EXPERIMENT-5

Student Name: Dushyant singh **UID:**22BET10060

Branch: BE -IT **Section/Group:**22BET_IOT-702(B)

Semester: 6th **Subject Code:** 22ITP-351

PROBLEM-1

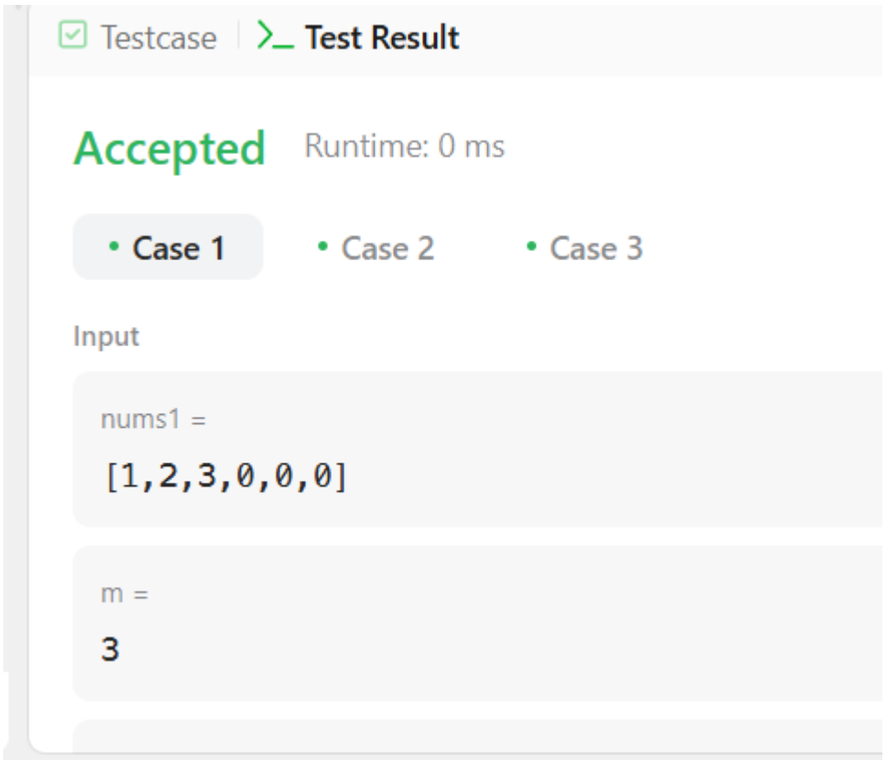
AIM:-

Merge Sorted Array

CODE:-

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m - 1;  // Last element in the valid part of nums1
        int j = n - 1;  // Last element in nums2
        int k = m + n - 1;  // Last position in nums1 (extended array)

        // Merge nums1 and nums2 from the end to the front
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k] = nums1[i];
                i--;
            } else {
                nums1[k] = nums2[j];
                j--;
            }
            k--;
        }

        // If there are remaining elements in nums2, copy them over to nums1
        while (j >= 0) {
            nums1[k] = nums2[j];
            j--;
            k--;
        }

    }
};
```

OUTPUT:-



PROBLEM-2

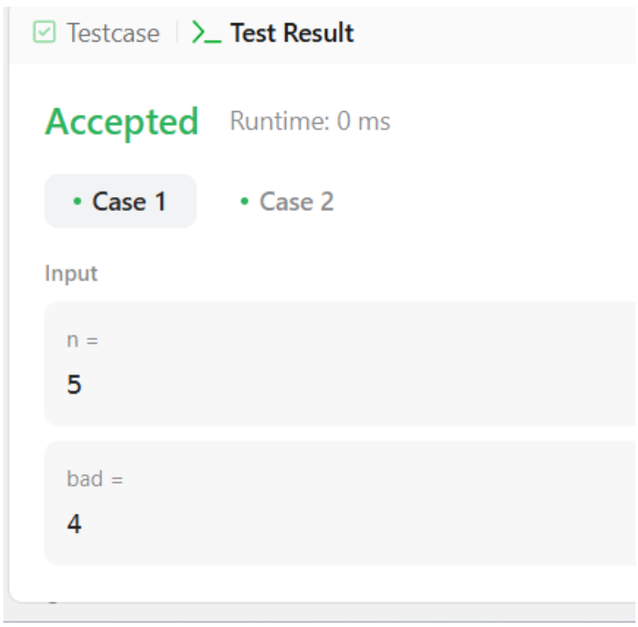
AIM:-

First Bad Version

CODE:-

