22BCS14842
GURJOT SINGH
22BCS_IOT-605-'A'
WORKSHEET-8,9,10
WORKSHEET-8

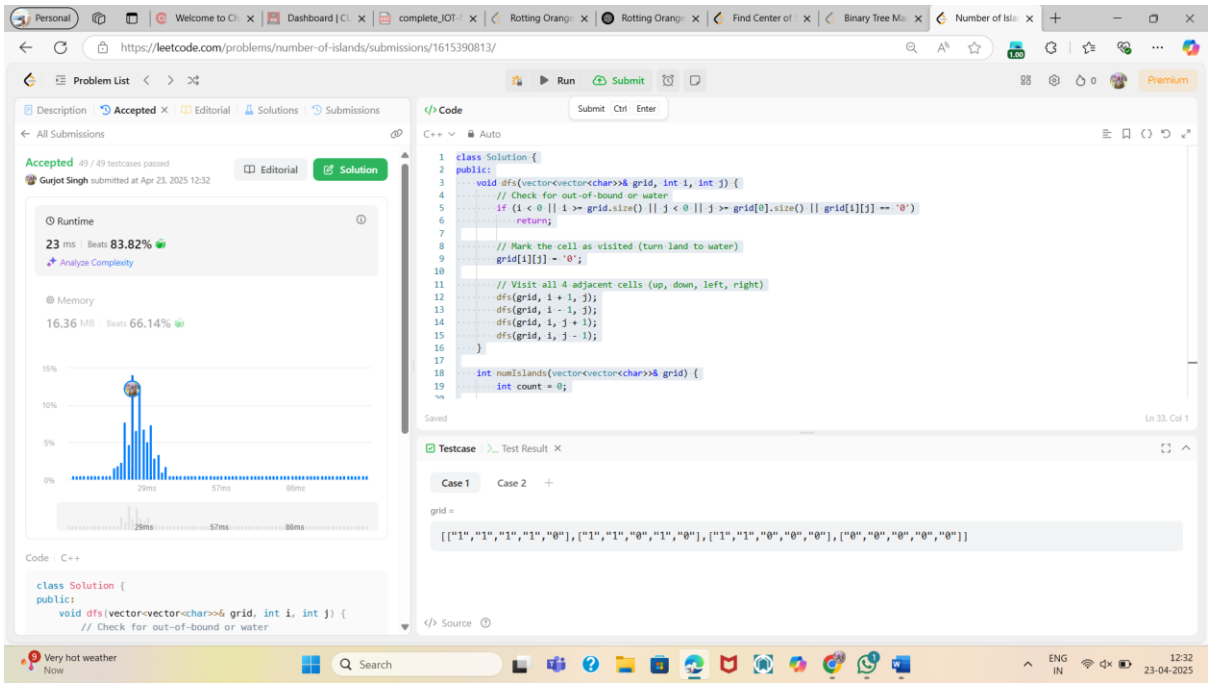
Ques- Maximum Units on a Truck

You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxes, unitsPerBox].
You are also given an integer truckSize, which is the maximum number of boxes you can put on the truck.

Return the maximum total number of units that can be put on the truck.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
    // Sort by units per box in descending order
    sort(boxTypes.begin(), boxTypes.end(), [](vector<int>& a, vector<int>& b) {
        return a[1] > b[1];
    });
    int maxUnits = 0;
    for (auto& box : boxTypes) {
        int boxCount = min(truckSize, box[0]);
        maxUnits += boxCount * box[1];
        truckSize -= boxCount;
        if (truckSize == 0) break;
    }

