**Ques 1. Longest Nice Subsrtring.**

**Code:**

class Solution {

public:

    string longestNiceSubstring(string s) {

        int n = s.length();

        if (n < 2) return "";

        for (int i = 0; i < n; ++i) {

            char ch = s[i];

            if (s.find(tolower(ch)) == string::npos || s.find(toupper(ch)) == string::npos) {

                string left = longestNiceSubstring(s.substr(0, i));

                string right = longestNiceSubstring(s.substr(i + 1));

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

            }

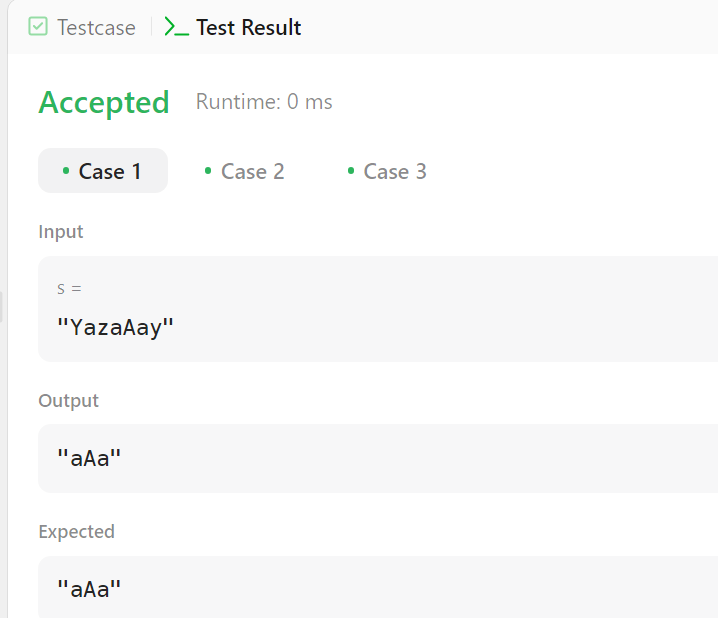
        }

        return s;

    }

};

**Output:**



**Ques 2. Reverse Bits.**

**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;

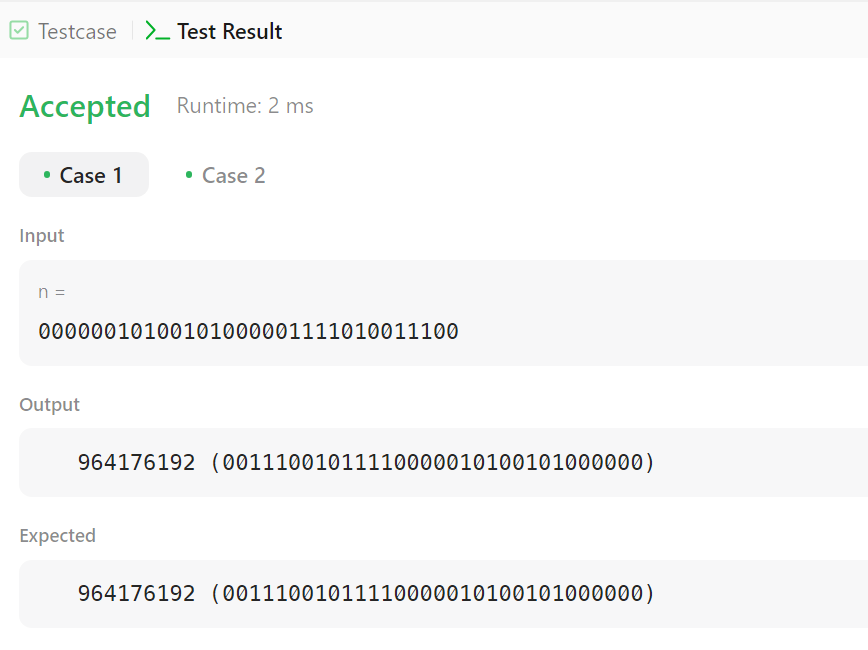
        }

        return result;