```
class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;

        while (left < right) {
            int mid = left + (right - left) / 2;  // Calculate mid safely

            if (isBadVersion(mid)) {
                right = mid;  // If mid is bad, the first bad version is at mid or earlier
            } else {
                left = mid + 1;  // If mid is good, the first bad version is later
            }
        }

        return left;

    }
};
```

OUTPUT:



PROBLEM-3

AIM:-

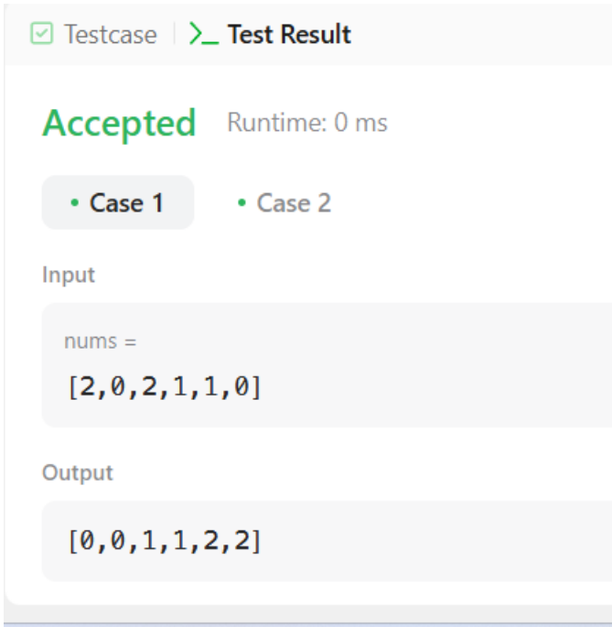
Sort Colors

CODE:-

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;

        while (mid <= high) {
            if (nums[mid] == 0) {
                // Swap 0 to the left part
                swap(nums[low], nums[mid]);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                // Move mid pointer forward
                mid++;
            } else {
                // Swap 2 to the right part
                swap(nums[mid], nums[high]);
                high--;
            }
        }

    }
```

};

OUTPUT:-



PROBLEM-4

AIM:-

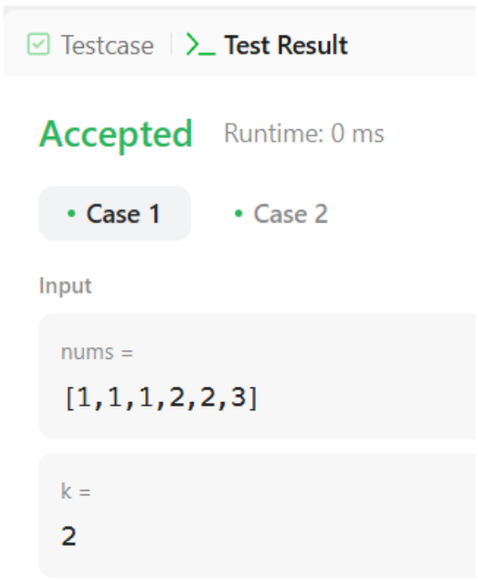
Top K frequent elements

CODE:-

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int, int> freqMap;
        for (int num : nums) {
            freqMap[num]++;
        }

        // Step 2: Use a min-heap to keep track of the k most frequent elements
        auto cmp = [](pair<int, int>& a, pair<int, int>& b) {
            return a.second > b.second; // Min-heap based on frequency
        };
        priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(cmp)> minHeap(cmp);

        // Step 3: Add elements to the heap
        for (auto& entry : freqMap) {
            minHeap.push(entry);
            if (minHeap.size() > k) {
                minHeap.pop(); // Remove the element with the smallest frequency
            }
        }
```

```
        // Step 4: Extract the k most frequent elements
        vector<int> result;
        while (!minHeap.empty()) {
            result.push_back(minHeap.top().first);
            minHeap.pop();
        }

        return result;