    return maxUnits;
}
```

```
int main() {
    vector<vector<int>> boxTypes = {{1, 3}, {2, 2}, {3, 1}};
    int truckSize = 4;
    cout << maximumUnits(boxTypes, truckSize) << endl; // Output: 8
    return 0;
}
```



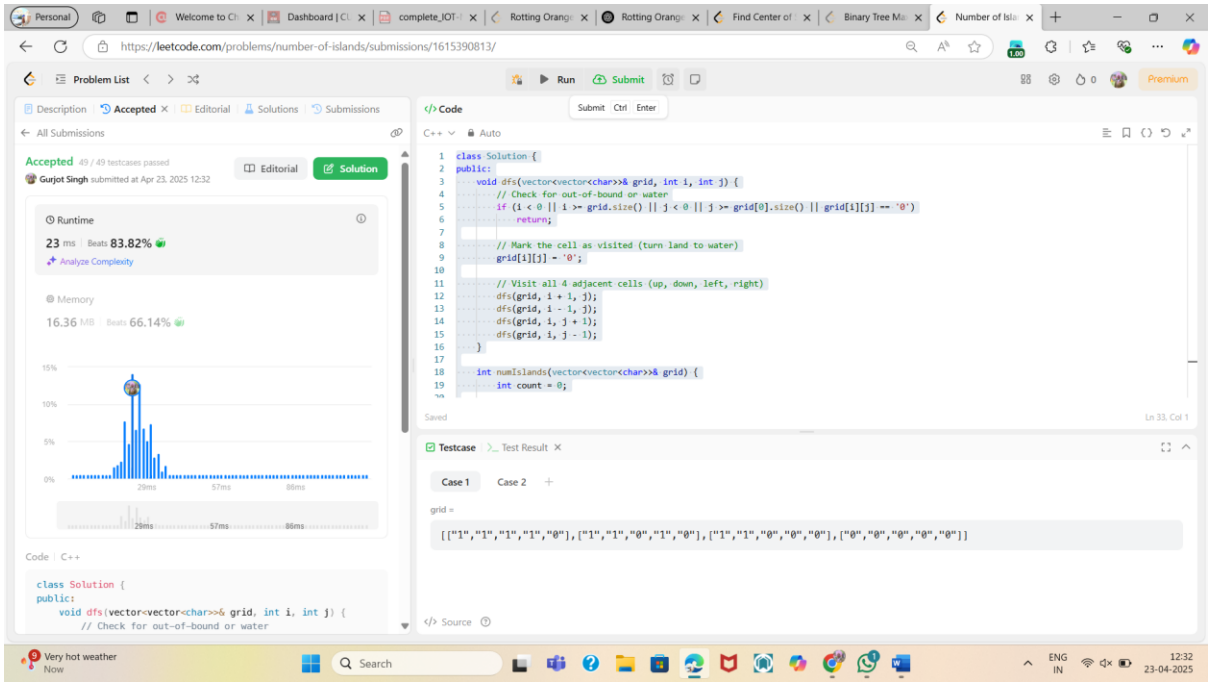
Ques-Minimum Operations to Make the Array Increasing

You are given an integer array nums. In one operation, you can choose an element and increase it by 1.

Return the minimum number of operations to make nums strictly increasing.

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

int minOperations(vector<int>& nums) {
    int operations = 0;
    for (int i = 1; i < nums.size(); i++) {
```

```
        if (nums[i] <= nums[i - 1]) {
            operations += (nums[i - 1] + 1 - nums[i]);
            nums[i] = nums[i - 1] + 1;
        }
    }
    return operations;
}

int main() {
    vector<int> nums = {1, 1, 1};
    cout << minOperations(nums) << endl; // Output: 3
    return 0;
}
```



Ques-Remove Stones to Maximize Total

You are given a max-heap problem: each pile of stones can have some stones removed, and after each operation, the pile becomes half its size (rounded down). You can perform k such operations.

Return the maximum number of stones left after performing all k operations.

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

int maxKelements(vector<int>& piles, int k) {
    priority_queue<int> pq(piles.begin(), piles.end());

    while (k-- > 0 && !pq.empty()) {
        int top = pq.top();
        pq.pop();
        pq.push((top + 2) / 3); // divide by 3 rounded up
    }

    long long total = 0;
    while (!pq.empty()) {
        total += pq.top();
        pq.pop();
    }

