**Experiment 5**

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

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

**Semester: 06TH Date of Performance: 04/03/25**

**Subject Name: Advance Programming Lab II Subject Code: 22CSP-351**

1. **Find the difference**

**Code:**

class Solution {

public:

char findTheDifference(string s, string t) {

char result = 0;

for (char c : s) result ^= c;

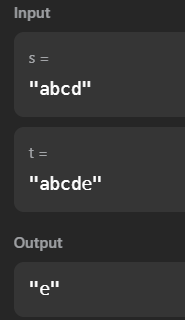
for (char c : t) result ^= c;

return result;

}

};

**Output:**

****

1. **Largest Perimeter Triangle**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Solution {

public:

int largestPerimeter(vector<int>& nums) {

sort(nums.begin(), nums.end(), greater<int>()); // Sort in descending order

for (int i = 0; i < nums.size() - 2; i++) {

if (nums[i] < nums[i + 1] + nums[i + 2]) { // Triangle inequality check

return nums[i] + nums[i + 1] + nums[i + 2];

}

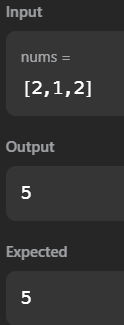
}

return 0; // No valid triangle found

}

};

**Output:**

****

1. **Third Maximum Number**

**Code:**

class Solution {

public:

int thirdMax(vector<int>& nums) {

long long first = LLONG\_MIN, second = LLONG\_MIN, third = LLONG\_MIN;

for (int num : nums) {

if (num == first || num == second || num == third) continue;

if (num > first) {

third = second;

second = first;

first = num;

} else if (num > second) {

third = second;

second = num;

} else if (num > third) {

third = num;

}

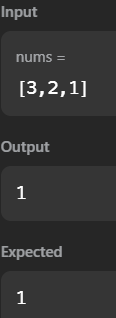
}

return (third == LLONG\_MIN) ? first : third;

}

};

**Output:**

****

1. **Sort Character By Frequency**

**Code:**

class Solution {

public:

string frequencySort(string s) {

unordered\_map<char, int> om;

for (char ch : s) {

om[ch]++;

}

vector<pair<int, char>> freq;

for (auto it : om) {

freq.push\_back({it.second, it.first});

}

sort(freq.rbegin(), freq.rend());

string result = "";

for (auto it : freq) {

result += string(it.first, it.second);

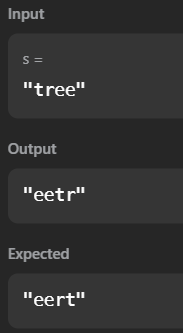
}

return result;

}

};

**Output:**

****

1. **Minimum Number of Arrows to Burst Balloons**

**Code:**

class Solution {

public:

int findMinArrowShots(vector<vector<int>>& points) {

if (points.empty()) return 0;

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

return a[1] < b[1]; // Sort by end coordinate

});

int arrows = 1;

int prevEnd = points[0][1];

for (const auto& balloon : points) {

if (balloon[0] > prevEnd) { // If balloon starts after the last arrow shot

arrows++;

prevEnd = balloon[1]; // Update the end of the new burst region

}

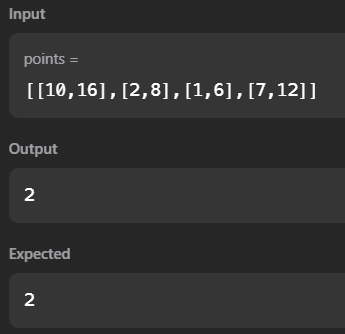
}

return arrows;

}

};

**Output:**

****

1. **Boats to Save People**

**Code:**

class Solution {

public:

int numRescueBoats(vector<int>& people, int limit) {

sort(people.begin(), people.end()); // Sort people by weight

int i = 0, j = people.size() - 1;

int boats = 0;

while (i <= j) {

if (people[i] + people[j] <= limit) {

i++; // Lightest person boards with the heaviest

}

j--; // Heaviest person boards (either alone or with lightest)

boats++;

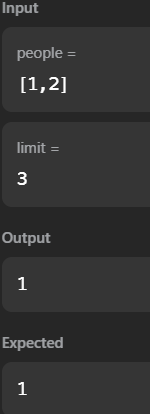
}

return boats;

}

};

**Output:**

****

1. **K Closest Points to Origin**

**Code:**

class Solution {

public:

vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {

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

for (const auto& point : points) {

int dist = point[0] \* point[0] + point[1] \* point[1]; // Squared Euclidean distance

minHeap.push({dist, point});

}

vector<vector<int>> result;

while (k-- > 0) {

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

minHeap.pop();

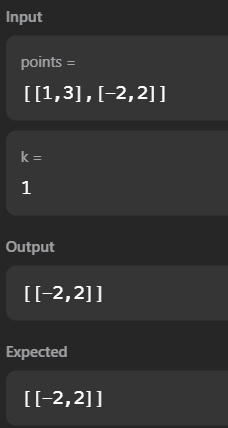
}

return result;

}

};

**Output:**

****

1. **Reduce Array Size to The Half**

**Code:**

class Solution {

public:

int minSetSize(vector<int>& arr) {

unordered\_map<int, int> freq;

for (int num : arr) freq[num]++; // Count occurrences

priority\_queue<int> maxHeap;

for (auto& [num, count] : freq) {

maxHeap.push(count); // Push frequencies into max heap

}

int removed = 0, sizeReduced = 0, halfSize = arr.size() / 2;

while (sizeReduced < halfSize) {

sizeReduced += maxHeap.top();

maxHeap.pop();

removed++;

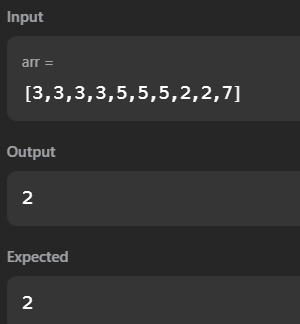
}

return removed;

}

};

**Output:**

****