**Name : Prince Sharma**

**UID : 22BCS14846**

**SEC : IOT-605(b)**

Ques no. 1 :- Longest Nice Substring(1763).

Solution:

class Solution {

public:

string longestNiceSubstring(string s) {

string output = "";

int count = 0;

for(int i = 0;i<s.length();i++){

int smallMask=0;

int largeMask = 0;

char ch = s[i];

int chint = 0;

if(ch>=65 && ch<=90){

chint = ch-'A';

largeMask = 1<<chint;

}

else{

chint = ch-'a';

smallMask = 1<<chint;

}

for(int j = i+1;j<s.length();j++){

ch = s[j];

if(ch>=65 && ch<=90){

chint = ch-'A';

largeMask |= 1<<chint;

}

else{

chint = ch-'a';

smallMask |= 1<<chint;

}

//checking for nice

if((smallMask^largeMask) == 0){

if(count<j-i+1){

count = j-i+1;

string temp(s.begin()+i,s.begin()+j+1);

output = temp;

}

}

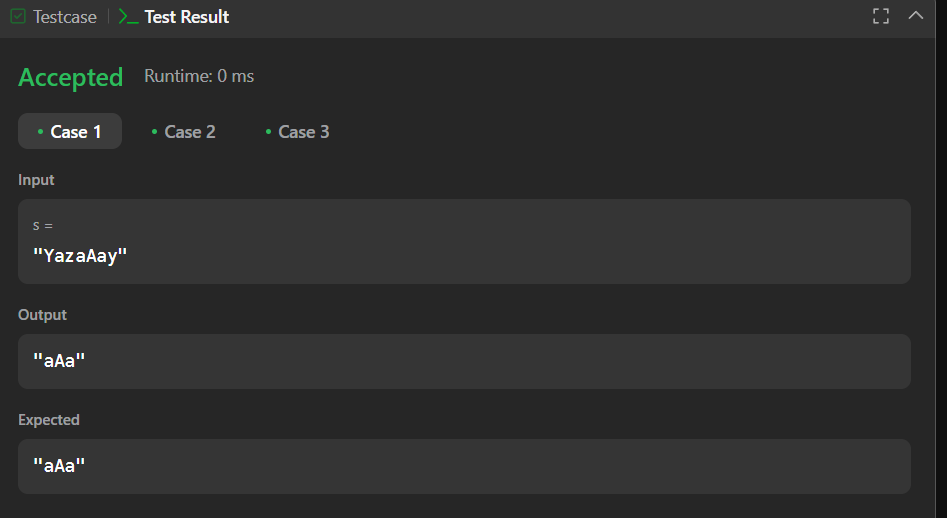
}

}

return output;

}

};



Ques no. 2 : Reverse Bits (190).

Solution :

class Solution {

public:

uint32\_t reverseBits(uint32\_t n) {

uint32\_t result = 0;

for (int i = 0; i < 32; i++) {

int bit = n & 1; // Extract the least significant bit

result = (result << 1) | bit; // Append the bit to the result

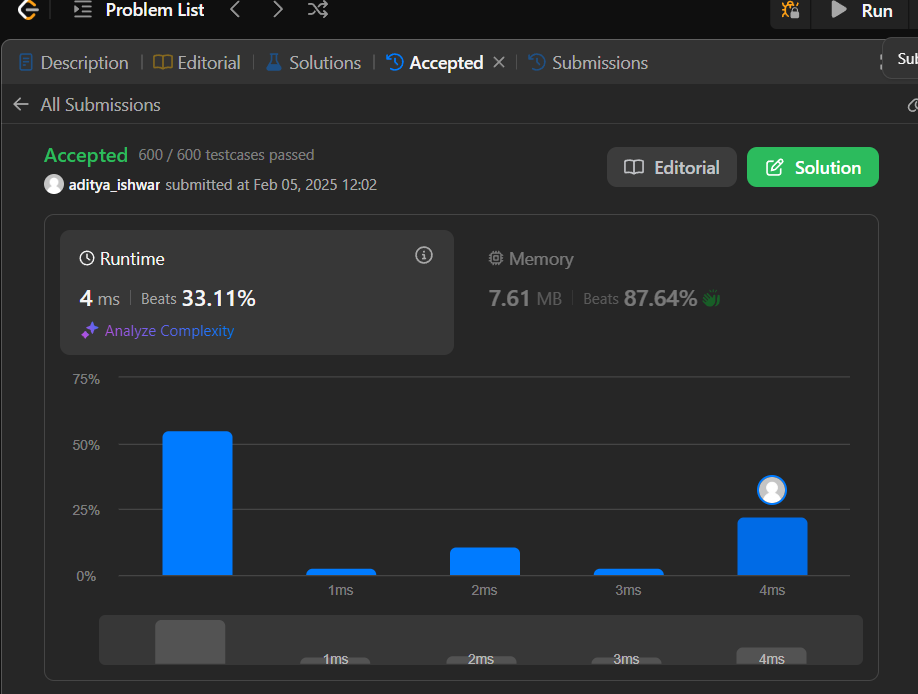
n = n >> 1; // Right-shift n to process the next bit

