Day 2

Array & Linked list

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

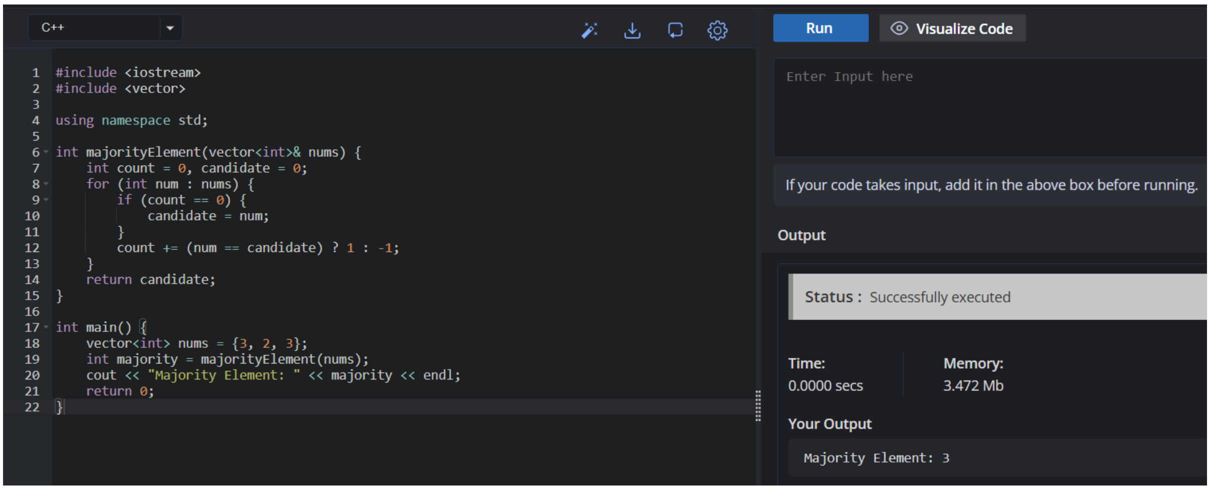
Follow-up: Could you solve the problem in linear time and in O(1) space?

Code:

```
#include <iostream>
#include <vector>

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> nums = {3, 2, 3};
    int majority = majorityElement(nums);
    cout << "Majority Element: " << majority << endl;
    return 0;
}
```

Output:



```
#include <iostream>
#include <vector>

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> nums = {3, 2, 3};
    int majority = majorityElement(nums);
    cout << "Majority Element: " << majority << endl;
    return 0;
}
```

Enter Input here

If your code takes input, add it in the above box before running.

Status : Successfully executed

Time: 0.0000 secs

Memory: 3.472 Mb

Your Output

```
Majority Element: 3
```

Question 2. Single Number

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.
You must implement a solution with a linear runtime complexity and use only constant extra space.

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

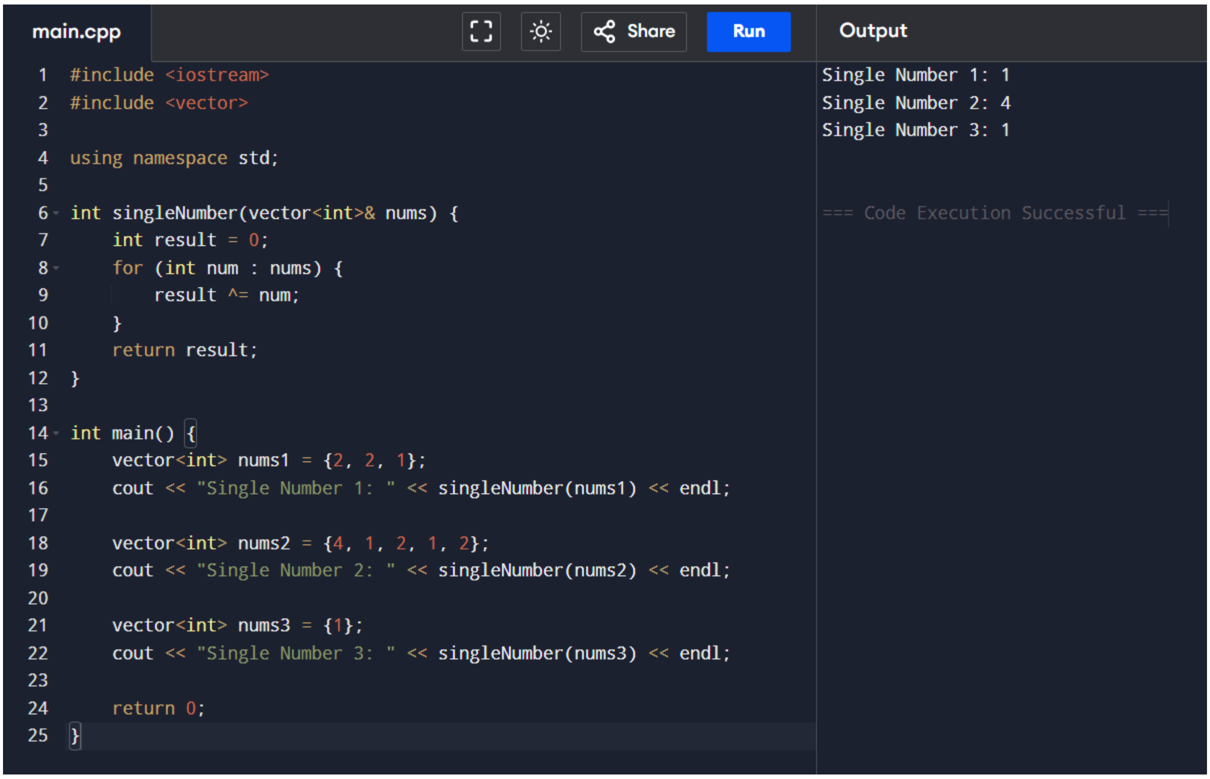
Example 2:
Input: nums = [4,1,2,1,2]
Output: 4

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

Code:

```
#include <iostream>
#include <vector>

using namespace std;

int singleNumber(vector<int>& nums) {
    int result = 0;
    for (int num : nums) {
        result ^= num;
    }
    return result;
}

int main() {
    vector<int> nums1 = {2, 2, 1};
    cout << "Single Number 1: " << singleNumber(nums1)  << endl;

    vector<int> nums2 = {4, 1, 2, 1, 2};
    cout << "Single Number 2: " << singleNumber(nums2)  << endl;

    vector<int> nums3 = {1};
    cout << "Single Number 3: " << singleNumber(nums3)  << endl;

    return 0;
}
```

Output:



```
#include <iostream>
#include <vector>

using namespace std;

int singleNumber(vector<int>& nums) {
    int result = 0;
    for (int num : nums) {
        result ^= num;
    }
    return result;
}

int main() {
    vector<int> nums1 = {2, 2, 1};
    cout << "Single Number 1: " << singleNumber(nums1) << endl;

    vector<int> nums2 = {4, 1, 2, 1, 2};
    cout << "Single Number 2: " << singleNumber(nums2) << endl;

    vector<int> nums3 = {1};
    cout << "Single Number 3: " << singleNumber(nums3) << endl;

    return 0;
}
```

```
Single Number 1: 1
Single Number 2: 4
Single Number 3: 1


=== Code Execution Successful ===
```

Question 3 Convert Sorted Array to Binary Search Tree

Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

Example 1:

Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:

Example 2:

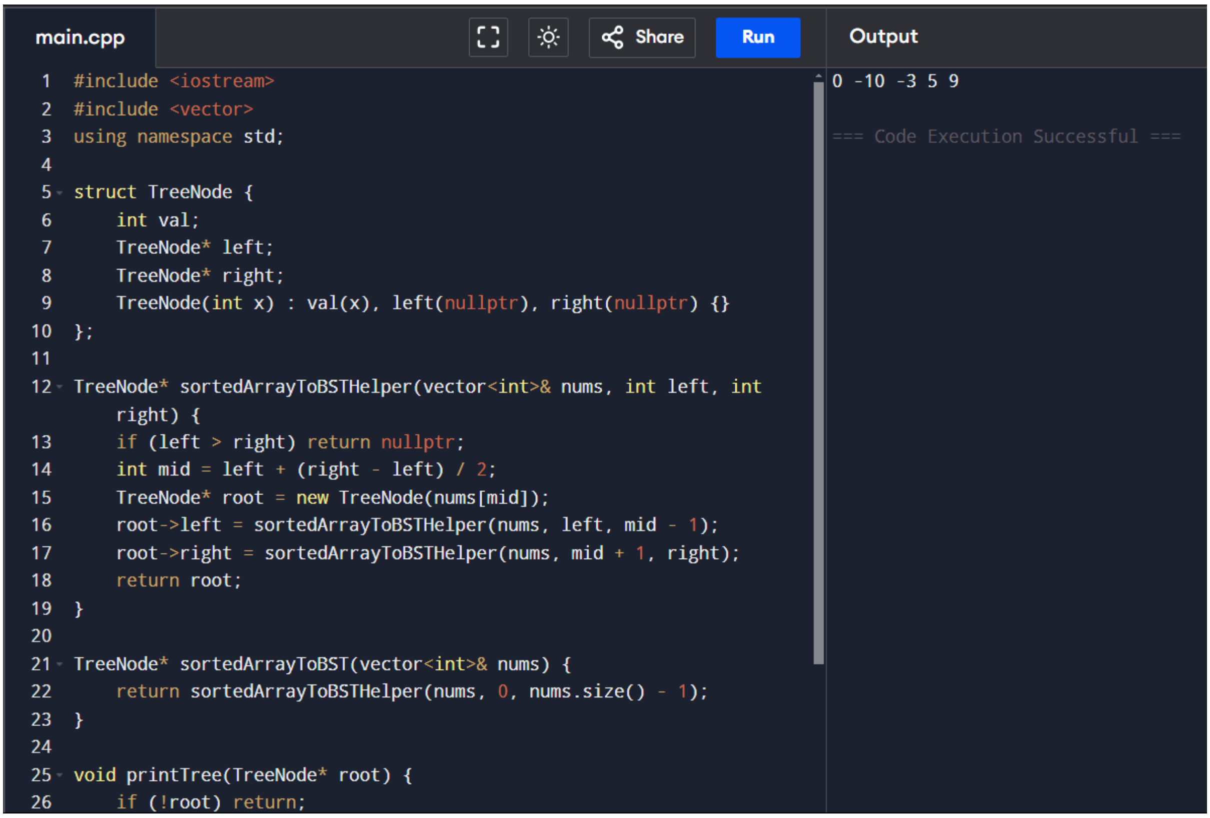
Input: nums = [1,3]
Output: [3,1]
Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

Code:

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

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* sortedArrayToBSTHelper(vector<int>& nums, int left, int right) {
   if (left > right) return nullptr;
   int mid = left + (right - left) / 2;
   TreeNode* root = new TreeNode(nums[mid]);
   root->left = sortedArrayToBSTHelper(nums, left, mid - 1);
   root->right = sortedArrayToBSTHelper(nums, mid + 1, right);
   return root;
}

TreeNode* sortedArrayToBST(vector<int>& nums) {
   return sortedArrayToBSTHelper(nums, 0, nums.size() - 1);
}

void printTree(TreeNode* root) {
   if (!root) return;
   cout << root->val << " ";
   printTree(root->left);
   printTree(root->right);
}
```

```
int main() {
    vector<int> nums = {-10, -3, 0, 5, 9};
    TreeNode* root = sortedArrayToBST(nums);
    printTree(root);
    return 0;
}
```

Output:



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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* sortedArrayToBSTHelper(vector<int>& nums, int left, int right) {
    if (left > right) return nullptr;
    int mid = left + (right - left) / 2;
    TreeNode* root = new TreeNode(nums[mid]);
    root->left = sortedArrayToBSTHelper(nums, left, mid - 1);
    root->right = sortedArrayToBSTHelper(nums, mid + 1, right);
    return root;
}

TreeNode* sortedArrayToBST(vector<int>& nums) {
    return sortedArrayToBSTHelper(nums, 0, nums.size() - 1);
}

void printTree(TreeNode* root) {
    if (!root) return;
```

```
0 -10 -3 5 9

=== Code Execution Successful ===
```

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]]

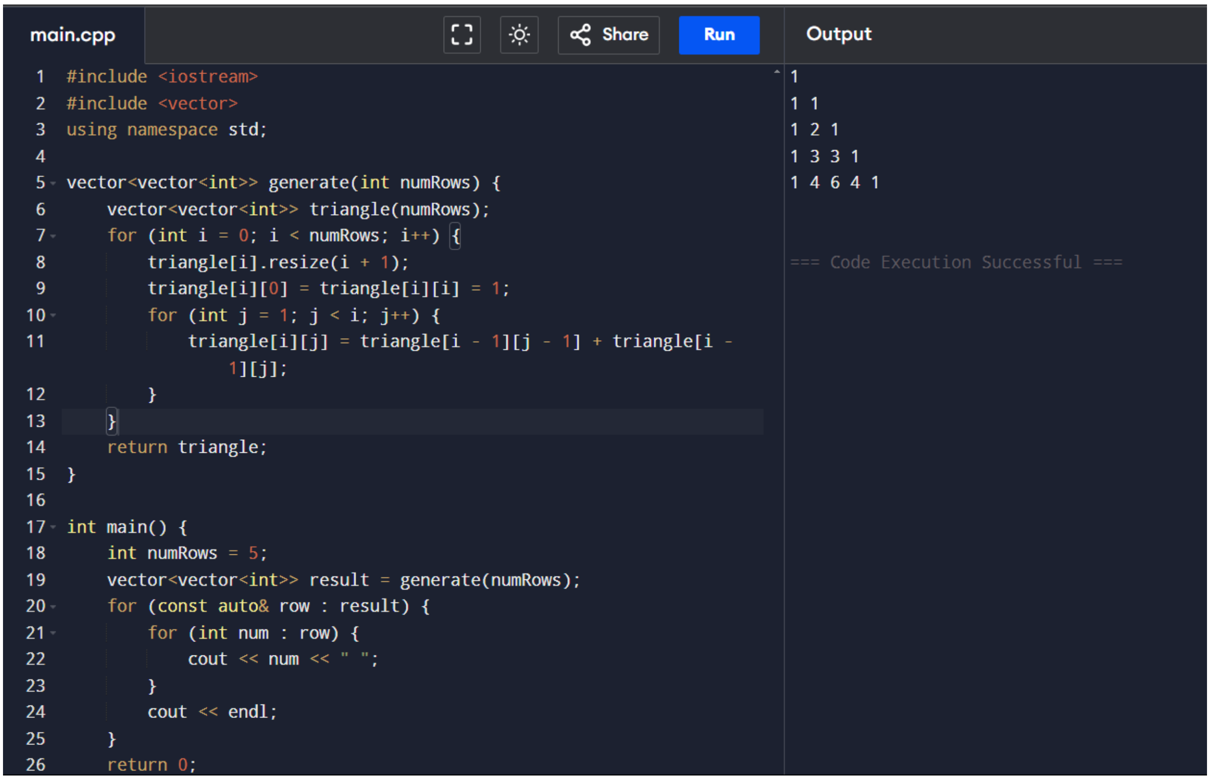
Example 2:

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

Code:

```
#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 (const auto& row : result) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:



```cpp
#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 (const auto& row : result) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }
    return 0;
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1


=== Code Execution Successful ===
```

Question 2. Remove Element

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
Return k.

Example 1:

Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).

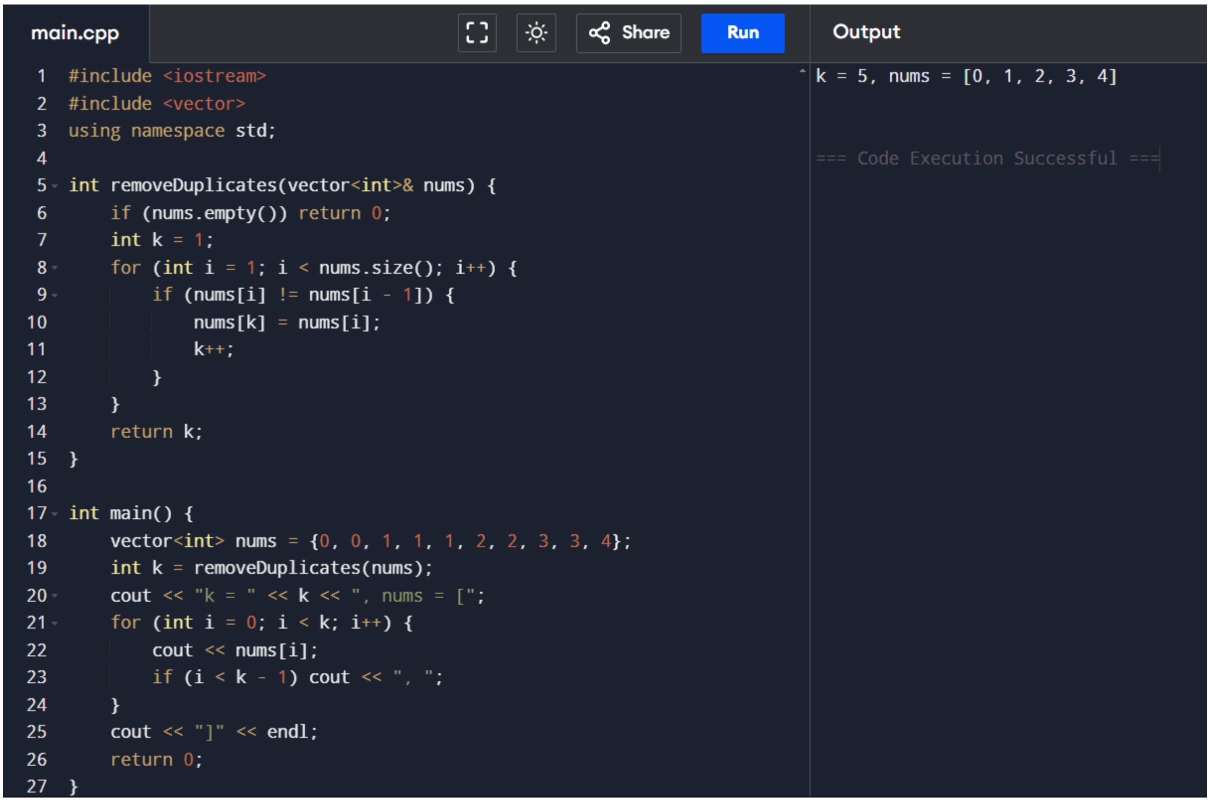
Example 2:

Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).

Code:

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

int removeDuplicates(vector<int>& nums) {
   if (nums.empty()) return 0;
   int k = 1;
   for (int i = 1; i < nums.size(); i++) {
      if (nums[i] != nums[i - 1]) {
         nums[k] = nums[i];
         k++;
      }
   }
   return k;
}

int main() {
   vector<int> nums = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};
   int k = removeDuplicates(nums);
   cout << "k = " << k << ", nums = [";
   for (int i = 0; i < k; i++) {
      cout << nums[i];
      if (i < k - 1) cout << ", ";
   }
   cout << "]" << endl;
   return 0;
}
```

