**Experiment 4**

**Name: Jatin Gautam UID: 22BET10252**

**Branch: BE-IT Section/Group: 22BET\_702-B**

**Semester: 6 Date of Performance: 14-02-25**

**Subject Name: Advanced Programming Lab-2 Subject Code: 22ITP-351**

**Problem 1.** Longest Nice Substring - A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

**Code:**

class Solution {

public:

bool check(string s){

vector<int> t1(26, 0), t2(26, 0);

for(int i=0; i<s.length(); i++) if(s[i] >= 'a' && s[i] <= 'z') t1[s[i] - 'a']++;

for(int i=0; i<s.length(); i++) if(s[i] >= 'A' && s[i] <= 'Z') t2[s[i] - 'A']++;

for(int i=0; i<26; i++){

if(t1[i] == 0 && t2[i] == 0) continue;

else if(t1[i] == 0 && t2[i] > 0) return false;

else if(t1[i] > 0 && t2[i] == 0) return false;

}

return true;

}

string longestNiceSubstring(string s) {

if(s == "") return "";

if(check(s)) return s;

unordered\_map<char,int> m;

vector<int> t1(26, 0), t2(26, 0);

for(int i=0; i<s.length(); i++) if(s[i] >= 'a' && s[i] <= 'z') t1[s[i] - 'a']++;

for(int i=0; i<s.length(); i++) if(s[i] >= 'A' && s[i] <= 'Z') t2[s[i] - 'A']++;

for(int i=0; i<26; i++){

if(t1[i] == 0 && t2[i] == 0) continue;

else if(t1[i] == 0 && t2[i] > 0) m[i + 'A']++;

else if(t1[i] > 0 && t2[i] == 0) m[i + 'a']++;

}

vector<string> ans; string temp = "";

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

if(m.find(s[i]) == m.end()) temp += s[i];

else {

ans.push\_back(temp); temp = "";

}

}

if(temp != "") ans.push\_back(temp);

string kk = "";

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

if(ans[i] == "") continue;

string t = longestNiceSubstring(ans[i]);

if(t == "") continue;

if(kk == "" || kk.length() < t.length()) kk = t;

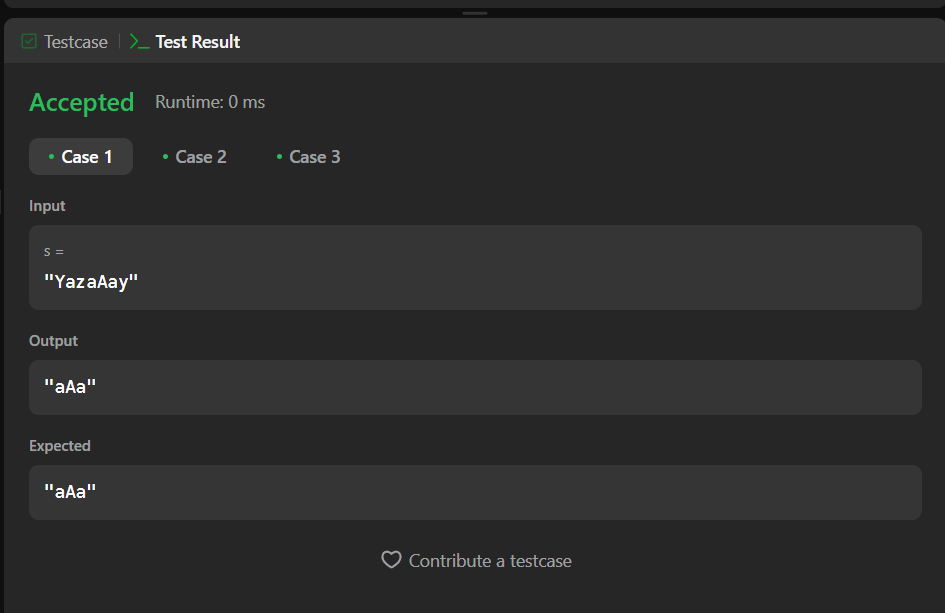
}

return kk;