}

return result;

}

};



Ques no. 3 : Number of 1 Bits (191).

Solution :

class Solution {

public:

int hammingWeight(uint32\_t n) {

int res = 0;

for (int i = 0; i < 32; i++) {

if ((n >> i) & 1) {

res += 1;

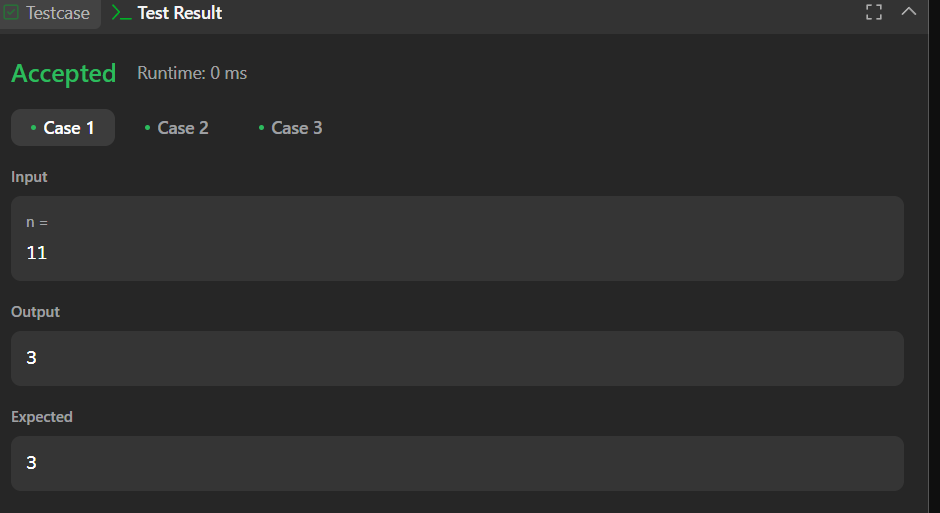
}

}

return res;

}

};



Ques no. 4: Maximum Subarray (53).

Solution:

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int res = nums[0];

int total = 0;

for (int n : nums) {

if (total < 0) {

total = 0;

}

total += n;

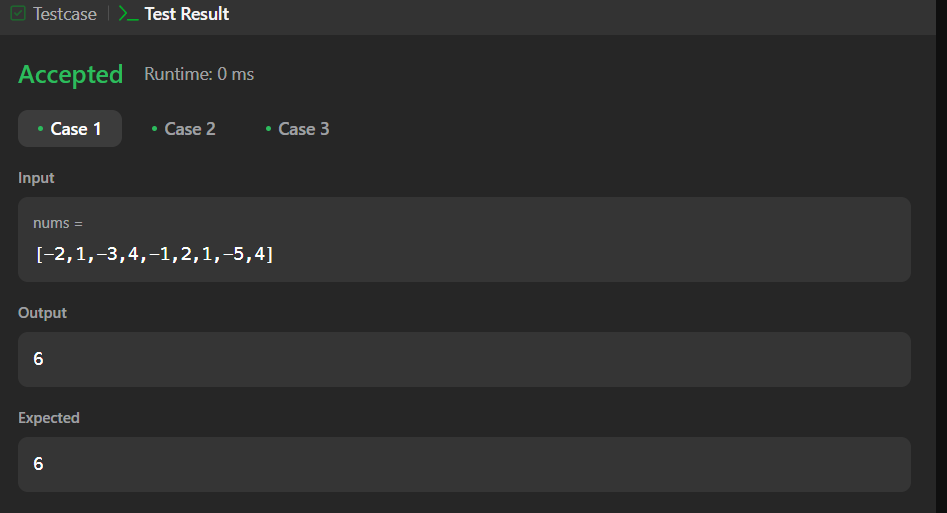
res = max(res, total);

