Experiment 2

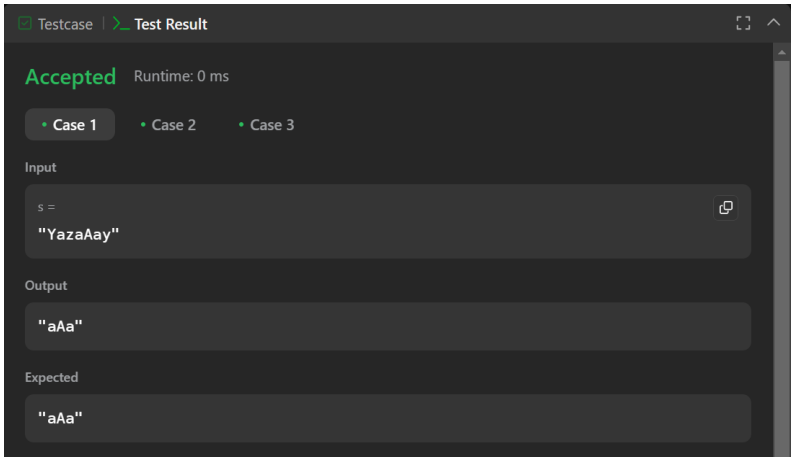
Student Name: Nishant Kumar
UID: 22BCS15009
Branch: CSE
Section/Group: 22BCS_FL_IOT-602 A
Semester: 6th
Date of Performance: 04/02/2025
Subject Name: Advanced Programming Lab - 2
Subject Code: 22CSP-351

Problem 1763. Longest Nice Substring

- **Implementation/Code:**

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        if (s.length() < 2) return "";

        unordered_set<char> charSet(s.begin(), s.end());

        for (int i = 0; i < s.length(); i++) {
            if (charSet.count(tolower(s[i])) && charSet.count(toupper(s[i]))) {
                continue;
            }
            string left = longestNiceSubstring(s.substr(0, i));
            string right = longestNiceSubstring(s.substr(i + 1));

            return left.length() >= right.length() ? left : right;
        }
        return s;
    }
};
```

- **Output:**

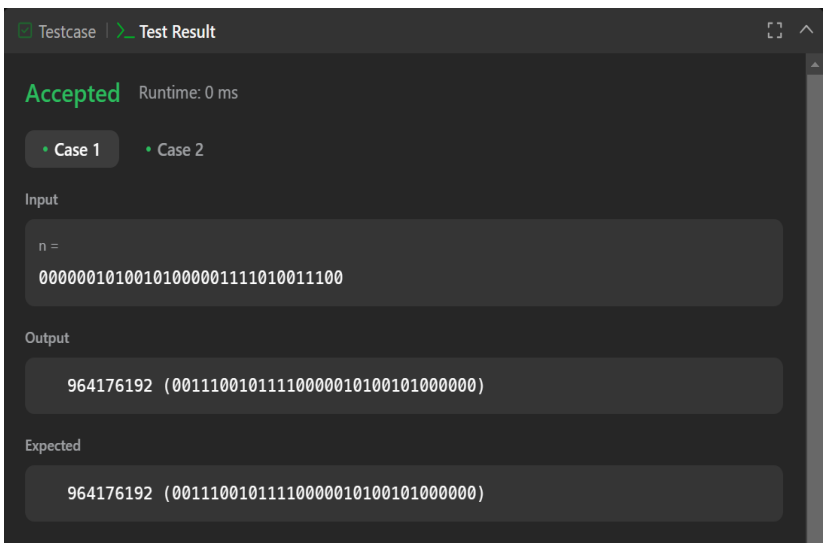


Problem 190. Reverse Bits

- **Implementation/Code:**

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```

- **Output:**

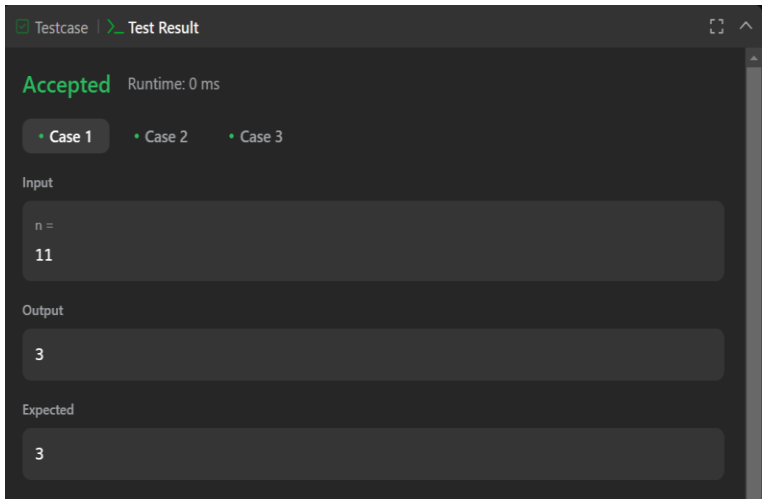


Problem 191. Number of 1 Bits

- **Implementation/Code:**

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

- **Output:**

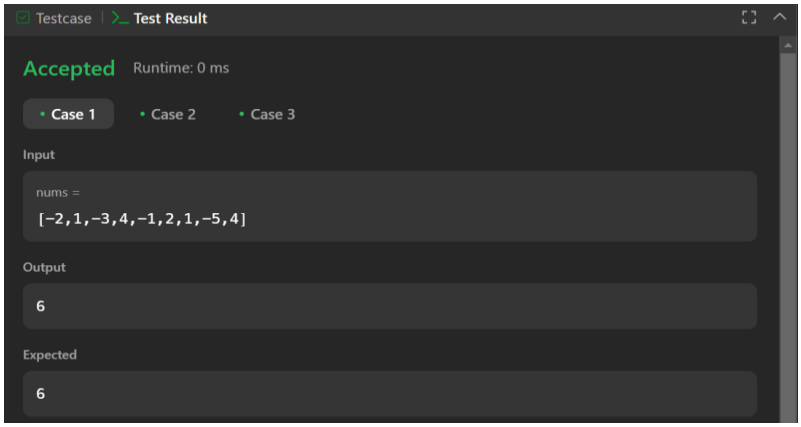


Problem 53. Maximum Subarray

- **Implementation/Code:**

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum = 0;
        int n = nums.size();
        int maximum = nums[0];
        for (int i = 0; i < n; i++) {
            sum += nums[i];
            maximum = max(maximum, sum);
            if (sum < 0) sum = 0;
        }
        return maximum;
    }
};
```