    }
};
```

OUTPUT:-



PROBLEM-5

AIM:-

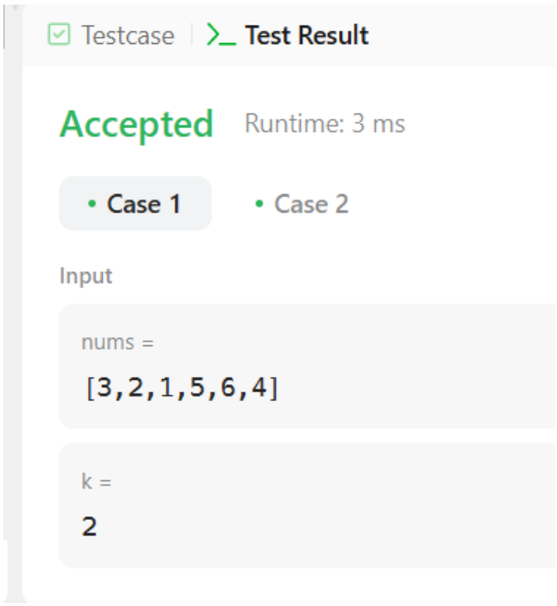
Kth Largest element in an array

CODE:-

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;

        // Process each element in the array
        for (int num : nums) {
            minHeap.push(num); // Push the current element
            if (minHeap.size() > k) {
                minHeap.pop(); // Remove the smallest element if the heap size exceeds k
            }
```

```
        }

        // The root of the min-heap will be the kth largest element
        return minHeap.top();

    }
};
```

OUTPUT:-



PROBLEM-6

AIM:-

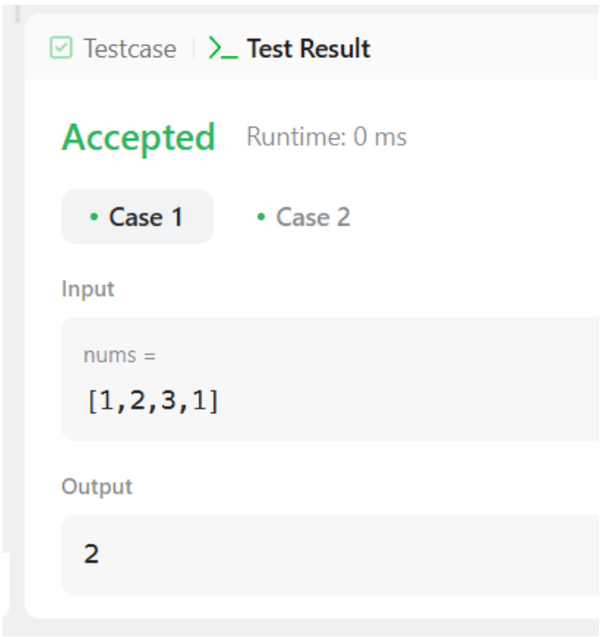
Find Peak Element

CODE:-

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;

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

            // Compare mid with its next element to decide the direction
            if (nums[mid] < nums[mid + 1]) {
                // Peak must be on the right half
                left = mid + 1;
            } else {
                // Peak must be on the left half or at mid itself
                right = mid;
            }
        }
```

```
        // At the end of the loop, left == right, which is the peak index
        return left;

    }
};
```

OUTPUT:-



PROBLEM-7

AIM:-

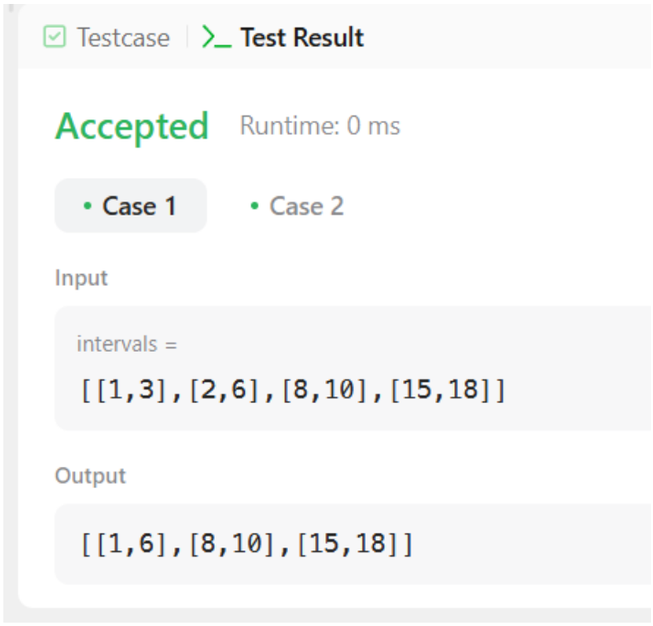
Merge Intervals

CODE:-

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

        // Step 2: Initialize a result list to store merged intervals
        std::vector<std::vector<int>> merged;

        // Step 3: Iterate through each interval
        for (const auto& interval : intervals) {
            // If merged is empty or there's no overlap, add the interval to the merged list
            if (merged.empty() || merged.back()[1] < interval[0]) {
                merged.push_back(interval);
            } else {
                // There's an overlap, so merge the current interval with the last merged one
                merged.back()[1] = std::max(merged.back()[1], interval[1]);
            }
```

```
        }

        // Step 4: Return the merged intervals
        return merged;

    }
};
```