    }

};

**Output:**



**Ques 3. Number of 1 Bits.**

**Code:**

class Solution {

public:

int hammingWeight(int n) {

int count = 0;

while (n) {

count += (n & 1); // Add 1 if the last bit is set

n >>= 1; // Right shift n to check the next bit

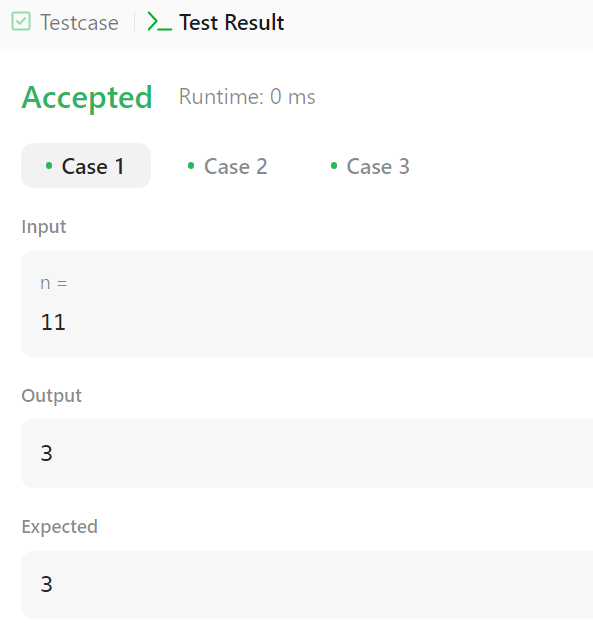
}

return count;

}

};

**Output:**



**Ques 4. Maximum Subarray.**

**Code:**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int maxSum = nums[0], currentSum = nums[0];

for (int i = 1; i < nums.size(); i++) {

currentSum = max(nums[i], currentSum + nums[i]);

maxSum = max(maxSum, currentSum);

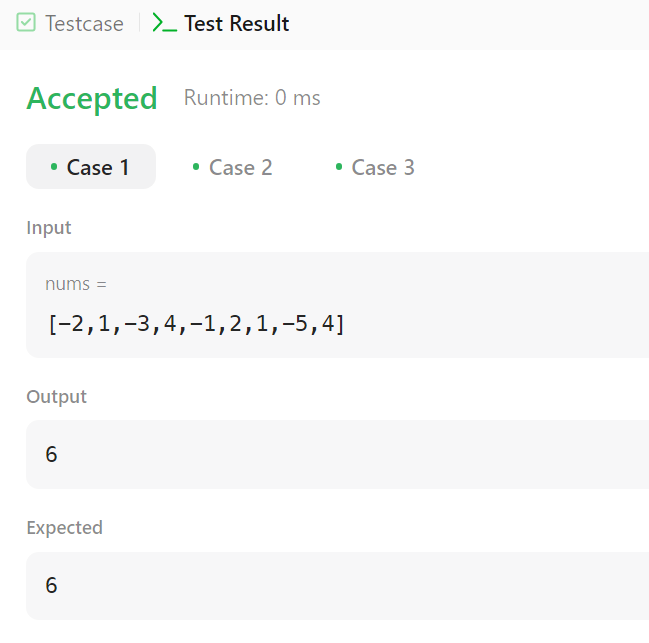
}

return maxSum;

}

};

**Output:**

****

**Ques 5. Search a 2-D matrix II.**

**Code:**

class Solution {

public:

bool searchMatrix(vector<vector<int>>& matrix, int target) {

int m = matrix.size(), n = matrix[0].size();