- **Output:**

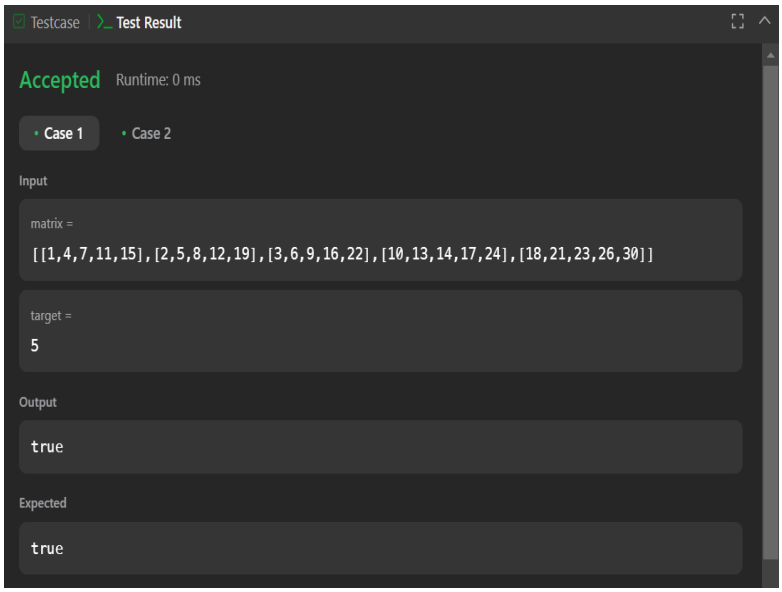


Problem 240. Search a 2D Matrix II

- **Implementation/Code:**

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int rows = matrix.size(), cols = matrix[0].size();
        int low = 0, high = rows * cols - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            int row = mid / cols, col = mid % cols;
            int midVal = matrix[row][col];