}

};

**Output:**



**Problem 2.** Reverse Bits - Reverse bits of a given 32 bits unsigned integer

**Code:**

class Solution {

public:

uint32\_t reverseBits(uint32\_t n) {

uint32\_t result = 0;

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

int bit = n & 1;

result = (result << 1) | bit;

n = n >> 1;

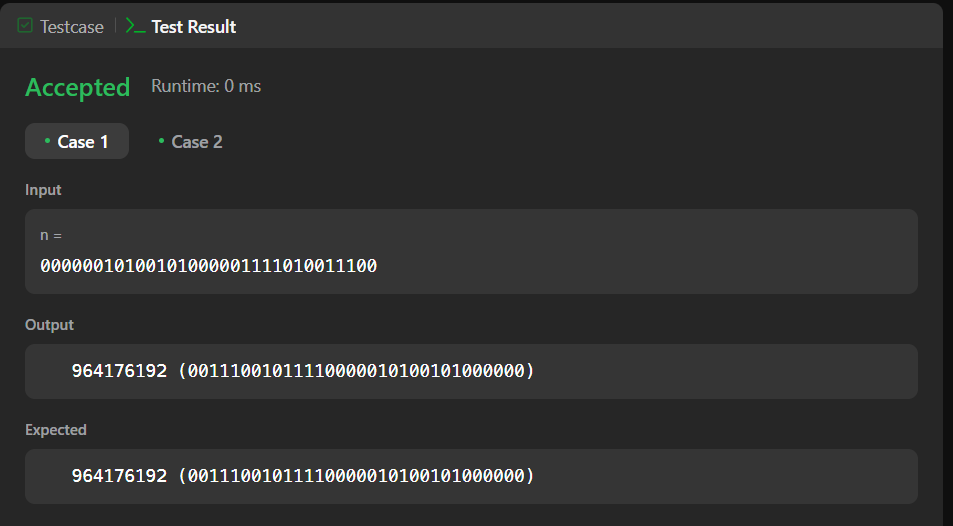
}

return result;

}

};

**Output:**



**Problem 3.** Number of 1 bits. - Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

**Code:**

class Solution {

public:

int hammingWeight(int n) {

int count = 0;

while (n ) {

n = n & (n - 1);

count++;

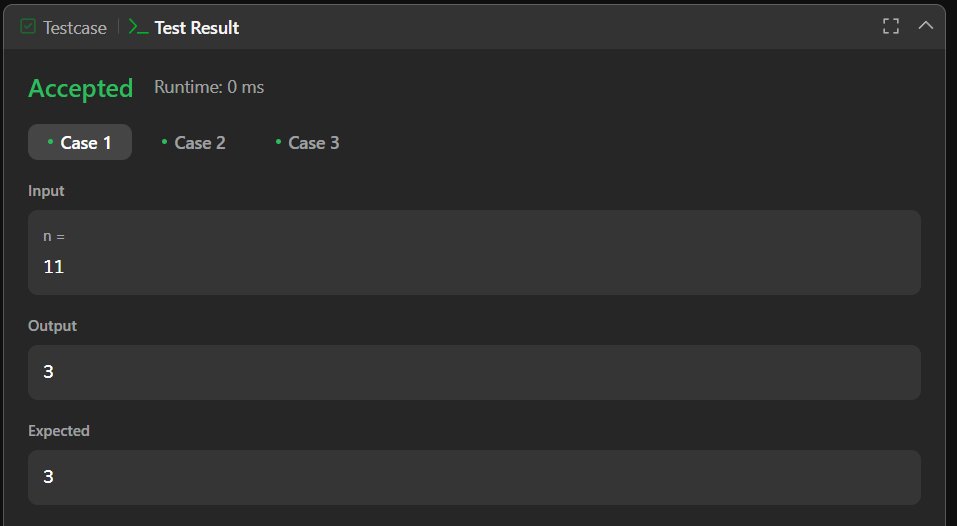
}

return count;