int row = 0, 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

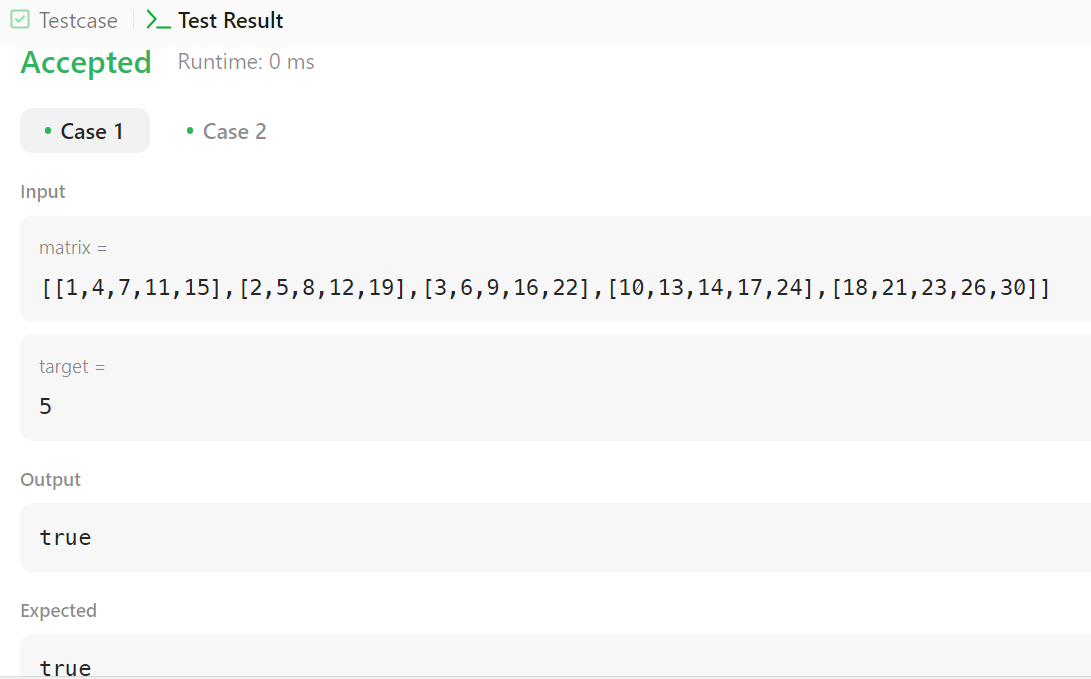
}

return false;

}

};

**Output:**



**Ques 6. Super Pow.**

**Code:**

class Solution {

public:

    const int MOD = 1337;

    int modPow(int a, int b) {

        int result = 1;

        a %= MOD;

        while (b > 0) { a = (a \* a) % MOD;

            b /= 2;}

        return result;

    }

    int superPow(int a, vector<int>& b) {

        a %= MOD;

        int result = 1;

        for (int digit : b) {

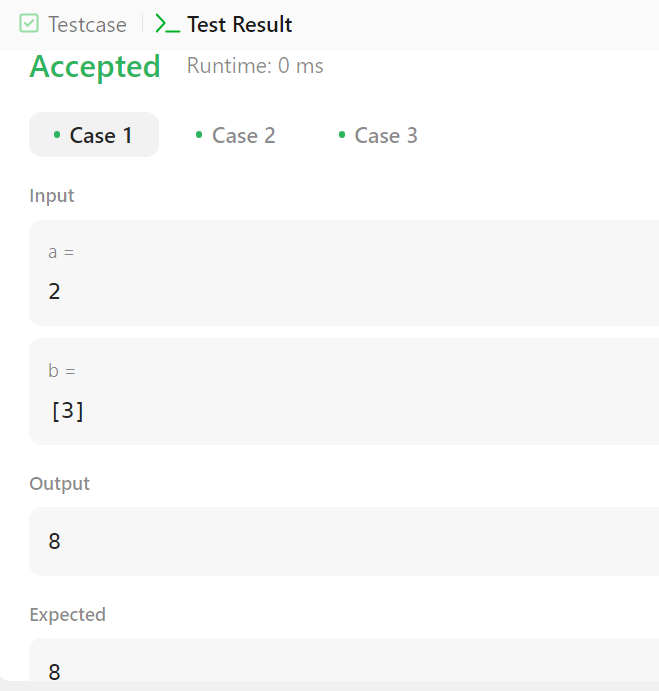
            result = (modPow(result, 10) \* modPow(a, digit)) % MOD; }

        return result;

