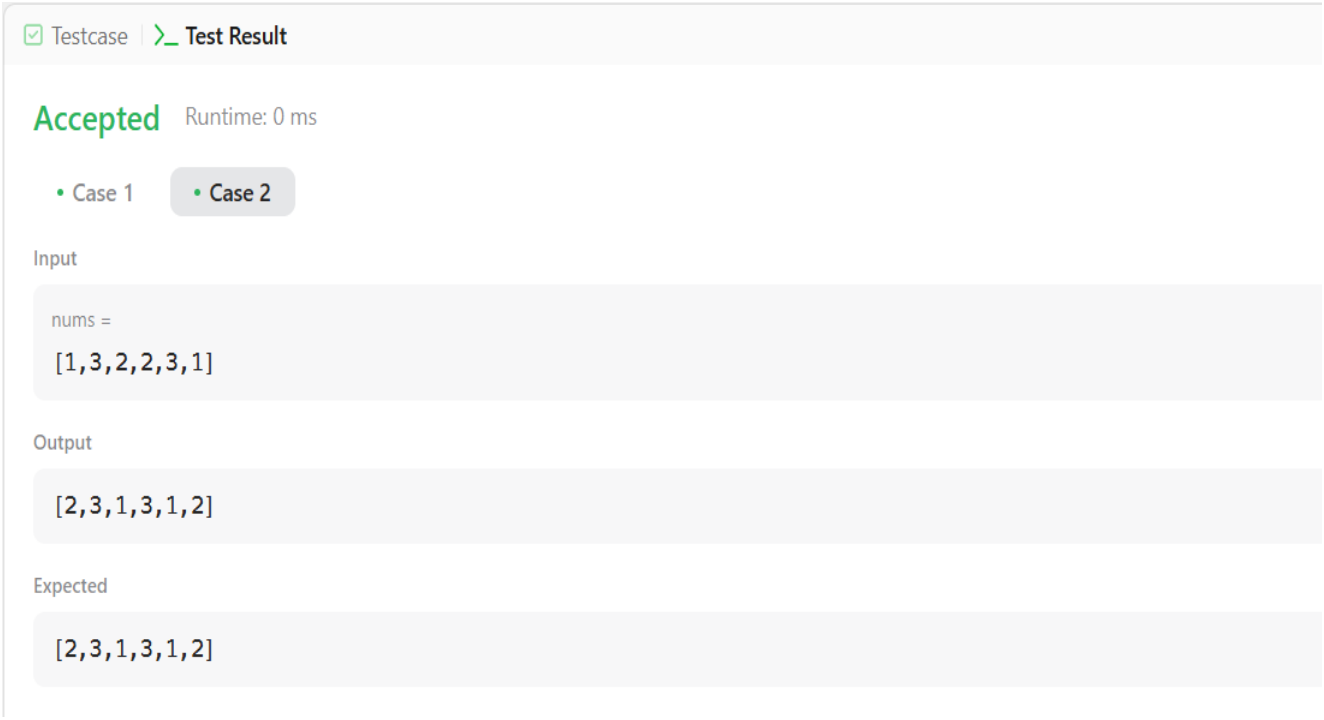


(C)Search a 2D matrix 2



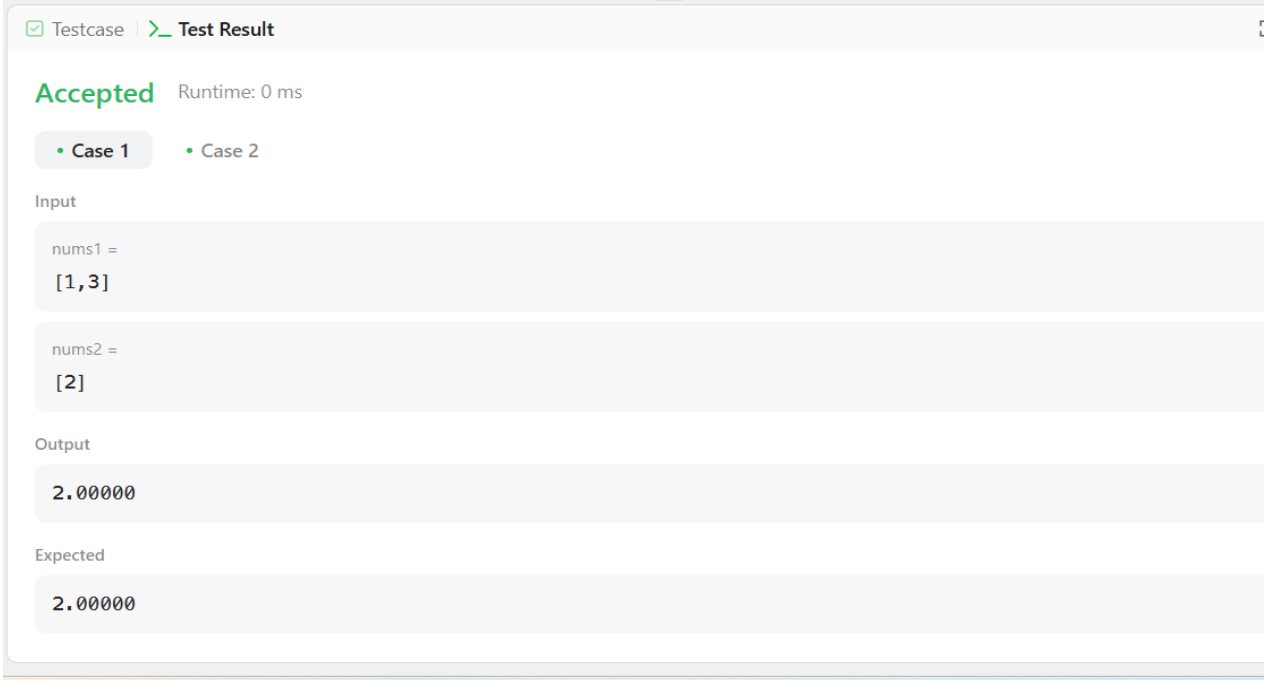
(D)Search in rotated sorted array

(E)Wiggle Sort 2



(F)Kth smallest element in a sorted matrix

(G)Median of two sorted arrays



5.Learning Outcomes:-

- Ability to analyze problems, evaluate information, and make logical decisions.
- Capability to identify, understand, and develop solutions to complex issues.
- Proficiency in expressing ideas clearly, both verbally and in writing
- Willingness to learn new skills and adjust to changing environments.
- Ability to work effectively with others in diverse environments.