**Experiment 2**

**Student Name: Mohd Arshad UID: 22BCS10091**

**Branch: CSE Section/Group: FL\_IOT-602**

**Semester: 6TH Date of Performance: 04/01/2**

**Subject Name: AP Subject Code: 22CSP-351**

**Q1: Longest Nice Substring**

**Code:**

class Solution {

public:

string longestNiceSubstring(string s) {

int n = s.size();

if (n < 2) return "";

auto isNice = [](const string& str) {

unordered\_set<char> seenLower, seenUpper;

for (char c : str) {

if (islower(c)) {

seenLower.insert(c);

} else if (isupper(c)) {

seenUpper.insert(tolower(c));

}

}

return seenLower == seenUpper;

};

string longestNice = "";

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

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

string sub = s.substr(i, j - i);

if (isNice(sub) && sub.size() > longestNice.size()) {

longestNice = sub;

}

}

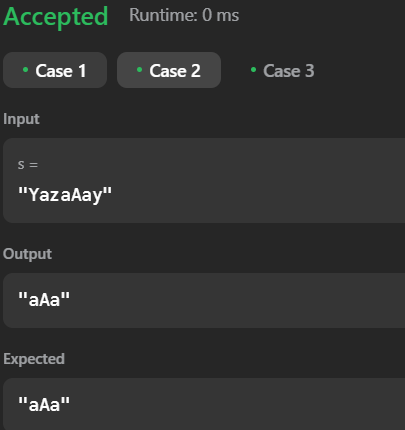
}

return longestNice;

}

};

**Output:**

****

**Q2: 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;

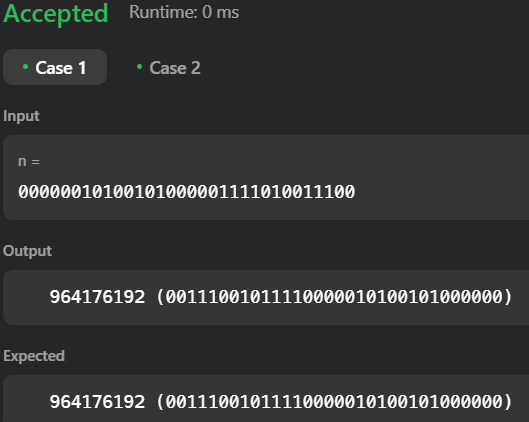
}

return result;

}

};

**Output:**

****

**Q3: Number of 1 bit**

**Code:**

class Solution {

public:

int hammingWeight(int n) {

int count = 0;

while (n) {

count += n & 1;

n >>= 1;

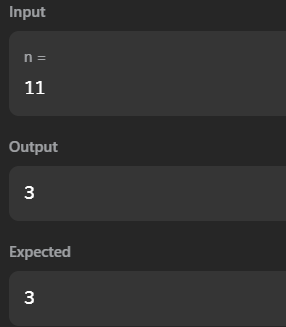
}

return count;

}

};

**Output:**

****

**Q4: 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);

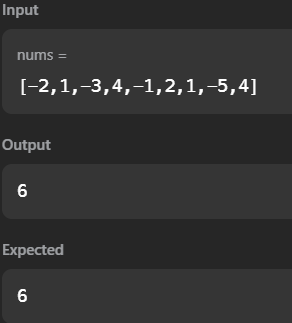
}

return maxSum;

}

};

**Output:**



**Q5: Search a 2D Matrix ||**

**Code:**

class Solution {

public:

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

if (matrix.empty() || matrix[0].empty()) return false;

int row = 0, col = matrix[0].size() - 1;

while (row < matrix.size() && col >= 0) {

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

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

else col--;

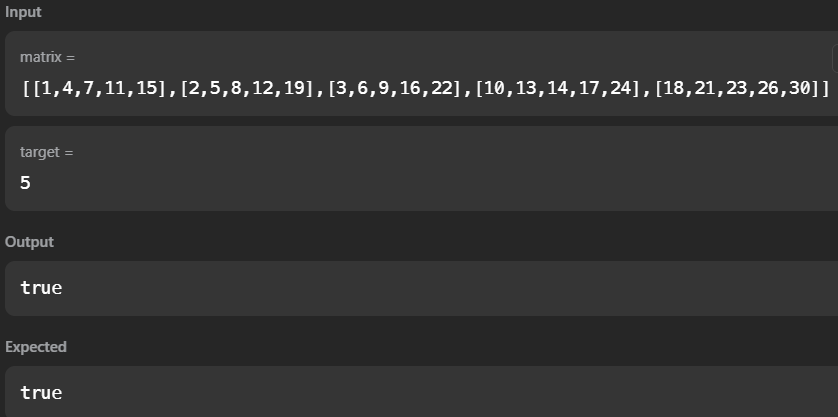
}

return false;