    }

};

**Output:**



**Ques 7. Beautiful Array.**

**Code:**

class Solution {

public:

vector<int> beautifulArray(int n) {

if (n == 1) return {1};

vector<int> odd = beautifulArray((n + 1) / 2);

vector<int> even = beautifulArray(n / 2);

vector<int> result;

for (int num : odd) result.push\_back(num \* 2 - 1);

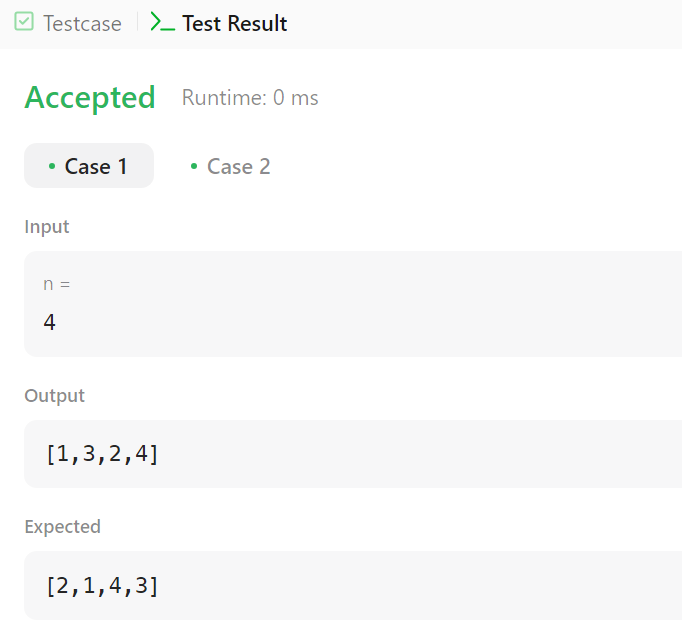
for (int num : even) result.push\_back(num \* 2);

return result;

}

};

**Output:**



**Ques 8. The SkyLine Problem.**

**Code:**

class Solution {

public:

vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

vector<vector<int>> result;

vector<pair<int, int>> events;

for (const auto& b : buildings) {

events.push\_back({b[0], -b[2]}); // Left edge, add height

events.push\_back({b[1], b[2]}); // Right edge, remove height

}

sort(events.begin(), events.end(), [](const pair<int, int>& a, const pair<int, int>& b) {

if (a.first == b.first) {

return a.second < b.second;

}

return a.first < b.first;

});

multiset<int> heights;

heights.insert(0);

int prevHeight = 0;

for (const auto& event : events) {

int x = event.first;

int h = event.second;

if (h < 0) {

heights.insert(-h);

} else {

heights.erase(heights.find(h));

}

int currentHeight = \*heights.rbegin();

// If the current height is different from the previous height, it's a key point

if (currentHeight != prevHeight) {

result.push\_back({x, currentHeight});

prevHeight = currentHeight;