}

return res;

}

};



Ques no. 5 : Search a 2D Matrix II (240).

Solution:

class Solution {

public:

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

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

int row = 0, col = m - 1;

while (row < n && col >= 0) {

if (matrix[row][col] == target) return true;

else if (matrix[row][col] < target) row++;

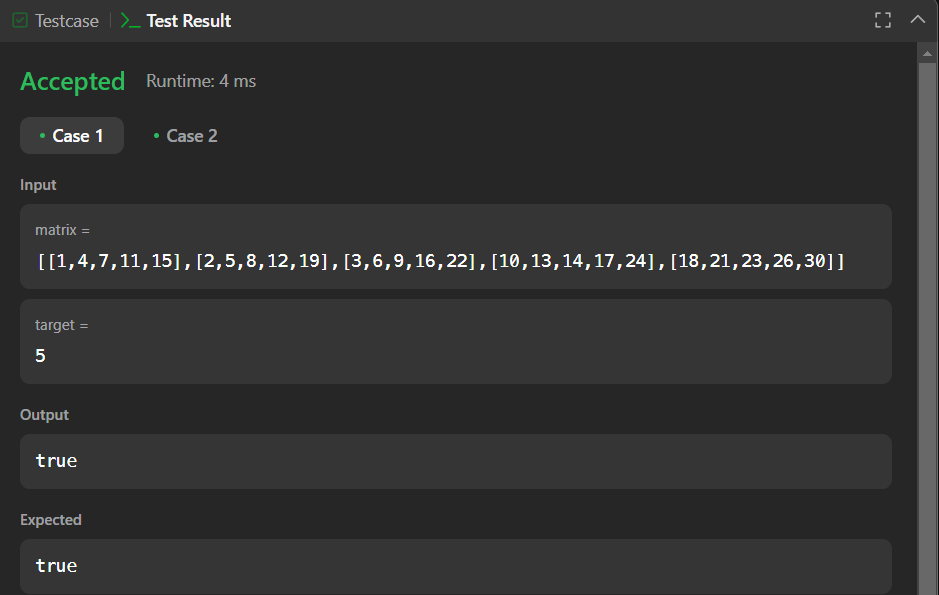
else col--;

}

return false;

}

};



Ques no. 6: Super Pow (372).

Solution :

class Solution {

const int base = 1337;

int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10

{

a %= base;

int result = 1;

for (int i = 0; i < k; ++i)

result = (result \* a) % base;

return result;

}

public:

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

if (b.empty()) return 1;

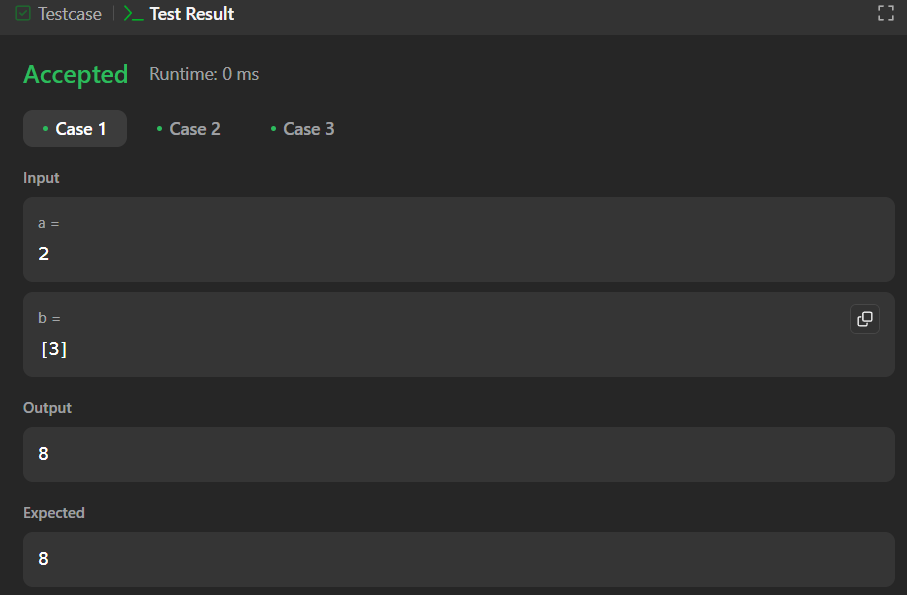
int last\_digit = b.back();