}

};

**Output:**

****

**Q6: Super Pow**

**Code:**

class Solution {

public:

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

const int MOD = 1337;

a %= MOD;

int result = 1;

for (int i = b.size() - 1; i >= 0; --i) {

result = (result \* powMod(a, b[i], MOD)) % MOD;

a = powMod(a, 10, MOD);

}

return result;

}

int powMod(int a, int b, int MOD) {

int res = 1;

while (b > 0) {

if (b % 2 == 1) res = (res \* a) % MOD;

a = (a \* a) % MOD;

b /= 2;

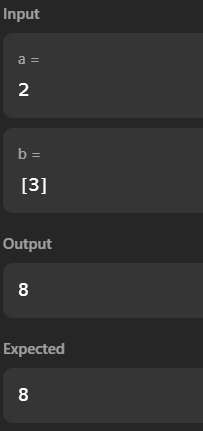
}

return res;

}

};

**Output:**

****

**Q7: Beautiful Array**

**Code:**

class Solution {

public:

vector<int> beautifulArray(int n) {

vector<int> result = {1};

while (result.size() < n) {

vector<int> temp;

for (int x : result) {

if (x \* 2 - 1 <= n) temp.push\_back(x \* 2 - 1);

}

for (int x : result) {

if (x \* 2 <= n) temp.push\_back(x \* 2);

}

result = temp;

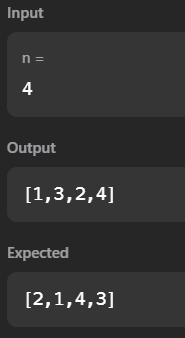
}

return result;

}

};

**Output:**

****

**Q8: The Skyline Problem**

**Code:**

class Solution {

public:

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

vector<vector<int>> result;

vector<pair<int, int>> heights;

for (auto& building : buildings) {

heights.push\_back({building[0], -building[2]});

heights.push\_back({building[1], building[2]});

}

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

multiset<int> activeHeights{0};

int prevMaxHeight = 0;

for (auto& height : heights) {

if (height.second < 0) {

activeHeights.insert(-height.second);

} else {

activeHeights.erase(activeHeights.find(height.second));

}

int currentMaxHeight = \*activeHeights.rbegin();

if (currentMaxHeight != prevMaxHeight) {

result.push\_back({height.first, currentMaxHeight});

prevMaxHeight = currentMaxHeight;

}

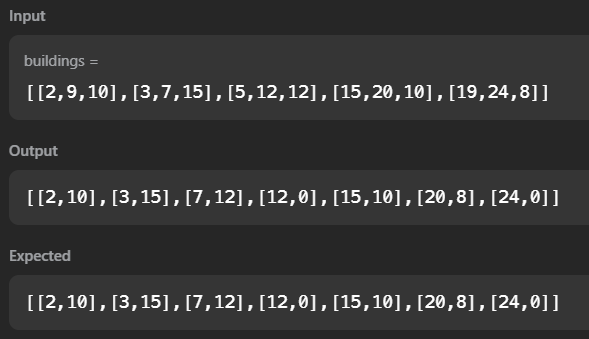
}

return result;

}

};

**Output:**

****

**Q9: Reverse Pairs**

**Code:**

class Solution {

public:

int reversePairs(vector<int>& nums) {

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

}

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);

count += merge(nums, left, mid, right);

return count;

}

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

int count = 0;

int j = mid + 1;

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

while (j <= right && (long long)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 = left; i <= right; ++i) {

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

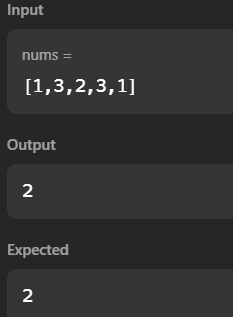
}

return count;

}

};