Output



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

int removeDuplicates(vector<int>& nums) {
    if (nums.empty()) return 0;
    int k = 1;
    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] != nums[i - 1]) {
            nums[k] = nums[i];
            k++;
        }
    }
    return k;
}

int main() {
    vector<int> nums = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};
    int k = removeDuplicates(nums);
    cout << "k = " << k << ", nums = [";
    for (int i = 0; i < k; i++) {
        cout << nums[i];
        if (i < k - 1) cout << ", ";
    }
    cout << "]" << endl;
    return 0;
}
```

```
k = 5, nums = [0, 1, 2, 3, 4]


=== Code Execution Successful ===
```

Question 3 Baseball Game :

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:

An integer x.
Record a new score of x.
'+'.
Record a new score that is the sum of the previous two scores.
'D'.
Record a new score that is the double of the previous score.
'C'.
Invalidate the previous score, removing it from the record.
Return the sum of all the scores on the record after applying all the operations.

The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.

Example 1:

Input: ops = ["5","2","C","D","+"]
Output: 30
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.

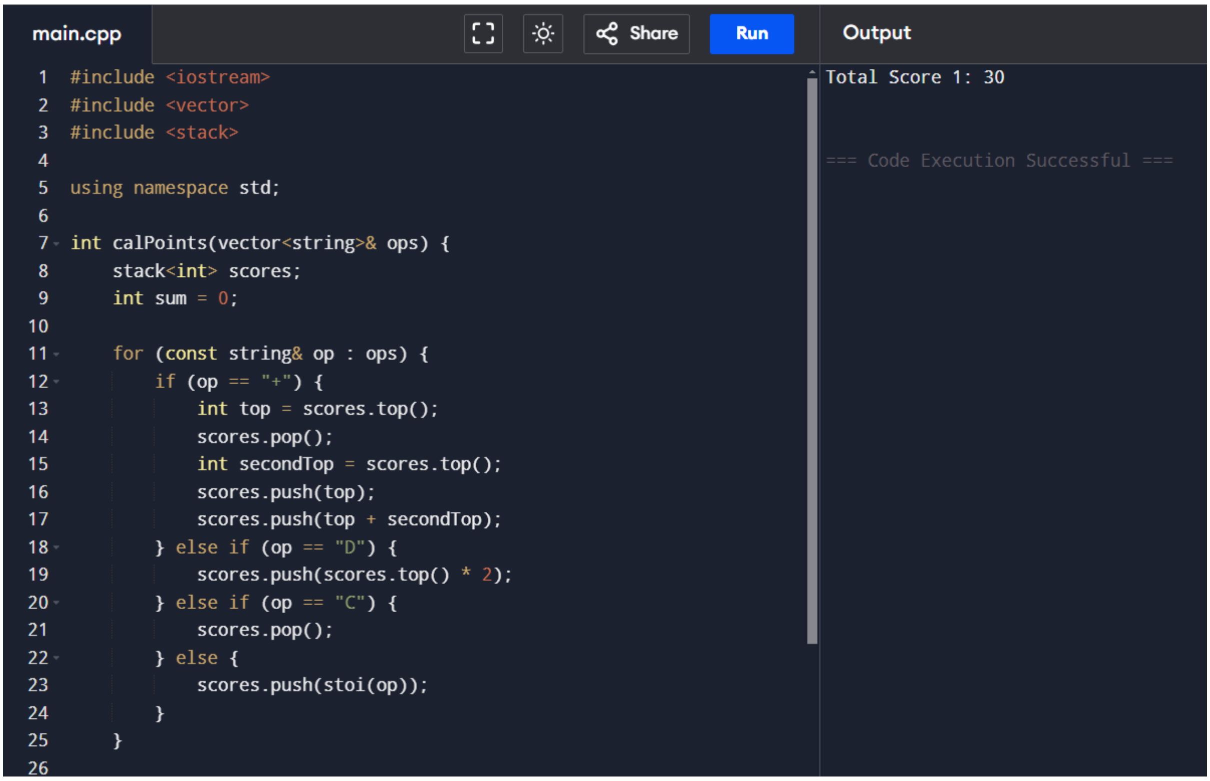
Example 2:

Input: ops = ["5","-2","4","C","D","9","+","+"]
Output: 27
Explanation:
"5" - Add 5 to the record, record is now [5].
"-2" - Add -2 to the record, record is now [5, -2].
"4" - Add 4 to the record, record is now [5, -2, 4].
"C" - Invalidate and remove the previous score, record is now [5, -2].
"D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].
"9" - Add 9 to the record, record is now [5, -2, -4, 9].
"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].
"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].
The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

Code:

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

int calPoints(vector<string>& ops) {
    stack<int> scores;
    int sum = 0;

    for (const string& op : ops) {
        if (op == "+") {
            int top = scores.top();
            scores.pop();
            int secondTop = scores.top();
            scores.push(top);
            scores.push(top + secondTop);
        } else if (op == "D") {
            scores.push(scores.top() * 2);
        } else if (op == "C") {
            scores.pop();
        } else {
            scores.push(stoi(op));
        }
    }

    while (!scores.empty()) {
        sum += scores.top();
        scores.pop();
    }

    return sum;
}

int main() {
    vector<string> ops1 = {"5", "2", "C", "D", "+"};
    cout << "Total Score 1: " << calPoints(ops1) << endl;

    return 0;
}
```

Output:



```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

int calPoints(vector<string>& ops) {
    stack<int> scores;
    int sum = 0;

    for (const string& op : ops) {
        if (op == "+") {
            int top = scores.top();
            scores.pop();
            int secondTop = scores.top();
            scores.push(top);
            scores.push(top + secondTop);
        } else if (op == "D") {
            scores.push(scores.top() * 2);
        } else if (op == "C") {
            scores.pop();
        } else {
            scores.push(stoi(op));
        }
    }

```

```
Total Score 1: 30


=== Code Execution Successful ===
```

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:

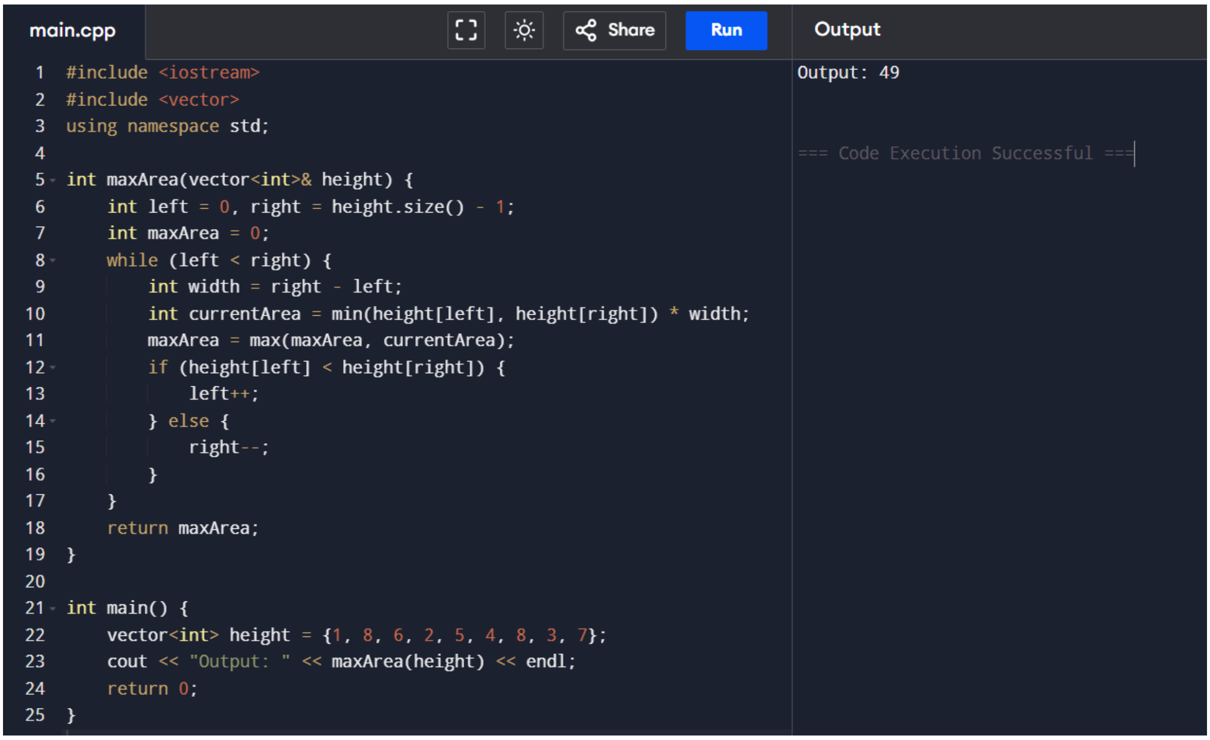
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Code:

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

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxArea = 0;
    while (left < right) {
        int width = right - left;
        int currentArea = min(height[left], height[right]) * width;
        maxArea = max(maxArea, currentArea);
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return maxArea;
}

int main() {
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << "Output: " << maxArea(height) << endl;
    return 0;
```

}

Output:



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

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxArea = 0;
    while (left < right) {
        int width = right - left;
        int currentArea = min(height[left], height[right]) * width;
        maxArea = max(maxArea, currentArea);
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return maxArea;
}

int main() {
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << "Output: " << maxArea(height) << endl;
    return 0;
}
```

```
Output: 49


=== Code Execution Successful ===
```

Question 2. Valid Sudoku

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

Each row must contain the digits 1-9 without repetition.
Each column must contain the digits 1-9 without repetition.
Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.
Note:

A Sudoku board (partially filled) could be valid but is not necessarily solvable.
Only the filled cells need to be validated according to the mentioned rules.

Example 1:

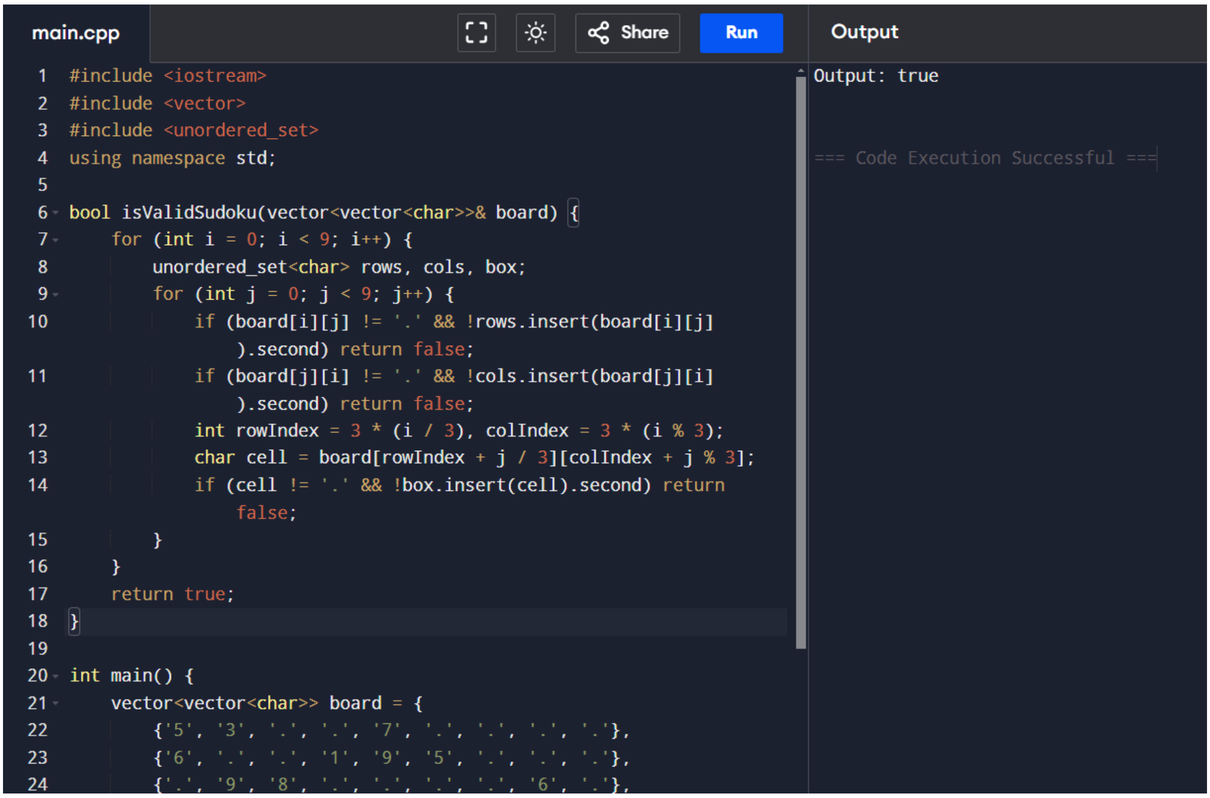
Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]

```
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: true
```

Code:

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

bool isValidSudoku(vector<vector<char>>& board) {
    for (int i = 0; i < 9; i++) {
        unordered_set<char> rows, cols, box;
        for (int j = 0; j < 9; j++) {
            if (board[i][j] != '.' && !rows.insert(board[i][j]).second) return false;
            if (board[j][i] != '.' && !cols.insert(board[j][i]).second) return false;
            int rowIndex = 3 * (i / 3), colIndex = 3 * (i % 3);
            char cell = board[rowIndex + j / 3][colIndex + j % 3];
            if (cell != '.' && !box.insert(cell).second) return false;
        }
    }
    return true;
}

int main() {
    vector<vector<char>> board = {
        {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
        {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
        {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
        {'8', '.', '.', '.', '6', '.', '.', '.', '3'},
        {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
        {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
        {'.', '6', '.', '.', '.', '.', '2', '8', '.'},
        {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
        {'.', '.', '.', '.', '8', '.', '.', '7', '9'}
    };
    cout << "Output: " << (isValidSudoku(board) ? "true" : "false") << endl;
    return 0;
}
```

Output:



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

bool isValidSudoku(vector<vector<char>>& board) {
    for (int i = 0; i < 9; i++) {
        unordered_set<char> rows, cols, box;
        for (int j = 0; j < 9; j++) {
            if (board[i][j] != '.' && !rows.insert(board[i][j]).second) return false;
            if (board[j][i] != '.' && !cols.insert(board[j][i]).second) return false;
            int rowIndex = 3 * (i / 3), colIndex = 3 * (i % 3);
            char cell = board[rowIndex + j / 3][colIndex + j % 3];
            if (cell != '.' && !box.insert(cell).second) return false;
        }
    }
    return true;
}

int main() {
    vector<vector<char>> board = {
        {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
        {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
        {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
```

```
Output: true


=== Code Execution Successful ===
```

Question 3 : Jump Game II

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

0 <= j <= nums[i] and
i + j < n
Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Example 1:

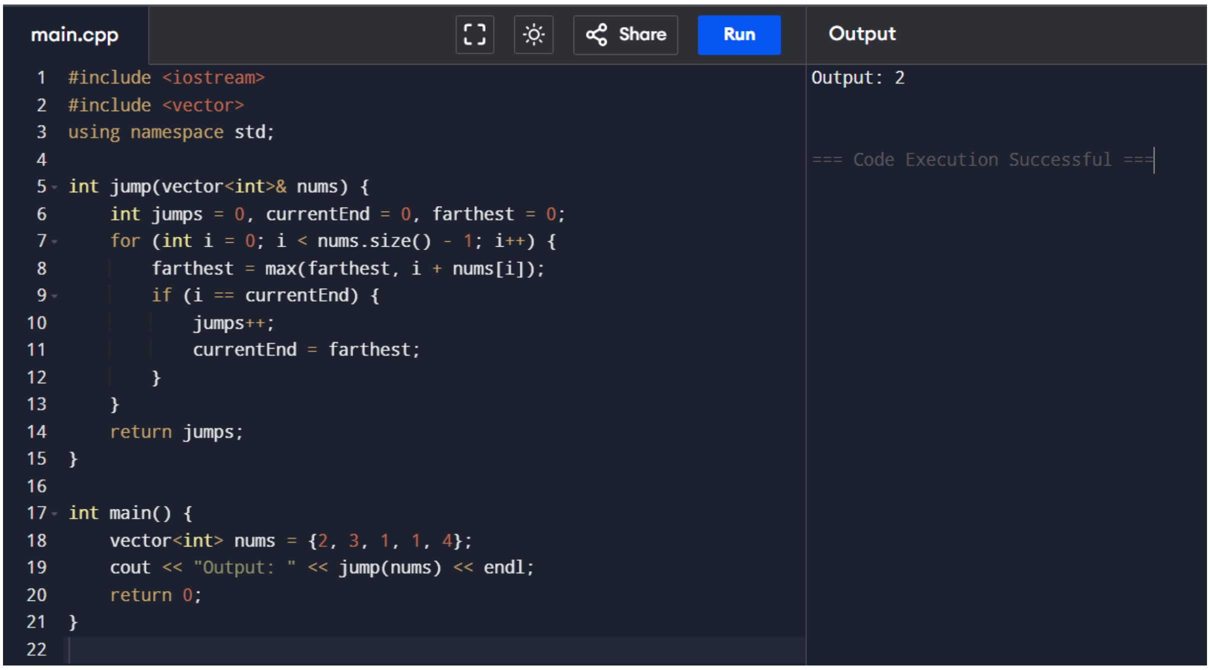
Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Code:

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

```
int jump(vector<int>& nums) {
    int jumps = 0, currentEnd = 0, farthest = 0;
    for (int i = 0; i < nums.size()  - 1; i++) {
        farthest = max(farthest, i + nums[i]);
        if (i == currentEnd) {
            jumps++;
            currentEnd = farthest;
        }
    }
    return jumps;
}

int main() {
    vector<int> nums = {2, 3, 1, 1, 4};
    cout << "Output: " << jump(nums) << endl;
    return 0;
}
```

Output:



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

int jump(vector<int>& nums) {
    int jumps = 0, currentEnd = 0, farthest = 0;
    for (int i = 0; i < nums.size() - 1; i++) {
        farthest = max(farthest, i + nums[i]);
        if (i == currentEnd) {
            jumps++;
            currentEnd = farthest;
        }
    }
    return jumps;
}

int main() {
    vector<int> nums = {2, 3, 1, 1, 4};
    cout << "Output: " << jump(nums) << endl;
    return 0;
}
```

```
Output: 2


=== Code Execution Successful ===
```

Hard

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.

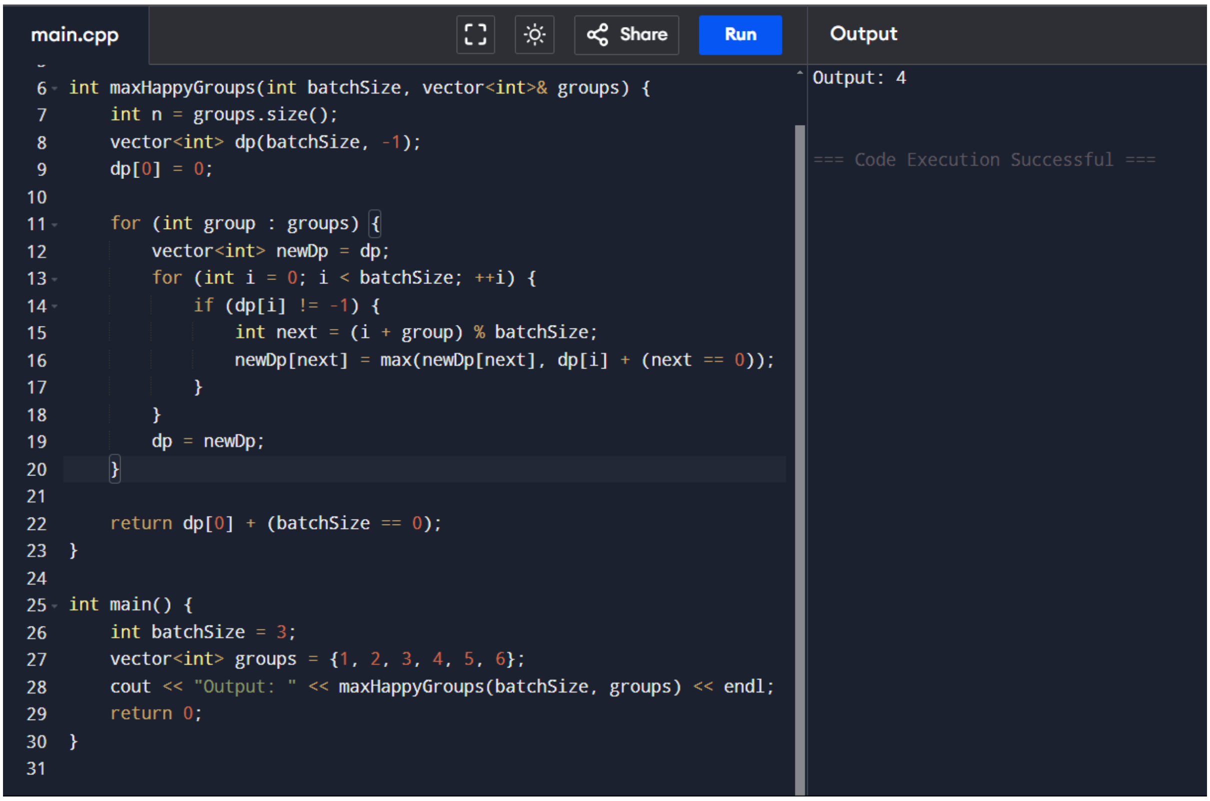
Example 1:

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.

Code:

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

int maxHappyGroups(int batchSize, vector<int>& groups) {
   int n = groups.size();
   vector<int> dp(batchSize, -1);
   dp[0] = 0;

   for (int group : groups) {
      vector<int> newDp = dp;
      for (int i = 0; i < batchSize; ++i) {
         if (dp[i] != -1) {
            int next = (i + group) % batchSize;
            newDp[next] = max(newDp[next], dp[i] + (next == 0));
         }
      }
      dp = newDp;
   }

   return dp[0] + (batchSize == 0);
}

int main() {
    int batchSize = 3;
    vector<int> groups = {1, 2, 3, 4, 5, 6};
    cout << "Output: " << maxHappyGroups(batchSize, groups) << endl;
    return 0;
}
```

Output:



```
int maxHappyGroups(int batchSize, vector<int>& groups) {
    int n = groups.size();
    vector<int> dp(batchSize, -1);
    dp[0] = 0;

    for (int group : groups) {
        vector<int> newDp = dp;
        for (int i = 0; i < batchSize; ++i) {
            if (dp[i] != -1) {
                int next = (i + group) % batchSize;
                newDp[next] = max(newDp[next], dp[i] + (next == 0));
            }
        }
        dp = newDp;
    }

    return dp[0] + (batchSize == 0);
}

int main() {
    int batchSize = 3;
    vector<int> groups = {1, 2, 3, 4, 5, 6};
    cout << "Output: " << maxHappyGroups(batchSize, groups) << endl;
    return 0;
}
```

```
Output: 4


=== Code Execution Successful ===
```

Question 2 Cherry Pickup II

You are given a rows x cols matrix grid representing a field of cherries where grid[i][j] represents the number of cherries that you can collect from the (i, j) cell.

You have two robots that can collect cherries for you:

Robot #1 is located at the top-left corner (0, 0), and
Robot #2 is located at the top-right corner (0, cols - 1).
Return the maximum number of cherries collection using both robots by following the rules below:

From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j + 1).
When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.

When both robots stay in the same cell, only one takes the cherries.
Both robots cannot move outside of the grid at any moment.
Both robots should reach the bottom row in grid.