            if (midVal == target) return true;
            else if (midVal < target) low = mid + 1;
            else high = mid - 1;
        }
        return false;
    }
};
```

- **Output:**



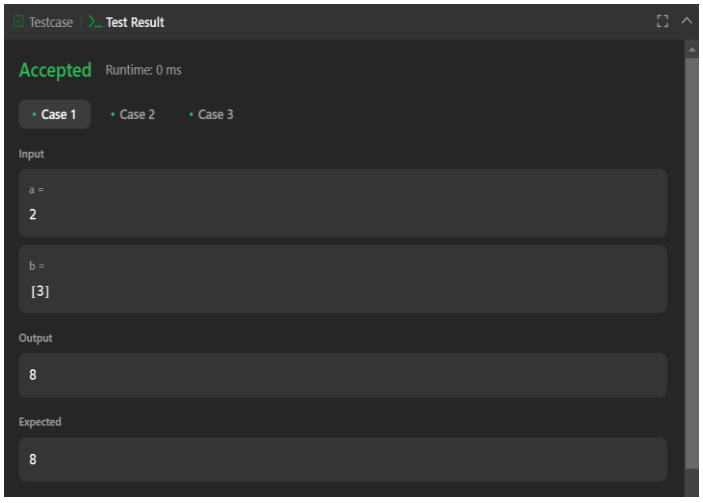
Problem 372. Super Pow

- **Implementation/Code:**

```
class Solution {
public:
    const int MOD = 1337;
```

```
    int powerMod(int a, int k) {
        a %= MOD;
        int res = 1;
        while (k > 0) {
            if (k % 2 == 1) {
                res = (res * a) % MOD;
            }
            a = (a * a) % MOD;
            k /= 2;
        }
        return res;
    }

    int superPow(int a, vector<int>& b) {
        int result = 1;
        for (int digit : b) {
            result = powerMod(result, 10) * powerMod(a, digit) % MOD;
        }
        return result;
    }
};
```

- **Output:**



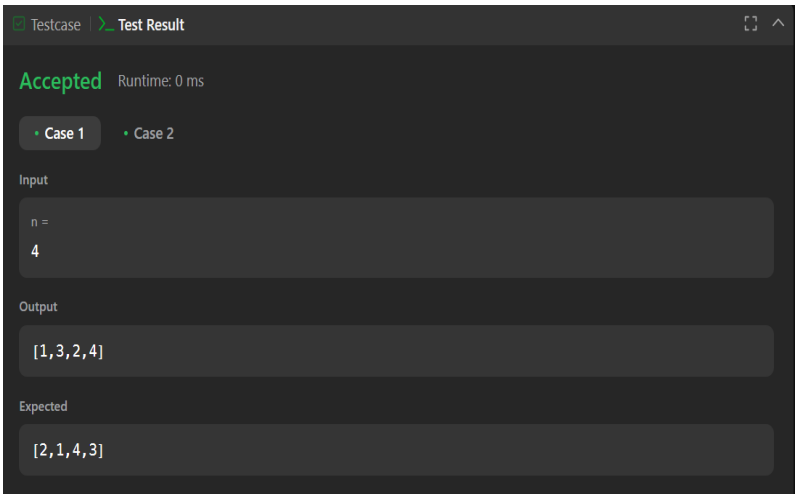
Problem 932. Beautiful Array

- **Implementation/Code:**

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1};

        vector<int> result;
```

```
        vector<int> oddPart = beautifulArray((n + 1) / 2);
        vector<int> evenPart = beautifulArray(n / 2);

        for (int num : oddPart) result.push_back(num * 2 - 1);
        for (int num : evenPart) result.push_back(num * 2);
        return result;
    }
};
```