**Output:**

****

**Q10: Longest Increasing Sequence ||**

**Code:**

class Solution {

public:

int lengthOfLIS(vector<int>& nums) {

if (nums.empty()) return 0;

vector<int> dp(nums.size(), 1);

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

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

if (nums[i] > nums[j]) {

dp[i] = max(dp[i], dp[j] + 1);

}

}

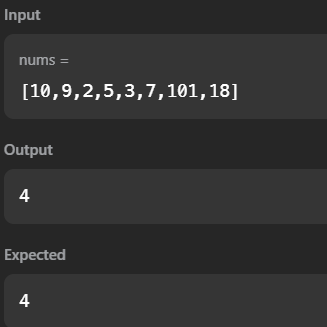
}

return \*max\_element(dp.begin(), dp.end());

}

};

**Output:**

****

**Q11: 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--];

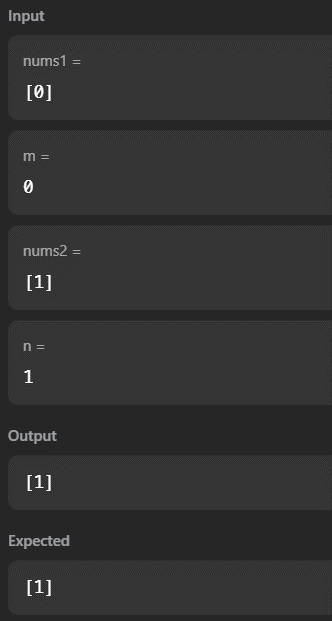
}

while (j >= 0) nums1[k--] = nums2[j--];

}

};

**Output:**



**Q12: 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;

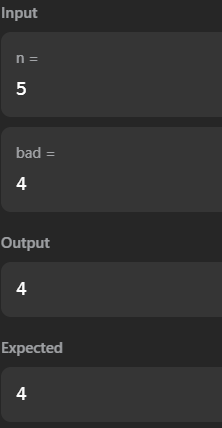
}

return left;

}

};

**Output:**

****

**Q13: Sort Colon**

**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++]);

else if (nums[mid] == 1) mid++;

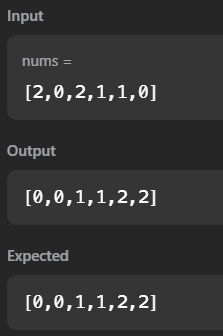
else swap(nums[mid], nums[high--]);

}

}

};

**Output:**

****

**Q14: Top K Frequent Element**

**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>> pq;

for (auto& [num, count] : freq) pq.push({count, num});

vector<int> result;

while (k--) {

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

pq.pop();

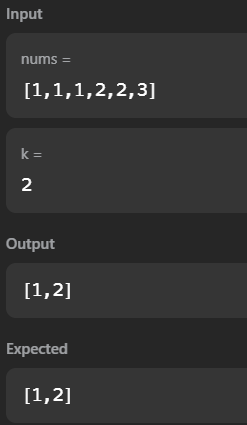
}

return result;

}

};

**Output:**

****

**Q:14 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();

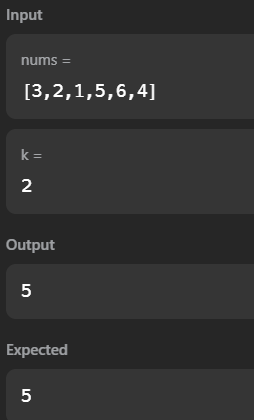
}

return minHeap.top();

}

};

**Output:**

****

**Q15: 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;

if (nums[mid] > nums[mid + 1]) right = mid;

else left = mid + 1;

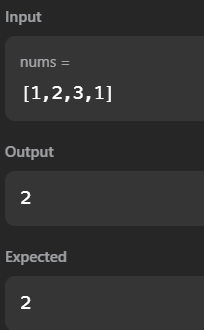
}

return left;

}

};

**Output:**

****

**Q17: Merge Intervals**

**Code:**

class Solution {

public:

vector<vector<int>> merge(vector<vector<int>>& intervals) {

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

vector<vector<int>> merged;

for (auto& interval : intervals) {

if (merged.empty() || merged.back()[1] < interval[0])

merged.push\_back(interval);

else

merged.back()[1] = max(merged.back()[1], interval[1]);