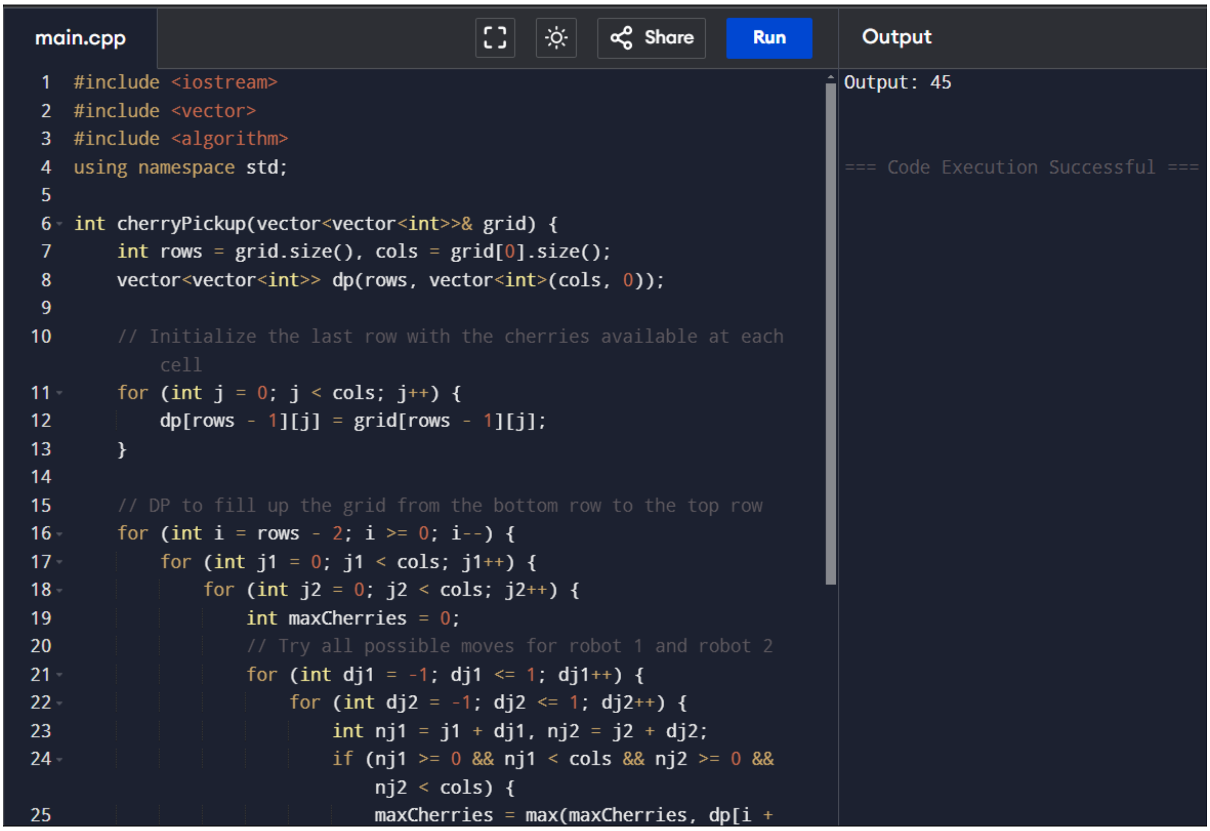
Example 1:

Input: grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]
Output: 24
Explanation: Path of robot #1 and #2 are described in color green and blue respectively.
Cherries taken by Robot #1, (3 + 2 + 5 + 2) = 12.
Cherries taken by Robot #2, (1 + 5 + 5 + 1) = 12.
Total of cherries: 12 + 12 = 24.

Code:

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

int cherryPickup(vector<vector<int>>& grid) {
    int rows = grid.size(), cols = grid[0].size();
    vector<vector<int>> dp(rows, vector<int>(cols, 0));

    // Initialize the last row with the cherries available at each cell
    for (int j = 0; j < cols; j++) {
        dp[rows - 1][j] = grid[rows - 1][j];
    }

    // DP to fill up the grid from the bottom row to the top row
    for (int i = rows - 2; i >= 0; i--) {
        for (int j1 = 0; j1 < cols; j1++) {
            for (int j2 = 0; j2 < cols; j2++) {
                int maxCherries = 0;
                // Try all possible moves for robot 1 and robot 2
                for (int dj1 = -1; dj1 <= 1; dj1++) {
                    for (int dj2 = -1; dj2 <= 1; dj2++) {
                        int nj1 = j1 + dj1, nj2 = j2 + dj2;
                        if (nj1 >= 0 && nj1 < cols && nj2 >= 0 && nj2 < cols) {
                            maxCherries = max(maxCherries, dp[i + 1][nj1] + dp[i + 1][nj2]);
                        }
                    }
                }
                dp[i][j1] = maxCherries + grid[i][j1];
            }
        }
    }
    return dp[0][0];  // Return the maximum cherries at the starting position
}

int main() {
    vector<vector<int>> grid = {{3,1,1},{2,5,1},{1,5,5},{2,1,1}};
    cout << "Output: " << cherryPickup(grid) << endl;
    return 0;
}
```

Output:



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

int cherryPickup(vector<vector<int>>& grid) {
    int rows = grid.size(), cols = grid[0].size();
    vector<vector<int>> dp(rows, vector<int>(cols, 0));

    // Initialize the last row with the cherries available at each
        cell
    for (int j = 0; j < cols; j++) {
        dp[rows - 1][j] = grid[rows - 1][j];
    }

    // DP to fill up the grid from the bottom row to the top row
    for (int i = rows - 2; i >= 0; i--) {
        for (int j1 = 0; j1 < cols; j1++) {
            for (int j2 = 0; j2 < cols; j2++) {
                int maxCherries = 0;
                // Try all possible moves for robot 1 and robot 2
                for (int dj1 = -1; dj1 <= 1; dj1++) {
                    for (int dj2 = -1; dj2 <= 1; dj2++) {
                        int nj1 = j1 + dj1, nj2 = j2 + dj2;
                        if (nj1 >= 0 && nj1 < cols && nj2 >= 0 &&
                            nj2 < cols) {
                            maxCherries = max(maxCherries, dp[i +
```

```
Output: 45


=== Code Execution Successful ===
```

Question 3: Maximum Number of Darts Inside of a Circular Dartboard

Alice is throwing n darts on a very large wall. You are given an array darts where darts[i] = [xi, yi] is the position of the ith dart that Alice threw on the wall.

Bob knows the positions of the n darts on the wall. He wants to place a dartboard of radius r on the wall so that the maximum number of darts that Alice throws lie on the dartboard.

Given the integer r, return the maximum number of darts that can lie on the dartboard.

Example 1:

Input: darts = [[-2,0],[2,0],[0,2],[0,-2]], r = 2
Output: 4

Explanation: Circle dartboard with center in (0,0) and radius = 2 contain all points.

Example 2:

Input: darts = [[-3,0],[3,0],[2,6],[5,4],[0,9],[7,8]], r = 5
Output: 5
Explanation: Circle dartboard with center in (0,4) and radius = 5 contain all points except the point (7,8).

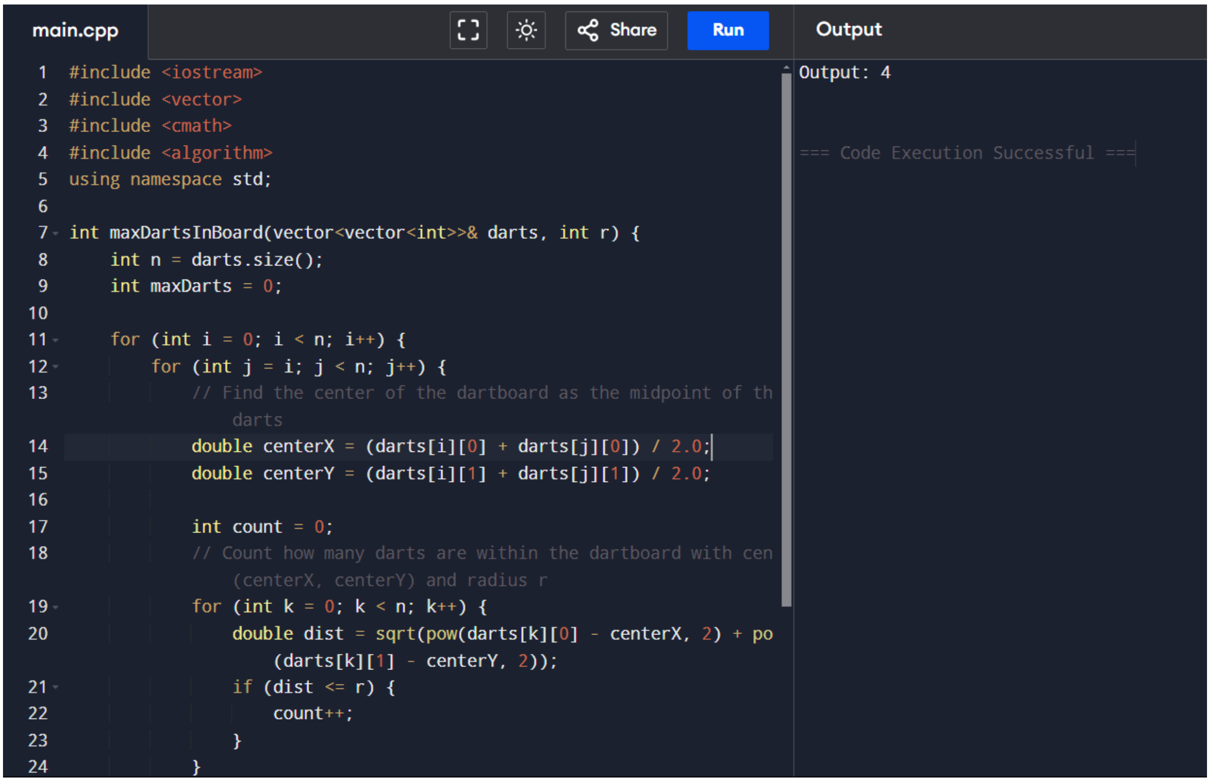
Constraints:

1 <= darts.length <= 100
darts[i].length == 2
-104 <= xi, yi <= 104
All the darts are unique
1 <= r <= 5000

Code:

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

int maxDartsInBoard(vector<vector<int>>& darts, int r) {
    int n = darts.size();
    int maxDarts = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            // Find the center of the dartboard as the midpoint of the two darts
            double centerX = (darts[i][0] + darts[j][0]) / 2.0;
            double centerY = (darts[i][1] + darts[j][1]) / 2.0;

            int count = 0;
            // Count how many darts are within the dartboard with center (centerX, centerY) and
radius r
            for (int k = 0; k < n; k++) {
                double dist = sqrt(pow(darts[k][0] - centerX, 2) + pow(darts[k][1] - centerY, 2));
                if (dist <= r) {
                    count++;
                }
            }
            maxDarts = max(maxDarts, count);
        }
    }

    return maxDarts;
}

int main() {
    vector<vector<int>> darts = {{-2, 0}, {2, 0}, {0, 2}, {0, -2}};
    int r = 2;
    cout << "Output: " << maxDartsInBoard(darts, r) << endl;
    return 0;
}
```

Output



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

int maxDartsInBoard(vector<vector<int>>& darts, int r) {
    int n = darts.size();
    int maxDarts = 0;

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            // Find the center of the dartboard as the midpoint of th
                darts
            double centerX = (darts[i][0] + darts[j][0]) / 2.0;
            double centerY = (darts[i][1] + darts[j][1]) / 2.0;

            int count = 0;
            // Count how many darts are within the dartboard with cen
                (centerX, centerY) and radius r
            for (int k = 0; k < n; k++) {
                double dist = sqrt(pow(darts[k][0] - centerX, 2) + po
                    (darts[k][1] - centerY, 2));
                if (dist <= r) {
                    count++;
                }
            }
```

```
Output: 4

=== Code Execution Successful ===
```

Very Hard

Question 1. Find Minimum Time to Finish All Jobs

You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the ith job.There are k workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized.

Return the minimum possible maximum working time of any assignment.

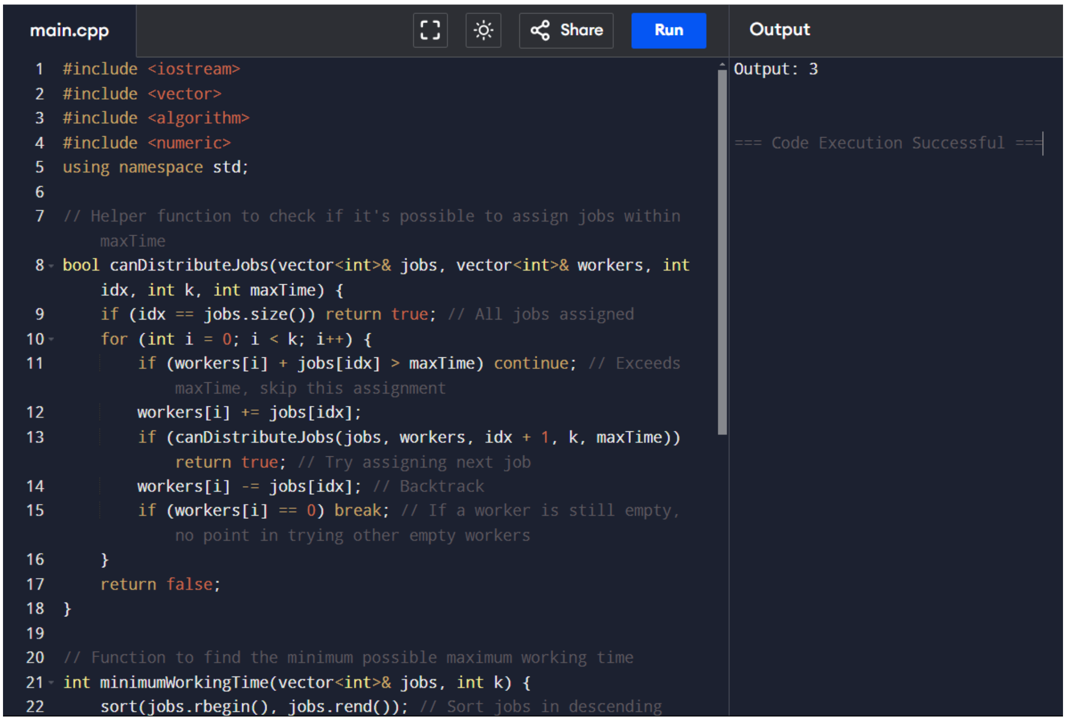
Example 1:

Input: jobs = [3,2,3], k = 3
Output: 3
Explanation: By assigning each person one job, the maximum time is 3.

Code:

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

bool canDistributeJobs(vector<int>& jobs, vector<int>& workers, int idx, int k, int
maxTime) {
    if (idx == jobs.size()) return true; // All jobs assigned
    for (int i = 0; i < k; i++) {
        if (workers[i] + jobs[idx] > maxTime) continue; // Exceeds maxTime, skip this
assignment
        workers[i] += jobs[idx];
        if (canDistributeJobs(jobs, workers, idx + 1, k, maxTime)) return true; // Try assigning
next job
        workers[i] -= jobs[idx]; // Backtrack
        if (workers[i] == 0) break; // If a worker is still empty, no point in trying other empty
workers
    }
    return false;
}

int minimumWorkingTime(vector<int>& jobs, int k) {
    sort(jobs.rbegin(), jobs.rend());
    int left = jobs[0];
    int right = accumulate(jobs.begin(), jobs.end(), 0;
    int result = right;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        vector<int> workers(k, 0); // Workers' workload
        if (canDistributeJobs(jobs, workers, 0, k, mid)) {
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return result;
}

int main() {
    vector<int> jobs = {3, 2, 3};
    int k = 3;
    cout << "Output: " << minimumWorkingTime(jobs,  k) << endl;
    return 0;
}
```

Output:



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

// Helper function to check if it's possible to assign jobs within maxTime
bool canDistributeJobs(vector<int>& jobs, vector<int>& workers, int idx, int k, int maxTime) {
    if (idx == jobs.size()) return true; // All jobs assigned
    for (int i = 0; i < k; i++) {
        if (workers[i] + jobs[idx] > maxTime) continue; // Exceeds maxTime, skip this assignment
        workers[i] += jobs[idx];
        if (canDistributeJobs(jobs, workers, idx + 1, k, maxTime)) return true; // Try assigning next job
        workers[i] -= jobs[idx]; // Backtrack
        if (workers[i] == 0) break; // If a worker is still empty, no point in trying other empty workers
    }
    return false;
}

// Function to find the minimum possible maximum working time
int minimumWorkingTime(vector<int>& jobs, int k) {
    sort(jobs.rbegin(), jobs.rend()); // Sort jobs in descending
```

```
Output: 3


=== Code Execution Successful ===
```

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 ith user knows, and
friendships[i] = [ui, vi] denotes a friendship between the users ui and vi.
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.

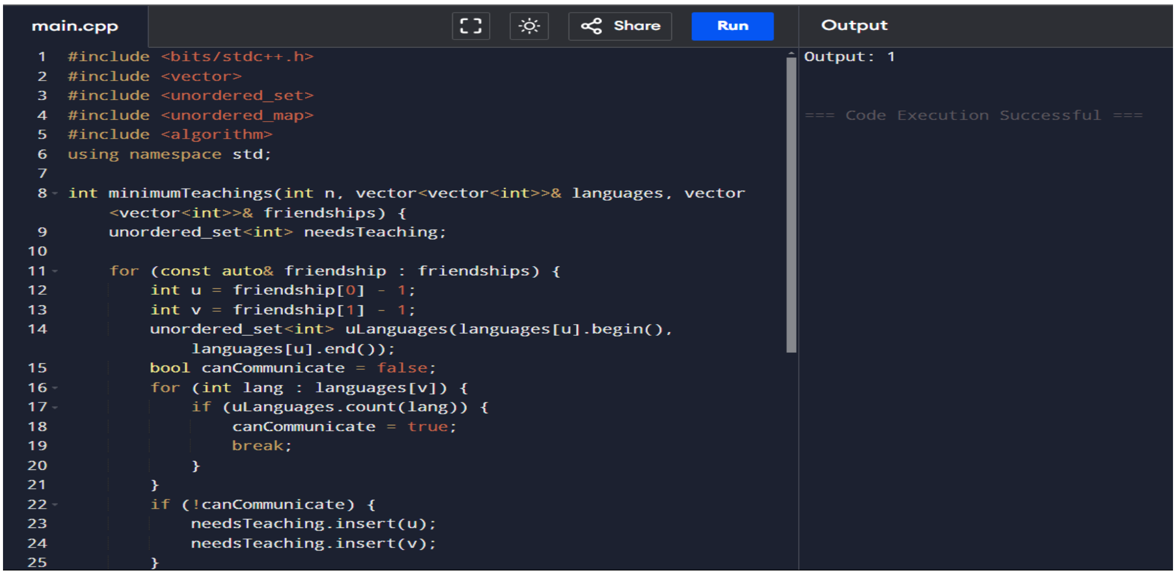
Constraints:

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

Code:

```
#include <bits/stdc++.h>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <algorithm>
using namespace std;

int minimumTeachings(int n, vector<vector<int>>& languages, vector<vector<int>>&
friendships) {
    unordered_set<int> needsTeaching;

    for (const auto& friendship : friendships) {
        int u = friendship[0] - 1;
        int v = friendship[1] - 1;
        unordered_set<int> uLanguages(languages[u].begin(), languages[u].end());
        bool canCommunicate = false;
        for (int lang : languages[v]) {
            if (uLanguages.count(lang)) {
                canCommunicate = true;
                break;
            }
        }
        if (!canCommunicate) {
            needsTeaching.insert(u);
            needsTeaching.insert(v);
        }
    }

    int minTeach = INT_MAX;

    for (int lang = 1; lang <= n; lang++) {
        int teachCount = 0;
        for (int user : needsTeaching) {
            if (find(languages[user].begin(), languages[user].end(), lang) ==
languages[user].end()) {
                teachCount++;
            }
        }
        minTeach = min(minTeach, teachCount);
    }
    return minTeach;
}

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

    cout << "Output: " << minimumTeachings(n, languages, friendships) << endl;
    return 0;
}
```

Output:



```
#include <bits/stdc++.h>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <algorithm>
using namespace std;

int minimumTeachings(int n, vector<vector<int>>& languages, vector
    <vector<int>>& friendships) {
    unordered_set<int> needsTeaching;

    for (const auto& friendship : friendships) {
        int u = friendship[0] - 1;
        int v = friendship[1] - 1;
        unordered_set<int> uLanguages(languages[u].begin(),
            languages[u].end());
        bool canCommunicate = false;
        for (int lang : languages[v]) {
            if (uLanguages.count(lang)) {
                canCommunicate = true;
                break;
            }
        }
        if (!canCommunicate) {
            needsTeaching.insert(u);
            needsTeaching.insert(v);
        }
```

```
Output: 1


=== Code Execution Successful ===
```

Question 3 Count Ways to Make Array With Product

You are given a 2D integer array, queries. For each queries[i], where queries[i] = [ni, ki], find the number of different ways you can place positive integers into an array of size ni such that the product of the integers is ki. As the number of ways may be too large, the answer to the ith query is the number of ways modulo 109 + 7.

Return an integer array answer where answer.length == queries.length, and answer[i] is the answer to the ith query.

Example 1:

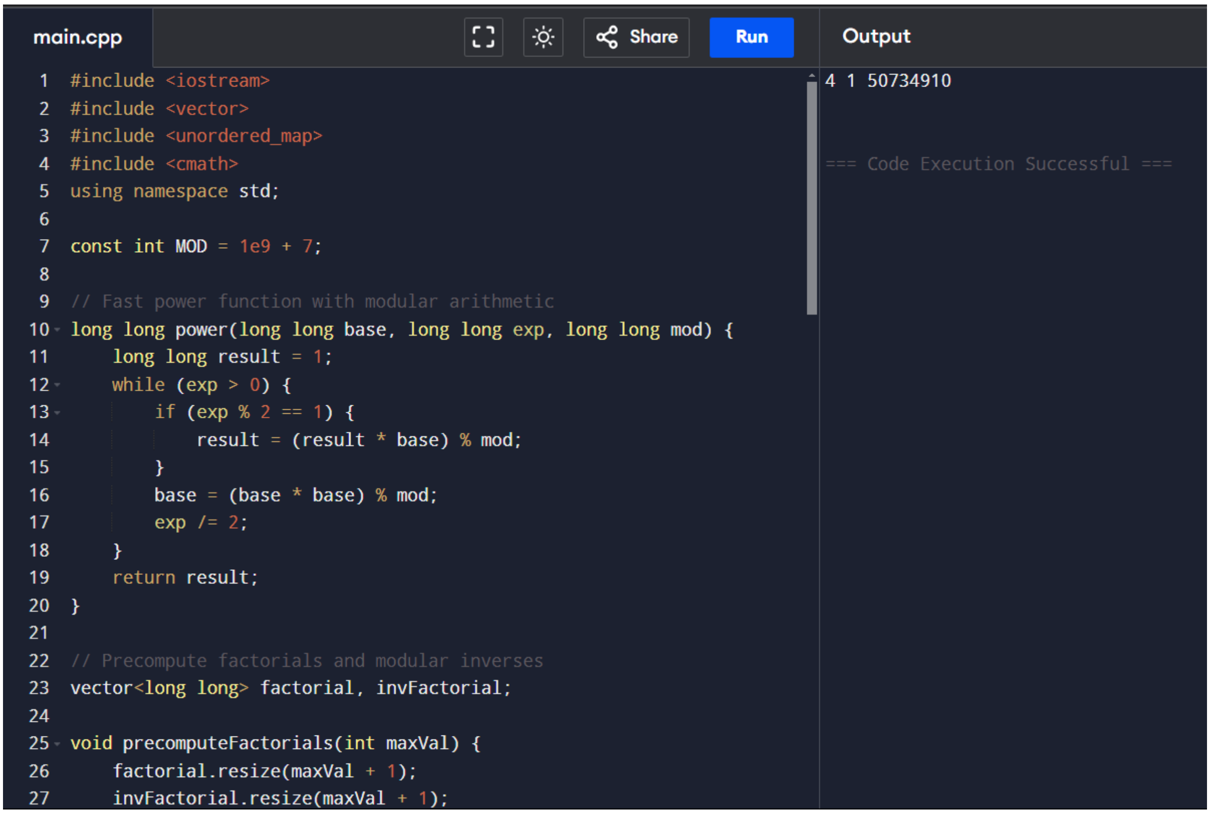
Input: queries = [[2,6],[5,1],[73,660]]
Output: [4,1,50734910]
Explanation: Each query is independent.
[2,6]: There are 4 ways to fill an array of size 2 that multiply to 6: [1,6], [2,3], [3,2], [6,1].
[5,1]: There is 1 way to fill an array of size 5 that multiply to 1: [1,1,1,1,1].
[73,660]: There are 1050734917 ways to fill an array of size 73 that multiply to 660. 1050734917 modulo 109 + 7 = 50734910.

Code:

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

const int MOD = 1e9 + 7;
```

```
// Fast power function with modular arithmetic
long long power(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

// Precompute factorials and modular inverses
vector<long long> factorial, invFactorial;

void precomputeFactorials(int maxVal) {
    factorial.resize(maxVal + 1);
    invFactorial.resize(maxVal + 1);

    factorial[0] = 1;
    for (int i = 1; i <= maxVal; i++) {
        factorial[i] = (factorial[i - 1] * i) % MOD;
    }

    invFactorial[maxVal] = power(factorial[maxVal], MOD - 2, MOD);
    for (int i = maxVal - 1; i >= 0; i--) {
        invFactorial[i] = (invFactorial[i + 1] * (i + 1)) % MOD;
    }
}

// Function to calculate nCr % MOD
long long nCr(int n, int r) {
    if (n < r) return 0;
    return factorial[n] * invFactorial[r] % MOD * invFactorial[n - r] % MOD;
}

// Prime factorization of k
unordered_map<int, int> primeFactorize(int k) {
    unordered_map<int, int> primeFactors;
    for (int i = 2; i * i <= k; i++) {
        while (k % i == 0) {
            primeFactors[i]++;
            k /= i;
        }
    }
    if (k > 1) {
        primeFactors[k]++;
    }
    return primeFactors;
}

// Solve each query
vector<int> waysToFillArray(vector<vector<int>>& queries) {
    int maxN = 0;
    for (const auto& query : queries) {
        maxN = max(maxN, query[0]);
    }

    precomputeFactorials(maxN + 40);

    vector<int> result;
    for (const auto& query : queries) {
        int n = query[0], k = query[1];

        unordered_map<int, int> primeFactors = primeFactorize(k);
        long long ways = 1;

        for (const auto& [prime, power] : primeFactors) {
            ways = (ways * nCr(n + power - 1, power)) % MOD;
        }
        result.push_back(ways);
    }

    return result;
}

int main() {
    vector<vector<int>> queries = {{2, 6}, {5, 1}, {73, 660}};
    vector<int> result = waysToFillArray(queries);

    for (int ans : result) {
        cout << ans << " ";
    }
    cout << endl;
    return 0;
}
```

Output:



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

const int MOD = 1e9 + 7;

// Fast power function with modular arithmetic
long long power(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

// Precompute factorials and modular inverses
vector<long long> factorial, invFactorial;

void precomputeFactorials(int maxVal) {
    factorial.resize(maxVal + 1);
    invFactorial.resize(maxVal + 1);
```

```
4 1 50734910


=== Code Execution Successful ===
```