- **Output:**



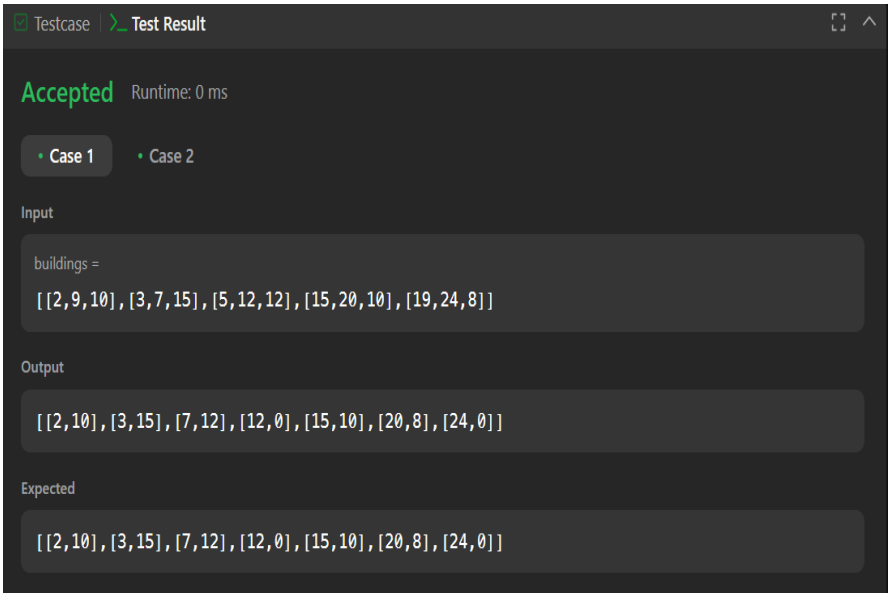
Problem 218. The Skyline Problem

- **Implementation/Code:**

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;

        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]);
            events.emplace_back(b[1], b[2]);
        }

        sort(events.begin(), events.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
            if (a.first != b.first) return a.first < b.first;
            return a.second < b.second;
        });

        vector<vector<int>> result;
        multiset<int> heights = {0};
        int prevMax = 0;
        for (auto& [x, h] : events) {
            if (h < 0) {
```

```
            heights.insert(-h);
        } else {
            heights.erase(heights.find(h));
        }

        int curMax = *heights.rbegin();
        if (curMax != prevMax) {
            result.push_back({x, curMax});
            prevMax = curMax;
        }
    }
    return result;
  }
};
```

- **Output:**



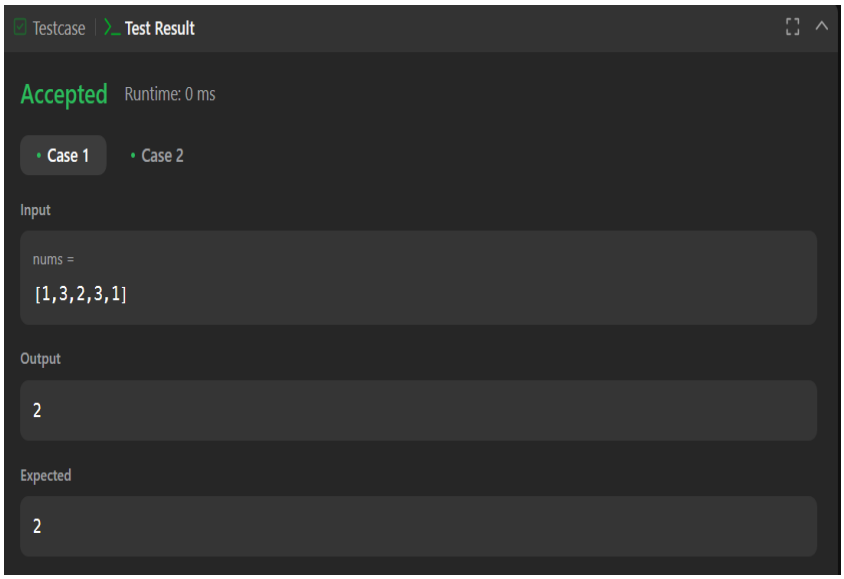
Problem 493. Reverse Pairs

- **Implementation/Code:**

```
class Solution {
public:
    int mergeSortAndCount(vector<int>& nums, int left, int right) {
        if (left >= right) return 0;

        int mid = left + (right - left) / 2;
        int count = mergeSortAndCount(nums, left, mid) + mergeSortAndCount(nums, mid +
1, right);

        int j = mid + 1;
        for (int i = left; i <= mid; i++) {
```

```
            while (j <= right && (long long)nums[i] > 2LL * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }
        vector<int> temp;
        int i = left, k = mid + 1;
        while (i <= mid && k <= right) {
            if (nums[i] <= nums[k]) {
                temp.push_back(nums[i++]);
            } else {
                temp.push_back(nums[k++]);
            }
        }
        while (i <= mid) temp.push_back(nums[i++]);
        while (k <= right) temp.push_back(nums[k++]);