b.pop\_back();

return powmod(superPow(a, b), 10) \* powmod(a, last\_digit) % base;

}

};



Ques no. 7 : Beautiful Array(932).

Solution :

class Solution {

public:

static bool comp(const int &a, const int &b){

int mask = 1;

while(true)

if((a&mask) == (b&mask)) mask = mask<<1;

else return (a&mask) > (b&mask);

}

vector<int> beautifulArray(int n) {

vector<int> answer;

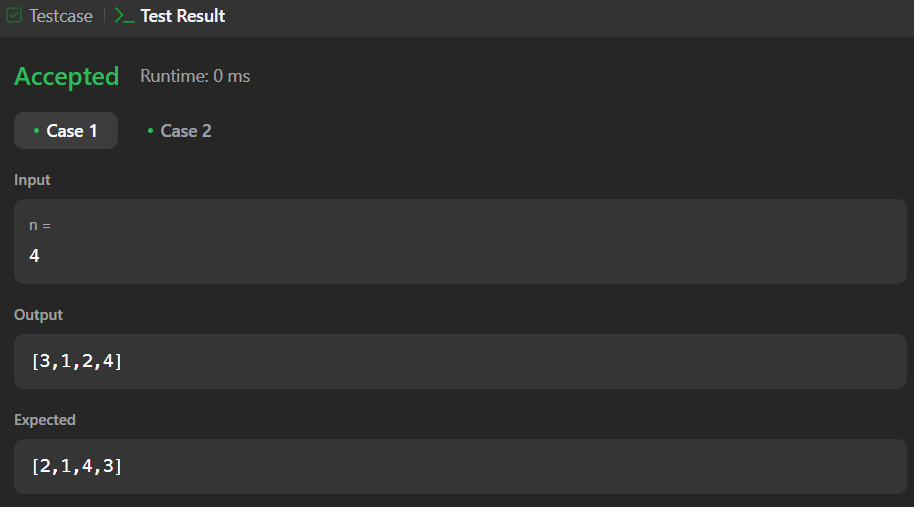
while(n) answer.push\_back(n--);

sort(answer.begin(), answer.end(), comp);

return answer;

}

};



Ques no. 8 : The Skyline Problem(218).

Solution :

class Solution {

public:

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

int edge\_idx = 0;

vector<pair<int, int>> edges;

priority\_queue<pair<int, int>> pq;

vector<vector<int>> skyline;

for (int i = 0; i < buildings.size(); ++i) {

const auto &b = buildings[i];

edges.emplace\_back(b[0], i);

edges.emplace\_back(b[1], i);

}

std::sort(edges.begin(), edges.end());

while (edge\_idx < edges.size()) {

int curr\_height;

const auto &[curr\_x, \_] = edges[edge\_idx];

while (edge\_idx < edges.size() &&

curr\_x == edges[edge\_idx].first) {

const auto &[\_, building\_idx] = edges[edge\_idx];

const auto &b = buildings[building\_idx];

if (b[0] == curr\_x)

pq.emplace(b[2], b[1]);

++edge\_idx;

}

while (!pq.empty() && pq.top().second <= curr\_x)

pq.pop();

curr\_height = pq.empty() ? 0 : pq.top().first;

if (skyline.empty() || skyline.back()[1] != curr\_height)

skyline.push\_back({curr\_x, curr\_height});

}

return skyline;

}

};



Ques no. 9: Reverse Pairs(493).