    return total;
}

int main() {
    vector<int> piles = {5, 4, 9};
    int k = 2;
    cout << maxKelements(piles, k) << endl; // Output: 12
    return 0;
}
```

WORKSHEET-9

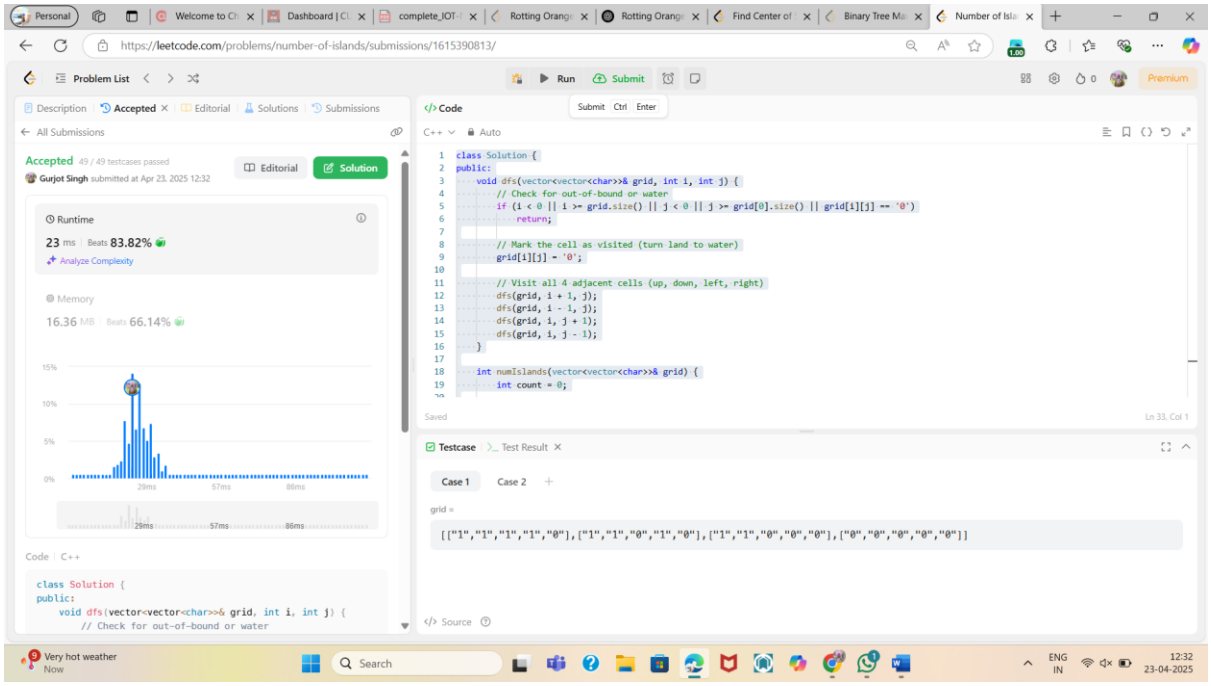
Ques- Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        // Check for out-of-bound or water
        if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0')
            return;

        // Mark the cell as visited (turn land to water)
        grid[i][j] = '0';

        // Visit all 4 adjacent cells (up, down, left, right)
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }

    int numIslands(vector<vector<char>>& grid) {
        int count = 0;

        for (int i = 0; i < grid.size(); i++) {
            for (int j = 0; j < grid[0].size(); j++) {
```

```
            if (grid[i][j] == '1') {
                count++;            // Found an island
                dfs(grid, i, j);    // Mark all connected land
            }
        }
    }

    return count;
  }
};
```



Ques- Course Schedule

Problem:

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1.
Some courses may have prerequisites, given as a list of prerequisites where prerequisites[i] = [a, b] means you must take course b before course a.