}

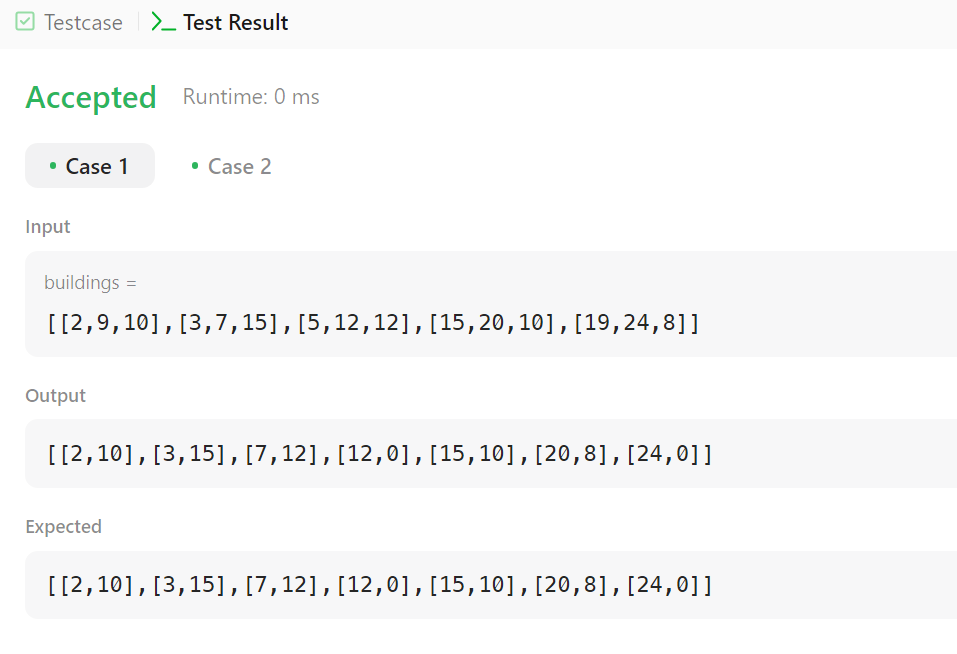
}

return result;

}

};

**Output:**

****

**Ques 9. Reverse Pairs.**

**Code:**

class Solution {

public:

int reversePairs(vector<int>& nums) {

return mergeSort(nums, 0, nums.size() - 1);

}

private:

int mergeSort(vector<int>& nums, int left, int right) {

if (left >= right) return 0;

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

int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

int j = mid + 1;

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

while (j <= right && nums[i] > 2 \* (long long)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 = 0; i < temp.size(); i++) {

nums[left + i] = temp[i];

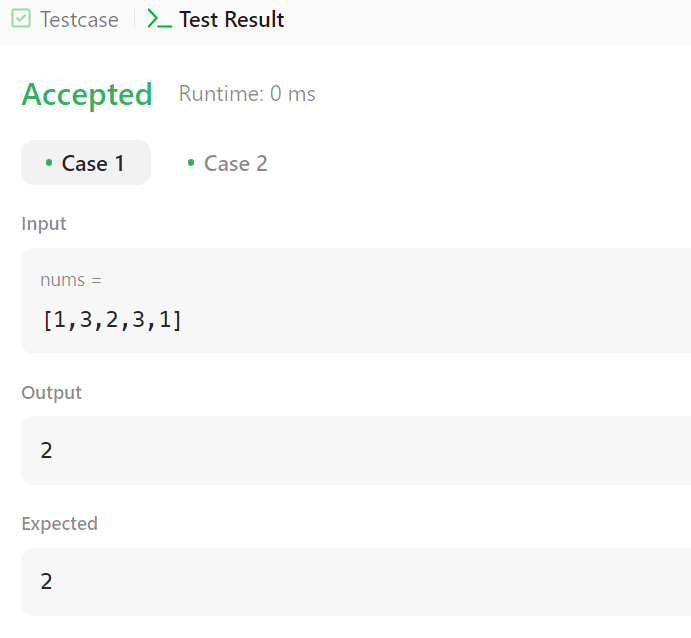
}

return count;

}

};

**Output:**



**Ques 10. Longest Increasing Subsequence II.**

**Code:**

class Solution {

public:

    int lengthOfLIS(vector<int>& nums, int k) {

        unordered\_map<int, int> dp

        int result = 0;

       int longest = 0;

            for (int i = num - k; i < num; ++i) {

                longest = max(longest, dp[i]);

            }

            dp[num] = longest + 1;

            result = max(result, dp[num]);