Solution :

class Solution {

public:

int reversePairs(vector<int>& nums) {

int n = nums.size();

long long reversePairsCount = 0;

for(int i=0; i<n-1; i++){

for(int j=i+1; j<n; j++){

if(nums[i] > 2\*(long long)nums[j]){

reversePairsCount++;

}

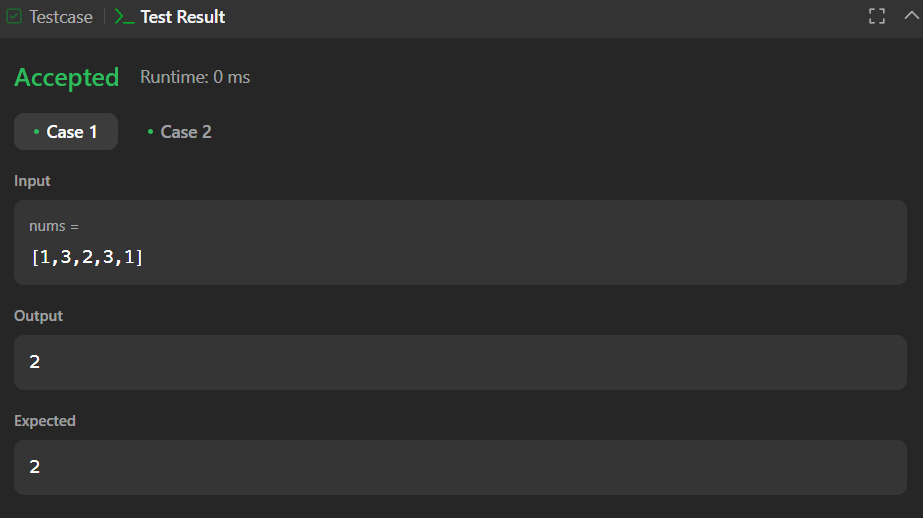
}

}

return reversePairsCount;

}

};



Ques no. 10: Longest Increasing Subsequence II (2407).

Solution :

class MaxSegmentTree {

public:

int n;

vector<int> tree;

MaxSegmentTree(int n\_) : n(n\_) {

int size = (int)(ceil(log2(n)));

size = (2 \* pow(2, size)) - 1;

tree = vector<int>(size);

}

int max\_value() { return tree[0]; }

int query(int l, int r) { return query\_util(0, l, r, 0, n - 1); }

int query\_util(int i, int qL, int qR, int l, int r) {

if (l >= qL && r <= qR) return tree[i];

if (l > qR || r < qL) return INT\_MIN;

int m = (l + r) / 2;

return max(query\_util(2 \* i + 1, qL, qR, l, m), query\_util(2 \* i + 2, qL, qR, m + 1, r));

}

void update(int i, int val) { update\_util(0, 0, n - 1, i, val); }

void update\_util(int i, int l, int r, int pos, int val) {

if (pos < l || pos > r) return;

if (l == r) {

tree[i] = max(val, tree[i]);

return;

}

int m = (l + r) / 2;

update\_util(2 \* i + 1, l, m, pos, val);

update\_util(2 \* i + 2, m + 1, r, pos, val);

tree[i] = max(tree[2 \* i + 1], tree[2 \* i + 2]);

}

};

class Solution {

public:

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

MaxSegmentTree tree(1e5 + 1);

for (int i : nums) {

int lower = max(0, i - k);

int cur = 1 + tree.query(lower, i - 1);