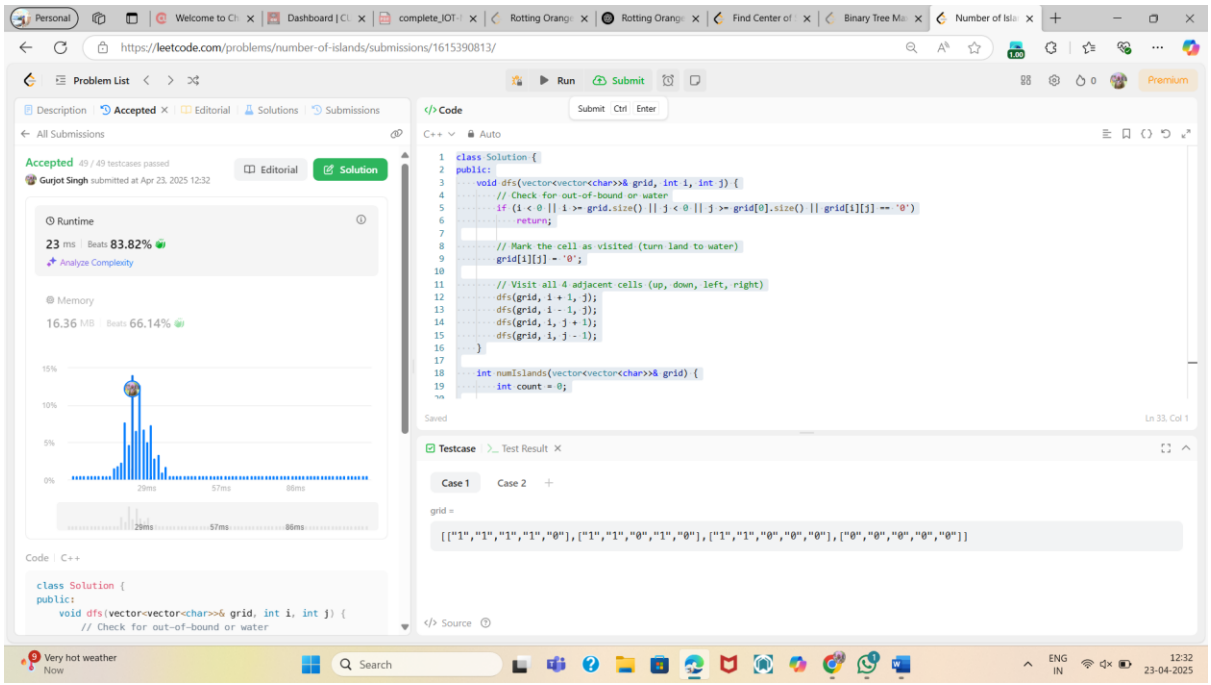
Return true if you can finish all courses. Otherwise, return false.

cpp

CopyEdit

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

bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    vector<vector<int>> adj(numCourses);
    vector<int> indegree(numCourses, 0);

    for (auto& p : prerequisites) {
        adj[p[1]].push_back(p[0]);
        indegree[p[0]]++;
    }

    queue<int> q;
    for (int i = 0; i < numCourses; i++) {
        if (indegree[i] == 0) q.push(i);
    }

    int count = 0;
    while (!q.empty()) {
        int curr = q.front();
        q.pop();
        count++;

        for (int neighbor : adj[curr]) {
            indegree[neighbor]--;
            if (indegree[neighbor] == 0)
                q.push(neighbor);
```

```
        }
    }

    return count == numCourses;
}

int main() {
    int numCourses = 2;
    vector<vector<int>> prerequisites = {{1, 0}};

