Day 2

Name:- Dependra **UID:- 22BCS10256** **Section:- KPIT_901/A**

Very Easy

Q 1 : Majority Elements

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊ n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:
Input: nums = [3,2,3]
Output: 3

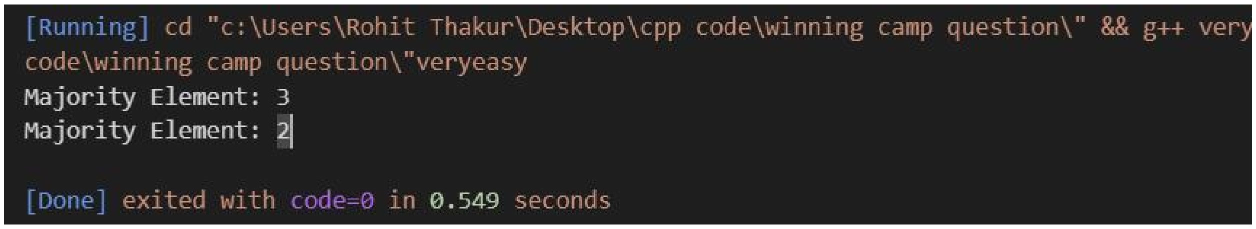
Example 2:
Input: nums = [2,2,1,1,1,2,2]
Output: 2

Constraints:
n == nums.length
1 <= n <= 5 * 104
-109 <= nums[i] <= 109

Code:-

```
#include <vector>
#include <iostream>
using namespace std;

int majorityElement(vector<int>& nums) {
    int count = 0, candidate = 0;
    for (int num : nums) {
        if (count == 0) {
            candidate = num;
        }
        count += (num == candidate) ? 1 : -1;
    }
    return candidate;
}

int main() {
    vector<int> nums1 = {3, 2, 3};
    cout << "Majority Element: " << majorityElement(nums1) << endl;

    vector<int> nums2 = {2, 2, 1, 1, 1, 2, 2};
    cout << "Majority Element: " << majorityElement(nums2) << endl;

    return 0;
}
```

Output:-



```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\" && g++ very
code\winning camp question\"veryeasy
Majority Element: 3
Majority Element: 2

[Done] exited with code=0 in 0.549 seconds
```

Easy

Question 1. Pascal's Triangle

Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Example 1:

Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

Input: numRows = 1
Output: [[1]]

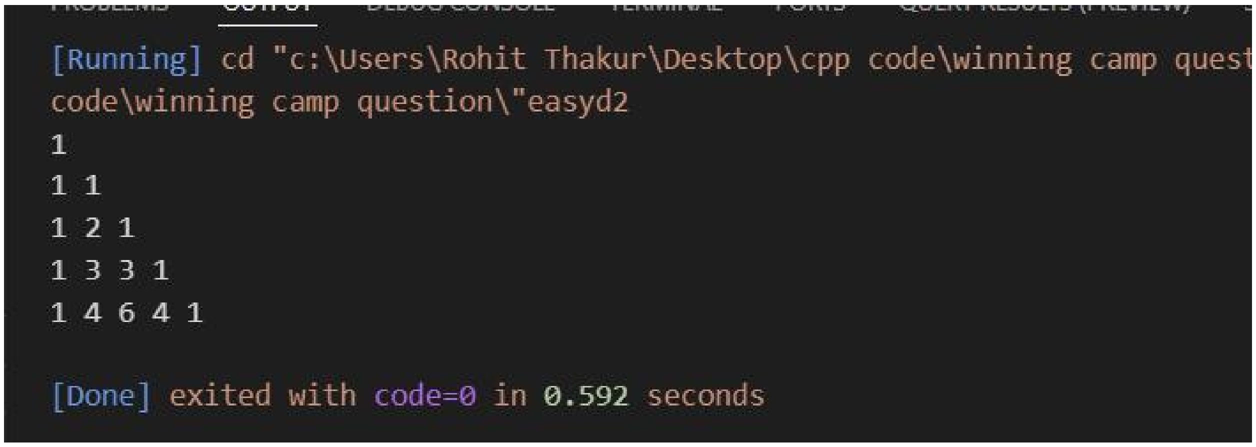
Constraints:

1 <= numRows <= 30

Code:-

```
#include <vector>
#include <iostream>
using namespace std;

vector<vector<int>> generate(int numRows) {
    vector<vector<int>> triangle(numRows);

    for (int i = 0; i < numRows; i++) {
        triangle[i].resize(i + 1);
        triangle[i][0] = triangle[i][i] = 1;

        for (int j = 1; j < i; j++) {
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }
    }

    return triangle;
}

int main() {
    int numRows = 5;
    vector<vector<int>> result = generate(numRows);

    for (const auto& row : result) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:-



```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp code\winning camp quest
code\winning camp question\"easyd2
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

[Done] exited with code=0 in 0.592 seconds
```

Medium:

Question 1. Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Example 1:

Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]
Output: 1

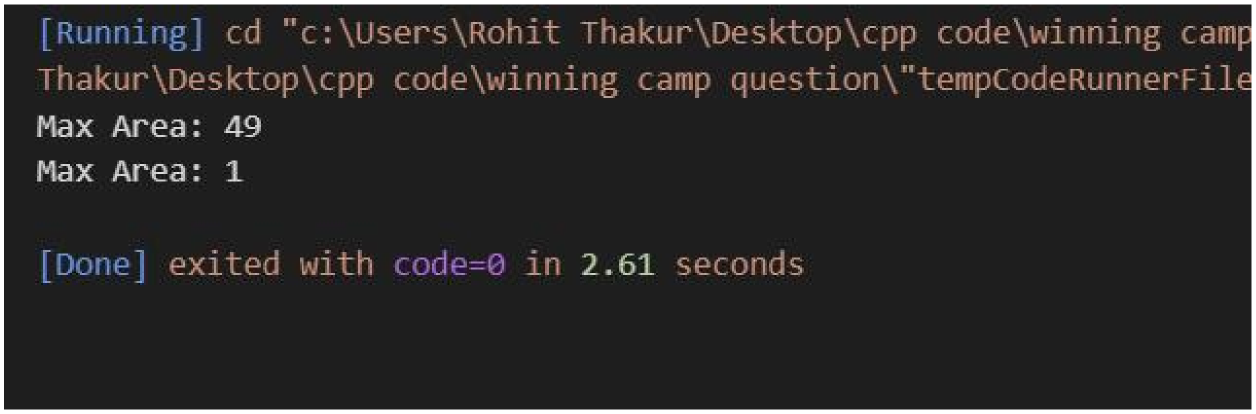
Constraints:

n == height.length
2 <= n <= 105
0 <= height[i] <= 104

Code:-

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int maxArea(vector<int>& height) {
    int max_water = 0;
    int left = 0, right = height.size() - 1;

    while (left < right) {
        int width = right - left;
        int current_height = min(height[left], height[right]);
        max_water = max(max_water, width * current_height);
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }

    return max_water;
}

int main() {
    vector<int> height1 = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << "Max Area: " << maxArea(height1) << endl;

    vector<int> height2 = {1, 1};
    cout << "Max Area: " << maxArea(height2) << endl;

    return 0;
}
```

Output:-



```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp code\winning camp
Thakur\Desktop\cpp code\winning camp question\"tempCodeRunnerFile
Max Area: 49
Max Area: 1

[Done] exited with code=0 in 2.61 seconds
```

<u>Hard</u>

Question 1. Maximum Number of Groups Getting Fresh Donuts

There is a donuts shop that bakes donuts in batches of batchSize. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer batchSize and an integer array groups, where groups[i] denotes that there is a group of groups[i] customers that will visit the shop. Each customer will get exactly one donut.

When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

Example 1:

Input: batchSize = 3, groups = [1,2,3,4,5,6]
Output: 4
Explanation: You can arrange the groups as [6,2,4,5,1,3]. Then the 1st, 2nd, 4th, and 6th groups will be happy.
Example 2:

Input: batchSize = 4, groups = [1,3,2,5,2,2,1,6]
Output: 4

Constraints:

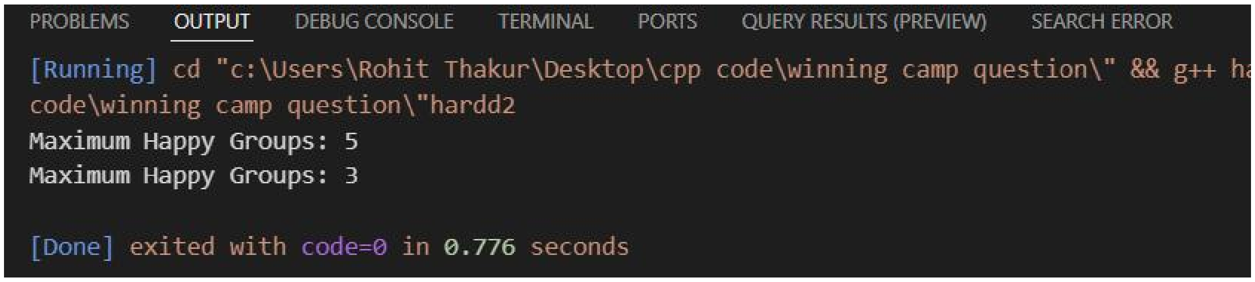
1 <= batchSize <= 9
1 <= groups.length <= 30
1 <= groups[i] <= 109

Code:-

```
#include <vector>
#include <unordered_map>
#include <iostream>
using namespace std;

class Solution {
public:
  int maxHappyGroups(int batchSize, vector<int>& groups) {
    vector<int> count(batchSize, 0);
    for (int group : groups) {
      count[group % batchSize]++;
```

```
        }
        int happyGroups = count[0];
        unordered_map<string, int> memo;
        return happyGroups + dfs(batchSize, count, memo);
    }

private:
    int dfs(int batchSize, vector<int>& count, unordered_map<string, int>& memo) {
        string key = encode(count);
        if (memo.find(key) != memo.end()) {
            return memo[key];
        }

        int maxHappy = 0;

        for (int i = 1; i < batchSize; i++) {
            if (count[i] > 0) {
                count[i]--;
                int additionalHappy = (i == batchSize - 1 || sum(count) % batchSize == 0) ? 1 : 0;
                maxHappy = max(maxHappy, additionalHappy + dfs(batchSize, count, memo));
                count[i]++;
            }
        }

        memo[key] = maxHappy;
        return maxHappy;
    }

    string encode(vector<int>& count) {
        string key = "";
        for (int c : count) {
            key += to_string(c) + ",";
        }
        return key;
    }

    int sum(vector<int>& count) {
        int total = 0;
        for (int c : count) {
            total += c;
        }
        return total;
    }
};

int main() {
    Solution solution;

    int batchSize1 = 3;
    vector<int> groups1 = {1, 2, 3, 4, 5, 6};
    cout << "Maximum Happy Groups: " << solution.maxHappyGroups(batchSize1, groups1) <<
endl;

    int batchSize2 = 4;
    vector<int> groups2 = {1, 3, 2, 5, 2, 2, 1, 6};
    cout << "Maximum Happy Groups: " << solution.maxHappyGroups(batchSize2, groups2) <<
endl;

    return 0;
}
```

Output:-



```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp code\winning camp question\" && g++ ha
code\winning camp question\"hardd2
Maximum Happy Groups: 5
Maximum Happy Groups: 3

[Done] exited with code=0 in 0.776 seconds
```

Very Hard

Question 2. Minimum Number of People to Teach

On a social network consisting of m users and some friendships between users, two users can communicate with each other if they know a common language.

You are given an integer n, an array languages, and an array friendships where:

There are n languages numbered 1 through n,
languages[i] is the set of languages the i^{th} user knows, and
friendships[i] = [u_i, v_i] denotes a friendship between the users u_i and v_i.
You can choose one language and teach it to some users so that all friends can communicate with each other. Return the minimum number of users you need to teach.

Note that friendships are not transitive, meaning if x is a friend of y and y is a friend of z, this doesn't guarantee that x is a friend of z.

Example 1:

Input: n = 2, languages = [[1],[2],[1,2]], friendships = [[1,2],[1,3],[2,3]]
Output: 1
Explanation: You can either teach user 1 the second language or user 2 the first language.
Example 2:

Input: n = 3, languages = [[2],[1,3],[1,2],[3]], friendships = [[1,4],[1,2],[3,4],[2,3]]
Output: 2
Explanation: Teach the third language to users 1 and 3, yielding two users to teach.