}

};

**Output:**



**Problem 4.** Max Subarray - Given an integer array nums, find the Subarray with the largest sum, and return its sum.

**Code:**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int maxSum = INT\_MIN;

int currentSum = 0;

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

currentSum += nums[i];

if (currentSum > maxSum) {

maxSum = currentSum;

}

if (currentSum < 0) {

currentSum = 0;

}

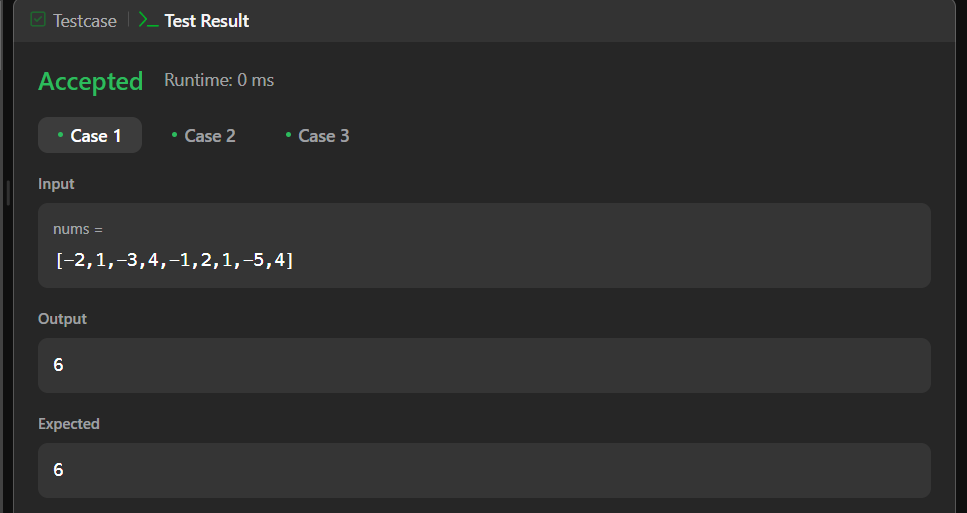
}

return maxSum;

}

};

**Output:**



**Problem 5.** Search 2d matrix 2 - Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

**Code:**

class Solution {

public:

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

int m = matrix.size(), n = m ? matrix[0].size() : 0, r = 0, c = n - 1;

while (r < m && c >= 0) {

if (matrix[r][c] == target) {

return true;

}

matrix[r][c] > target ? c-- : r++;

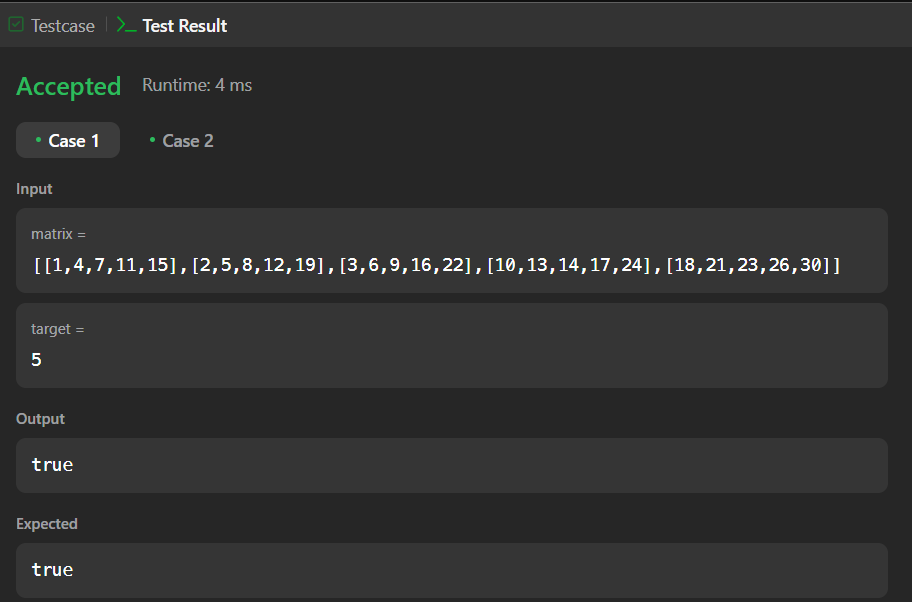
}

return false;