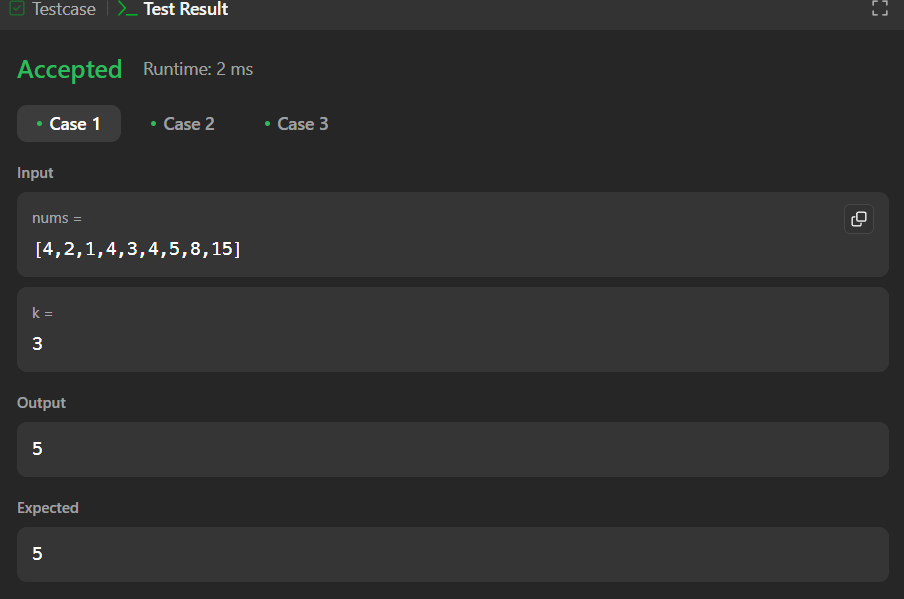
tree.update(i, cur);

}

return tree.max\_value();

}

};



Ques no. 11: Merge Sorted Array (88).

Solution :

class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

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

nums1[i] = nums2[j];

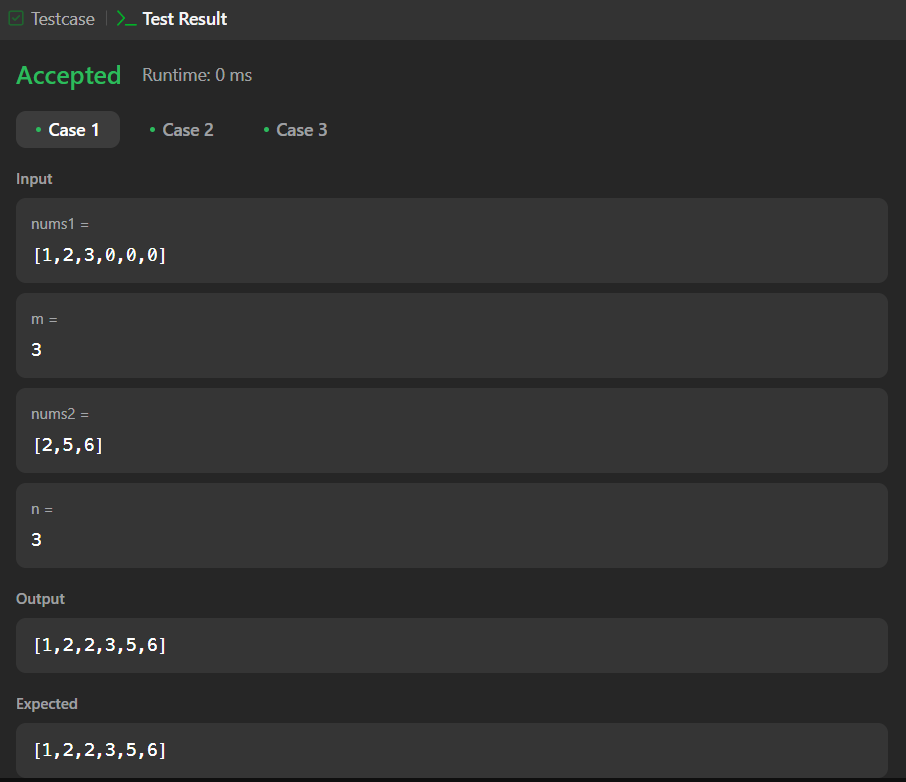
i++;

}

sort(nums1.begin(),nums1.end());

}

};



Ques no. 12: First Bad Version (278).

Solution :

/\*

By :: savetrees

Used :: Binary Search

\*/

class Solution {

public:

int firstBadVersion(int n) {

int low=1;

int high=n;

while(low<=high)

{

int mid=low+(high-low)/2;

if(isBadVersion(mid))high=mid-1;

else low=mid+1;

}

return low;

}

};



Ques no. 13 : Sort Colors (75).