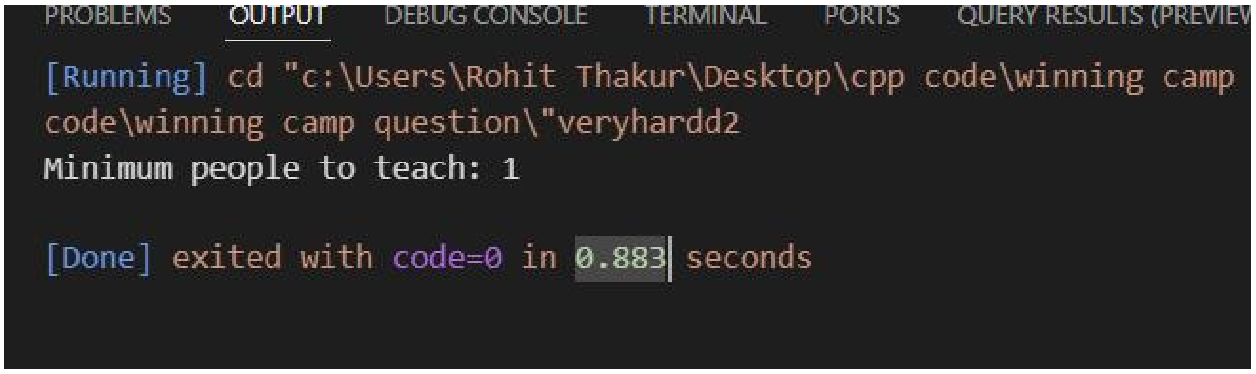
Constraints:

2 <= n <= 500
languages.length == m
1 <= m <= 500
1 <= languages[i].length <= n
1 <= languages[i][j] <= n
1 <= u_i < v_i <= languages.length
1 <= friendships.length <= 500
All tuples (u_i, v_i) are unique
languages[i] contains only unique values

Code:-

```
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <algorithm>
#include <climits> // Include this for INT_MAX
#include <cstdio>  // For printf

using namespace std;

int minTeach(int n, vector<vector<int>>& languages, vector<vector<int>>& friendships) {
    int m = languages.size();

    // Convert each person's languages into a set for quick lookup
    vector<unordered_set<int>> languageSet(m);
    for (int i = 0; i < m; ++i) {
        languageSet[i] = unordered_set<int>(languages[i].begin(), languages[i].end());
    }

    // Find all friend pairs who cannot communicate
    vector<pair<int, int>> cannotCommunicate;
    for (const auto& f : friendships) {
        int a = f[0] - 1, b = f[1] - 1; // Convert to 0-based indexing
        bool canCommunicate = false;
        for (int lang : languageSet[a]) {
            if (languageSet[b].count(lang)) {
                canCommunicate = true;
                break;
            }
        }
        if (!canCommunicate) {
            cannotCommunicate.emplace_back(a, b);
        }
    }

    // If all friends can already communicate, no teaching is required
    if (cannotCommunicate.empty()) {
        return 0;
    }

    // Try teaching each language and calculate the minimum number of people to teach
    int minTeachCount = INT_MAX;
    for (int lang = 1; lang <= n; ++lang) {
        unordered_set<int> toTeach; // Set of people to teach the current language

        for (const auto& pair : cannotCommunicate) {
            int a = pair.first, b = pair.second;
            if (!languageSet[a].count(lang)) {
                toTeach.insert(a);
            }
            if (!languageSet[b].count(lang)) {
                toTeach.insert(b);
            }
        }

        minTeachCount = min(minTeachCount, static_cast<int>(toTeach.size()));
    }

    return minTeachCount;
}

int main() {
    vector<vector<int>> languages = {{1, 2}, {3, 4}, {1, 3}};
    vector<vector<int>> friendships = {{1, 2}, {2, 3}, {1, 3}};
    int n = 4;

    int result = minTeach(n, languages, friendships);
    printf("Minimum people to teach: %d\n", result); // Output: 1
    return 0;
}
```

Output:-



```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp code\winning camp
code\winning camp question\"veryhardd2
Minimum people to teach: 1

[Done] exited with code=0 in 0.883 seconds
```