}

};

**Output:**



**Problem 6.** Super Pow-Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

**Code:**

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;

}

};

**Output:**



**Problem 7.** Beautiful Array- An array nums of length n is beautiful if:

* nums is a permutation of the integers in the range [1, n].
* For every 0 <= i < j < n, there is no index k with i < k < j where 2 \* nums[k] == nums[i] + nums[j].

**Code:**

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--);

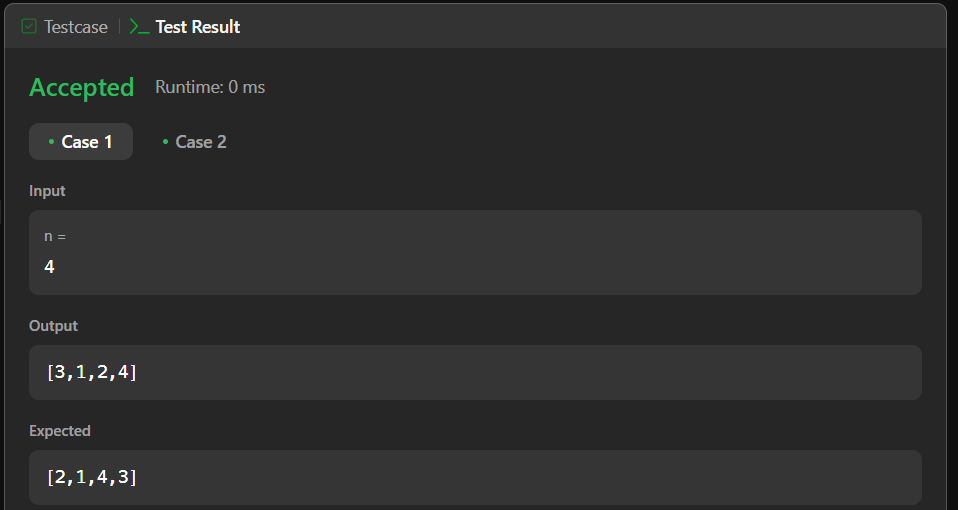
sort(answer.begin(), answer.end(), comp);

return answer;

}

};

**Output:**



**Problem 8.** The Skyline Problem-A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return the skyline formed by these buildings collectively.

**Code:**

class Solution {

public:

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

vector<vector<int>> ans;

multiset<int> pq{0};

vector<pair<int, int>> points;

for(auto b: buildings){

points.push\_back({b[0], -b[2]});

points.push\_back({b[1], b[2]});}

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

int ongoingHeight = 0;

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

int currentPoint = points[i].first;

int heightAtCurrentPoint = points[i].second;

if(heightAtCurrentPoint < 0){

pq.insert(-heightAtCurrentPoint);

} else {

pq.erase(pq.find(heightAtCurrentPoint));}

auto pqTop = \*pq.rbegin();

if(ongoingHeight != pqTop){

ongoingHeight = pqTop;

ans.push\_back({currentPoint, ongoingHeight});

}

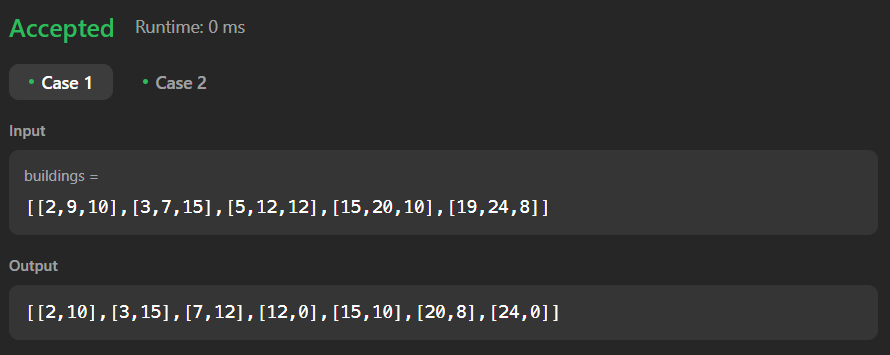
}

return ans;