        for (int i = left; i <= right; i++) {
            nums[i] = temp[i - left];
        }
        return count;
    }
    int reversePairs(vector<int>& nums) {
        return mergeSortAndCount(nums, 0, nums.size() - 1);
    }
};
```

- **Output:**



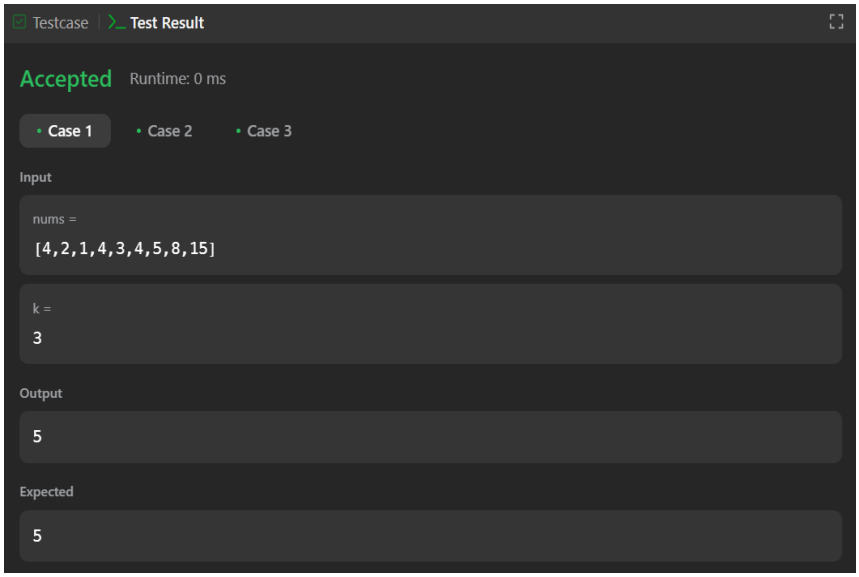
Problem 2407. Longest Increasing Subsequence II

- **Implementation/Code:**

```
class Solution {
public:
    class SegmentTree {
    public:
        vector<int> tree;
        int size;

        SegmentTree(int n) {
            size = n;
            tree.resize(4 * n, 0);
        }

        int query(int left, int right, int node, int start, int end) {
            if (start > right || end < left) return 0;
            if (start >= left && end <= right) return tree[node];

            int mid = (start + end) / 2;
            return max(query(left, right, 2 * node + 1, start, mid),
                       query(left, right, 2 * node + 2, mid + 1, end));
        }

        void update(int index, int value, int node, int start, int end) {
            if (start == end) {
                tree[node] = value;
                return;
            }

            int mid = (start + end) / 2;
            if (index <= mid)
                update(index, value, 2 * node + 1, start, mid);
            else
                update(index, value, 2 * node + 2, mid + 1, end);

            tree[node] = max(tree[2 * node + 1], tree[2 * node + 2]);
        }
    };

    int lengthOfLIS(vector<int>& nums, int k) {
        int maxNum = *max_element(nums.begin(), nums.end());
        SegmentTree segTree(maxNum + 1);

        int res = 0;
        for (int num : nums) {
            int bestPrev = segTree.query(max(0, num - k), num - 1, 0, 0, maxNum);
```

```
            int currentLIS = bestPrev + 1;
            segTree.update(num, currentLIS, 0, 0, maxNum);
            res = max(res, currentLIS);
        }
        return res;
    }
};
```

- **Output:**



Problem 88. Merge Sorted Arrays

- **Implementation/Code:**

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<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--];
        }
    }
};
```

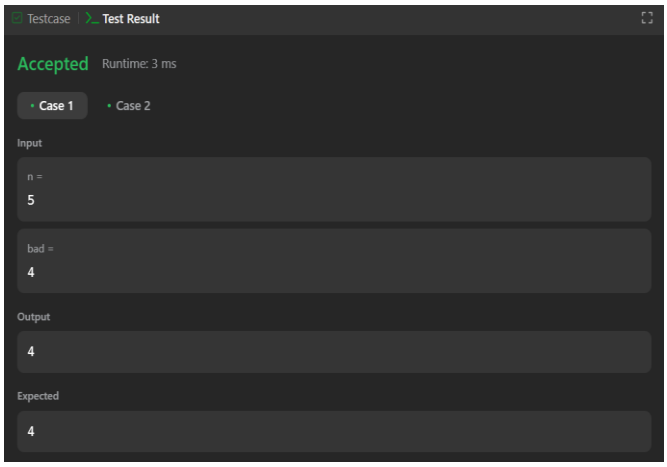
- **Output:**

Problem 278. First Bad Version

- **Implementation/Code:**

```
class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
};
```