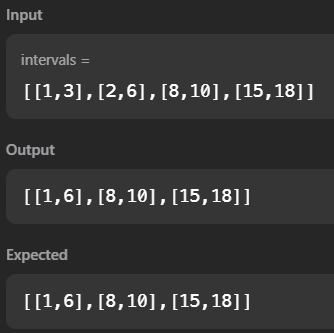
}

return merged;

}

};

**Output:**

****

**Q18: 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 (nums[left] <= nums[mid]) {

if (nums[left] <= target && target < nums[mid]) right = mid - 1;

else left = mid + 1;

} else {

if (nums[mid] < target && target <= nums[right]) left = mid + 1;

else right = mid - 1;

}

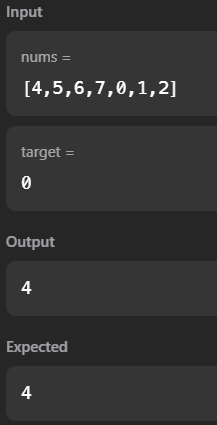
}

return -1;

}

};

**Output:**

****

**Q19: Wiggle Sort II**

**Code:**

class Solution {

public:

void wiggleSort(vector<int>& nums) {

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

if ((i % 2 == 0 && nums[i] > nums[i - 1]) || (i % 2 == 1 && nums[i] < nums[i - 1])) {

swap(nums[i], nums[i - 1]);

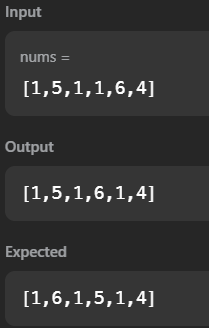
}

}

}

};

**Output:**

****

**Q20: Kth Smallest Element in a Sorted Matrix**

**Code:**

class Solution {

public:

int kthSmallest(vector<vector<int>>& matrix, int k) {

int n = matrix.size();

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

return matrix[a.first][a.second] > matrix[b.first][b.second];

};

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

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

minHeap.push({i, 0});

}

while (k--) {

auto [row, col] = minHeap.top();

minHeap.pop();

if (col + 1 < n) {

minHeap.push({row, col + 1});

}

if (k == 0) {

return matrix[row][col];

}

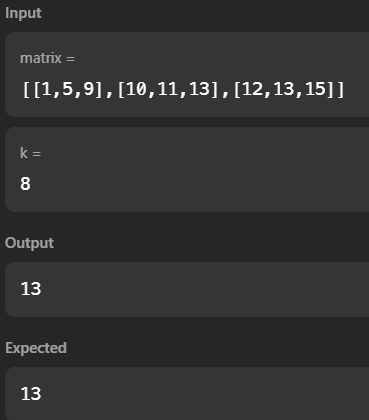
}

return -1;

}

};

**Output:**



**Q21: Median of Two Sorted Arrays**

**Code:**

class Solution {

public:

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

int m = nums1.size(), n = nums2.size();

if (m > n) return findMedianSortedArrays(nums2, nums1);

int left = 0, right = m, midCount = (m + n + 1) / 2;

while (left <= right) {

int mid1 = left + (right - left) / 2;

int mid2 = midCount - mid1;

int l1 = (mid1 == 0) ? INT\_MIN : nums1[mid1 - 1];

int l2 = (mid2 == 0) ? INT\_MIN : nums2[mid2 - 1];

int r1 = (mid1 == m) ? INT\_MAX : nums1[mid1];

int r2 = (mid2 == n) ? INT\_MAX : nums2[mid2];

if (l1 <= r2 && l2 <= r1) {

if ((m + n) % 2 == 0)

return (max(l1, l2) + min(r1, r2)) / 2.0;

else

return max(l1, l2);

} else if (l1 > r2) {

right = mid1 - 1;

} else {

left = mid1 + 1;

}

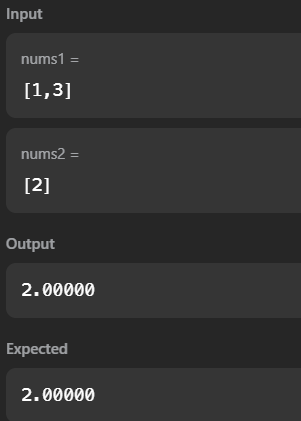
}

return 0.0;

}

};

**Output:**

****