OUTPUT:-



PROBLEM-8

AIM:-

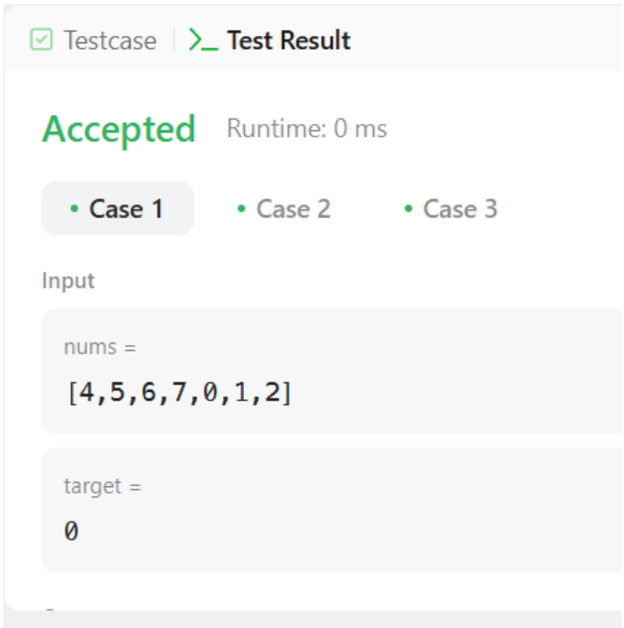
Search in Rotated Sorted Array

CODE:-

```
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 the target is found, return the index
            }
```

```
        // Check if the left side is sorted
        if (nums[left] <= nums[mid]) {
            if (nums[left] <= target && target < nums[mid]) {
                right = mid - 1;  // Target is in the left part
            } else {
                left = mid + 1;   // Target is in the right part
            }
        }
        // If the right side is sorted
        else {
            if (nums[mid] < target && target <= nums[right]) {
                left = mid + 1;   // Target is in the right part
            } else {
                right = mid - 1;  // Target is in the left part
            }
        }
    }

    return -1;

  }
};
```

OUTPUT:-



PROBLEM-9

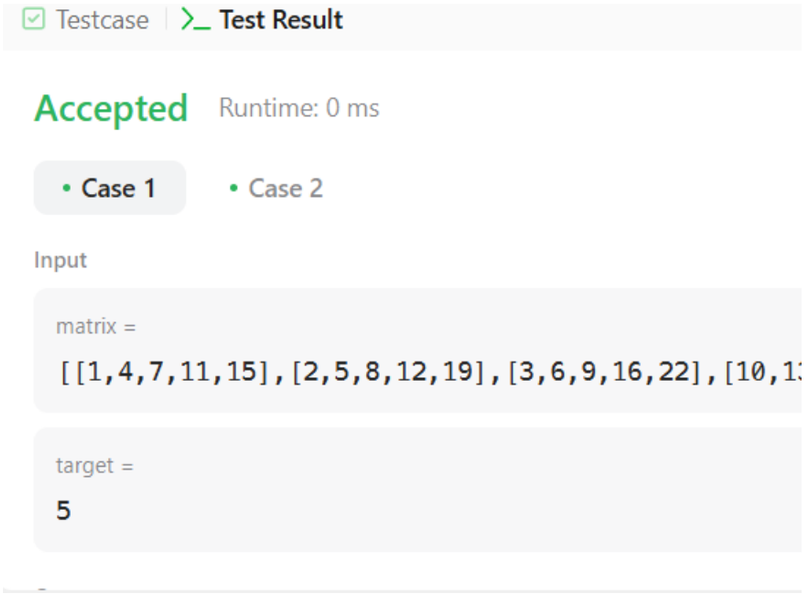
AIM:-

Search a 2D Matrix II

CODE:-

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size();
        int n = matrix[0].size();

        int row = 0;
        int col = n - 1;  // Start from the top-right corner

        while (row < m && col >= 0) {
            if (matrix[row][col] == target) {
                return true;  // Found the target
            }
            else if (matrix[row][col] > target) {
                col--;  // Move left
            }
            else {
                row++;  // Move down
            }
        }

        return false;

    }
};
```

OUTPUT:-


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,1
target =
5