    cout << (canFinish(numCourses, prerequisites) ? "true" : "false") << endl;
    return 0;
}
```



Ques- Longest Increasing Path in a Matrix

Problem:

Given an m x n integers matrix, return the length of the longest increasing path in the matrix.

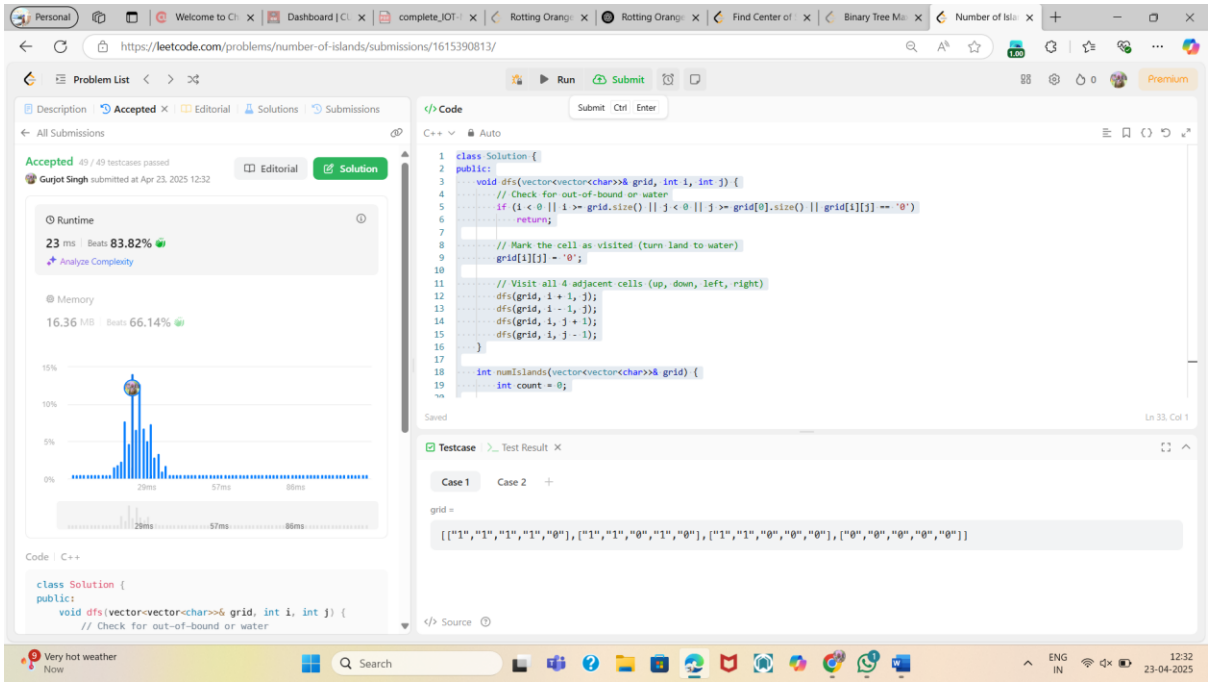
From each cell, you can move in 4 directions (up, down, left, right). You may not move diagonally or move outside the boundary.

cpp

CopyEdit

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

class Solution {
public:
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        int m = matrix.size();
        int n = matrix[0].size();
        vector<vector<int>> dp(m, vector<int>(n, 0));
        int maxLen = 0;

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                maxLen = max(maxLen, dfs(matrix, dp, i, j));
            }
        }
        return maxLen;
    }

private:
    vector<int> dir = {-1, 0, 1, 0, -1};

    int dfs(vector<vector<int>>& matrix, vector<vector<int>>& dp, int i, int j) {
        if (dp[i][j] != 0) return dp[i][j];
```

```
        int maxPath = 1;
        for (int d = 0; d < 4; d++) {
            int x = i + dir[d];
            int y = j + dir[d + 1];

            if (x >= 0 && x < matrix.size() &&
                y >= 0 && y < matrix[0].size() &&
                matrix[x][y] > matrix[i][j]) {
                maxPath = max(maxPath, 1 + dfs(matrix, dp, x, y));
            }
        }
        dp[i][j] = maxPath;
        return maxPath;
    }
};

int main() {
    Solution sol;
    vector<vector<int>> matrix = {
        {9, 9, 4},
        {6, 6, 8},
        {2, 1, 1}
    };

    cout << sol.longestIncreasingPath(matrix) << endl;
    return 0;
}
```



WORKSHEET-10

Ques- Find the Celebrity

Suppose you are at a party with n people (labeled from 0 to n - 1).
A celebrity is someone who:

- **Everyone knows them.**
- **They know no one.**

Return the celebrity's label or -1 if no celebrity exists.

```
class Solution {
public:
    int findCelebrity(int n) {
        int candidate = 0;

        // Step 1: Find the candidate
        for (int i = 1; i < n; ++i) {
            if (knows(candidate, i)) {
                candidate = i;
```

```
            }
        }

        // Step 2: Verify the candidate
        for (int i = 0; i < n; ++i) {
            if (i != candidate) {
                if (knows(candidate, i) || !knows(i, candidate)) {
                    return -1;
                }
            }
        }

        return candidate;
    }
};
```

Ques- Pascal's Triangle

Given an integer numRows, generate the first numRows of Pascal's Triangle.

Each number is the sum of the two numbers directly above it.

```
#include <iostream>
#include <vector>
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 (auto& row : result) {
        for (int val : row) cout << val << " ";
        cout << endl;
    }

    return 0;
}
```

Ques- Hamming Distance

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, return the Hamming distance between them.

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

int hammingDistance(int x, int y) {
    int xorVal = x ^ y;
    int count = 0;
```

```
    while (xorVal) {
        count += xorVal & 1;
        xorVal >>= 1;
    }
    return count;
}

int main() {
    int x = 1, y = 4;
    cout << hammingDistance(x, y) << endl; // Output: 2
    return 0;
}
```