Solution :

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int n = nums.size();

       int low =0, mid =0, high = n-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]);

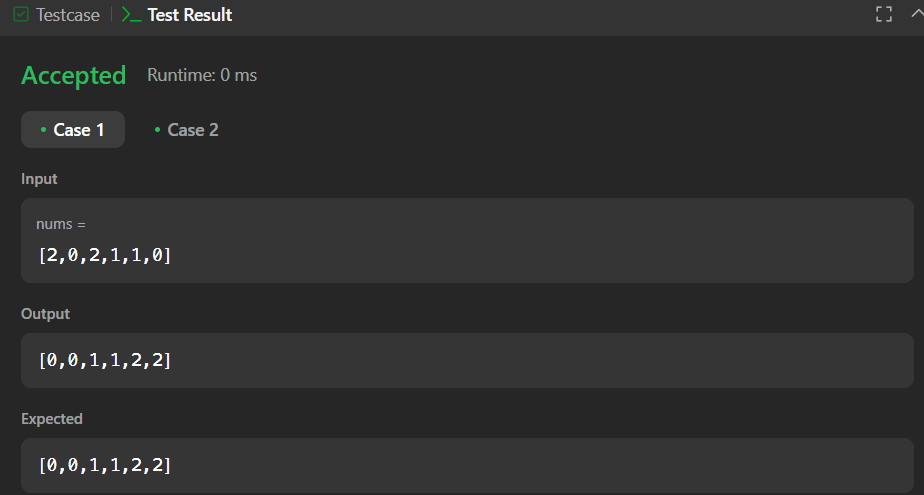
            high--;

        }

       }

    }

};



Ques no. 14 : Top K Frequent Elements (347).

Solution :

class Solution {

public:

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

unordered\_map<int, int> counter;

for (int n : nums) {

counter[n]++;

}

auto comp = [](pair<int, int>& a, pair<int, int>& b) {

return a.second < b.second;

};

priority\_queue<pair<int, int>, vector<pair<int, int>>, decltype(comp)> heap(comp);

for (auto& entry : counter) {

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

}

vector<int> res;

while (k-- > 0) {

res.push\_back(heap.top().first);

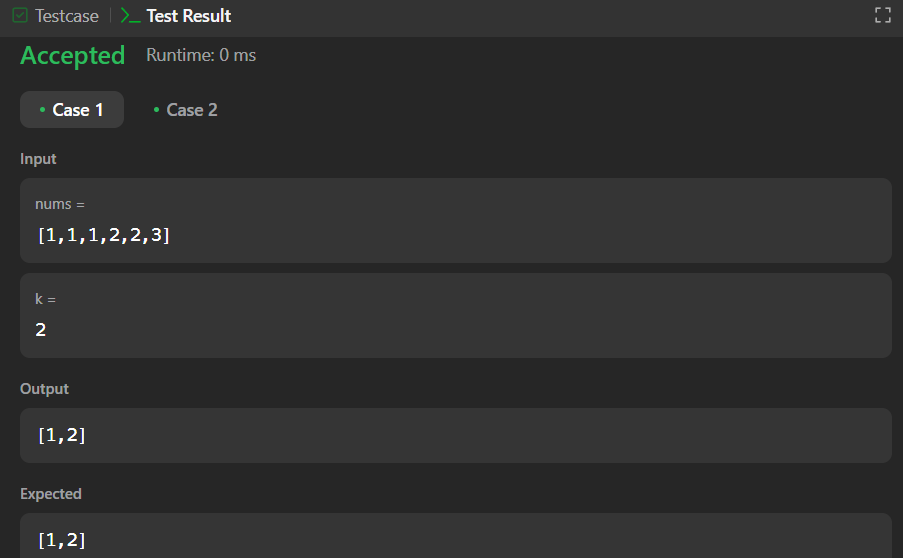
heap.pop();

}

return res;

}

};



Ques no. 15 : Kth Largest Element in an Array(215).

Solution :

class Solution {

public:

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

std::sort(nums.begin(), nums.end(), std::greater<int>());

return nums[k-1];

}

};