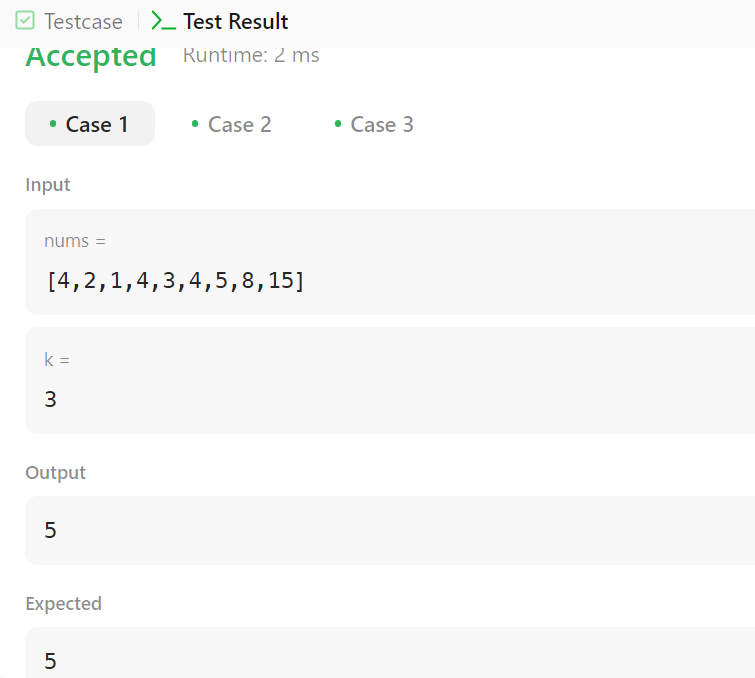
        }

        return result;

    }

};

**Output:**



**Ques 11. Merge Sorted Array.**

**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--];

        }

        for (int i = 0; i < m + n; i++) {

            cout << nums1[i] << " ";

        }

    }

};

**Output:**

****

**Ques 12. 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;

if (isBadVersion(mid)) {

right = mid;

} else {

left = mid + 1;

}

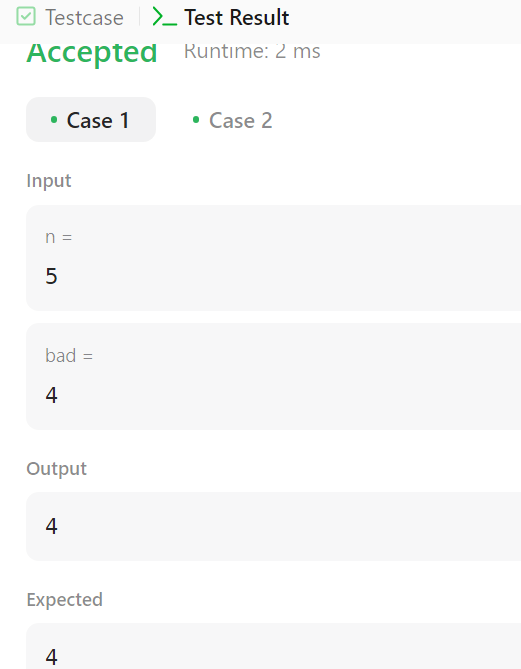
}

return left;

}

};

**Output:**

****

**Ques 13. 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(nums[low], nums[mid]);

low++;

mid++;

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

mid++;

} else {

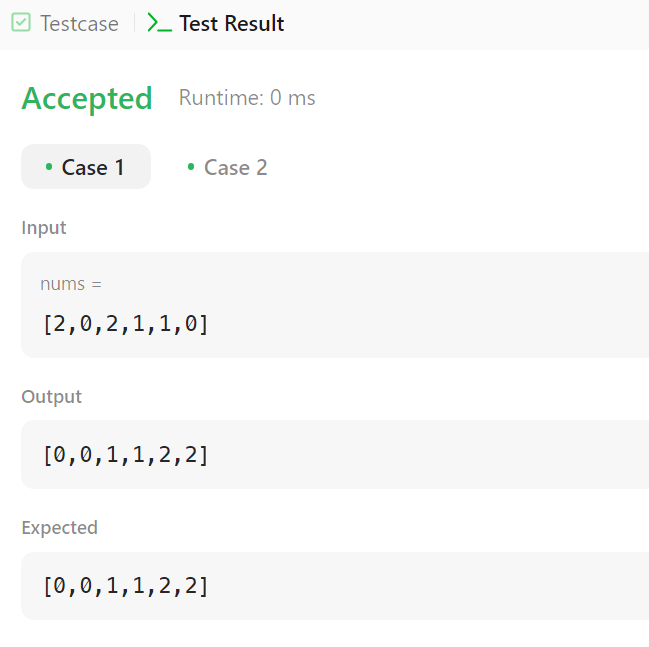
swap(nums[mid], nums[high]);