- **Output:**

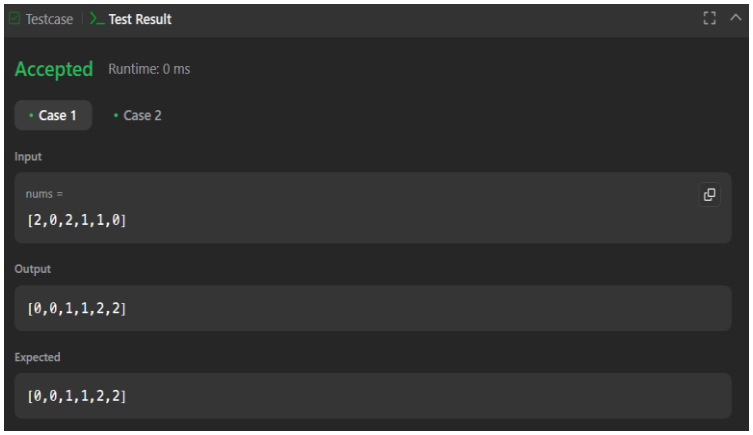


Problem 75. Sort Colours

- **Implementation/Code:**

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int count0 =0, count1=0, count2=0;
        for(int i=0;i<nums.size();i++)
        {
            if(nums[i]==0)
                count0++;
            else if(nums[i]==1)
                count1++;
            else
                count2++;
        }
        int i=0;
        while(count0>0)
        {
            nums[i++]=0;
            count0--;
        }
        while(count1>0)
        {
            nums[i++]=1;
            count1--;
        }
        while(count2>0)
        {
            nums[i++]=2;
            count2--;
        }
    }
};
```

- **Output:**

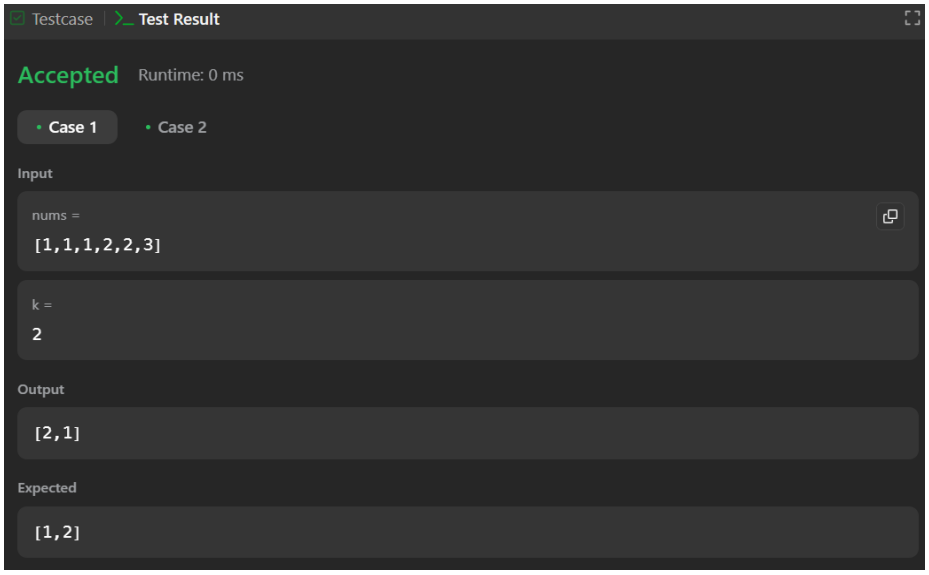


Problem 347. Top K Frequent Elements

- **Implementation/Code:**

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

        for (auto& [num, count] : freq) {
            minHeap.push({count, num});
            if (minHeap.size() > k) {
                minHeap.pop();
            }
        }
        vector<int> result;
        while (!minHeap.empty()) {
            result.push_back(minHeap.top().second);
            minHeap.pop();
        }
        return result;
    }
};
```

- **Output:**



Problem 215. Kth Largest Element in an Array

- **Implementation/Code:**

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

        for (int i = 0; i < k; i++) {
            minHeap.push(nums[i]);
        }

        for (int i = k; i < nums.size(); i++) {
            if (nums[i] > minHeap.top()) {
                minHeap.pop();
                minHeap.push(nums[i]);
            }
        }
        return minHeap.top();
    }
};
```

- **Output:**