}

};

**Output:**



**Problem 9.** Reverse Pairs- Given an integer array nums, return the number of reverse pairs in the array.

**Code:**

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++;

}

}

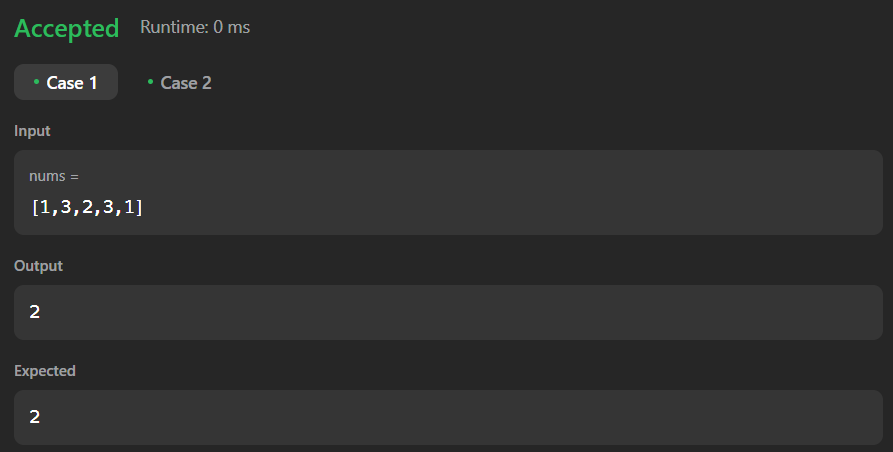
}

return reversePairsCount;

}

};

**Output:**



**Problem 10.** Longest increasing subsequence 2- You are given an integer array nums and an integer k. Find the longest subsequence of nums that meets the following requirements:

**Code:**

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);

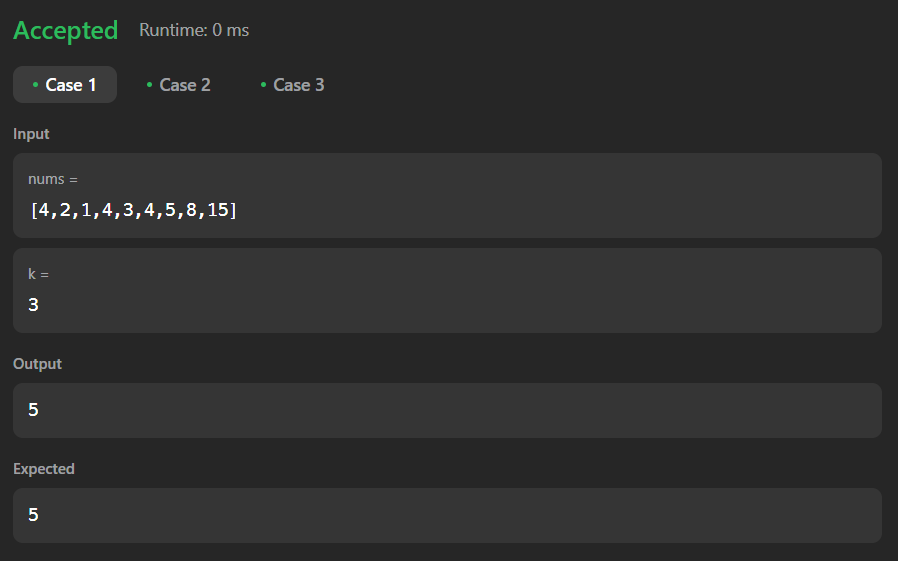
}

return tree.max\_value();

}

};

**Output:**



# 