high--; }

}}

};

**Output:**

****

**Ques 14. 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]++;

}

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> minHeap;

for (auto& entry : freqMap) {

minHeap.push({entry.second, entry.first});

if (minHeap.size() > k) {

minHeap.pop();

}

}

vector<int> result;

while (!minHeap.empty()) {

result.push\_back(minHeap.top().second);

minHeap.pop();

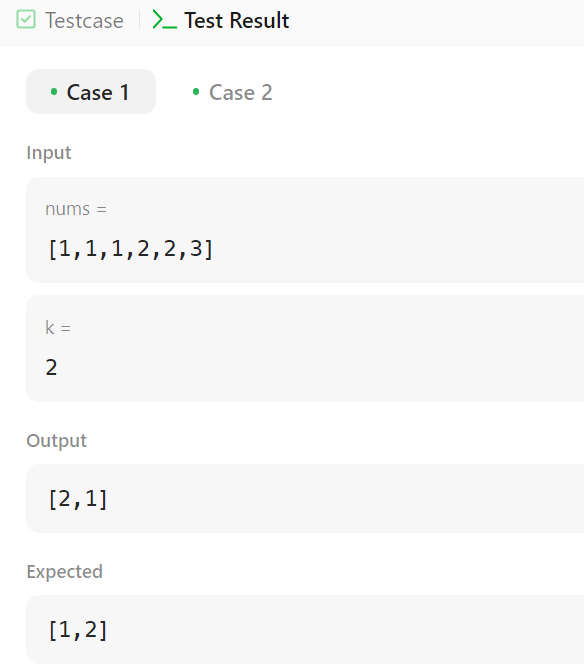
}

return result;

}

};

**Output:**

****

**Ques 15. Kth Largest Element in an array.**

**Code:**

class Solution {

public:

    int findKthLargest(vector<int>& nums, int k) {

        priority\_queue<int, vector<int>, greater<int>> minHeap;

        for (int num : nums) {

            minHeap.push(num);

            if (minHeap.size() > k) {

                minHeap.pop();

            }

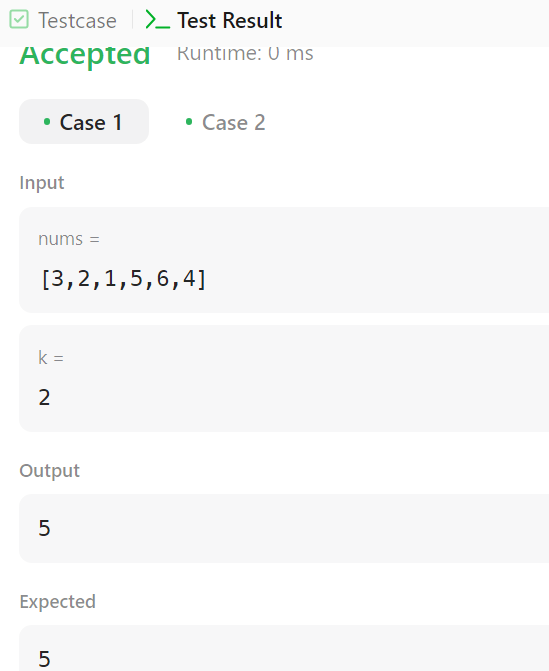
        }

        return minHeap.top();

    }

};

**Output:**

****