# PROBLEM STATEMENT - The King’s Feast

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cin >> n; // input number of plates

vector<int> arr(n);

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

cin >> arr[i]; // input food quantities

}

int max\_food = arr[0]; // assume first plate has max food

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

if (arr[i] > max\_food) {

max\_food = arr[i];

}

}

cout << max\_food << endl; // output the maximum food plate

return 0;

}

## PROBLEM STATEMENT - The Lost Soldier

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cin >> n; // total soldiers numbered 0 to n

vector<int> arr(n);

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

cin >> arr[i];

}

// Expected sum of numbers from 0 to n

int total = n \* (n + 1) / 2;

// Actual sum of given soldiers

int sum = 0;

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

sum += arr[i];

}

// Missing soldier number

int missing = total - sum;

cout << missing << endl;

return 0;

}

# PROBLEM STATEMENT - Potion Mixing (Two Sum)

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

int main() {

int n;

cin >> n; // number of potions

vector<int> arr(n);

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

cin >> arr[i];

}

int target;

cin >> target;

unordered\_map<int, int> mp; // {potion\_strength, index}

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

int needed = target - arr[i];

if (mp.find(needed) != mp.end()) {

// Found the two potions

cout << "Indices (" << mp[needed] + 1 << "," << i + 1 << ")" << endl;

return 0;

}

mp[arr[i]] = i;

}

cout << "No such potions found" << endl;

return 0;

}

# PROBLEM STATEMENT - The Secret Message

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int n;

cin >> n; // number of elements

vector<int> arr(n);

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

cin >> arr[i];

}

// Reverse the array

reverse(arr.begin(), arr.end());

// Print reversed array

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

cout << arr[i] << " ";

}

return 0;

}

# PROBLEM STATEMENT - The King’s Parade

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cin >> n; // number of soldiers

vector<int> arr(n);

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

cin >> arr[i];

}

bool sorted = true;

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

if (arr[i] < arr[i - 1]) {

sorted = false;

break;

}

}

if (sorted)

cout << "true" << endl;

else

cout << "false" << endl;

return 0;

}

# PROBLEM STATEMENT - The Treasure Island

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n, m;

cin >> n >> m; // number of rows and columns

vector<vector<int>> grid(n, vector<int>(m));

// Input grid values

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

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

cin >> grid[i][j];

}

}

int maxSum = -1e9; // initialize with a very small number

int rowIndex = -1;

// Find row with maximum gold

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

int rowSum = 0;

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

rowSum += grid[i][j];

}

if (rowSum > maxSum) {

maxSum = rowSum;

rowIndex = i + 1; // store 1-based index

}

}

cout << "Row " << rowIndex << " (sum=" << maxSum << ")" << endl;

return 0;

}

# PROBLEM STATEMENT - The Spiral Library

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n, m;

cin >> n >> m; // number of rows and columns

vector<vector<int>> matrix(n, vector<int>(m));

// Input matrix

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

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

cin >> matrix[i][j];

}

}

vector<int> result;

int top = 0, bottom = n - 1;

int left = 0, right = m - 1;

while (top <= bottom && left <= right) {

// Traverse left → right

for (int i = left; i <= right; i++)

result.push\_back(matrix[top][i]);

top++;

// Traverse top → bottom

for (int i = top; i <= bottom; i++)

result.push\_back(matrix[i][right]);

right--;

// Traverse right → left

if (top <= bottom) {

for (int i = right; i >= left; i--)

result.push\_back(matrix[bottom][i]);

bottom--;

}

// Traverse bottom → top

if (left <= right) {

for (int i = bottom; i >= top; i--)

result.push\_back(matrix[i][left]);

left++;

}

}

// Print spiral order

cout << "[";

for (int i = 0; i < result.size(); i++) {

cout << result[i];

if (i != result.size() - 1) cout << ",";

}

cout << "]" << endl;

return 0;

}

# PROBLEM STATEMENT - The Royal Diagonal

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n, m;

cin >> n >> m; // rows and columns (square matrix expected)

vector<vector<int>> matrix(n, vector<int>(m));

// Input matrix

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

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

cin >> matrix[i][j];

}

}

int primarySum = 0, secondarySum = 0;

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

primarySum += matrix[i][i]; // Primary diagonal

secondarySum += matrix[i][n - i - 1]; // Secondary diagonal

}

cout << "Primary Diagonal Sum = " << primarySum << endl;

cout << "Secondary Diagonal Sum = " << secondarySum << endl;

return 0;

}

# PROBLEM STATEMENT - The Messenger’s Path

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

bool canReach(vector<vector<int>>& grid) {

int n = grid.size();

int m = grid[0].size();

// If start or end is blocked

if (grid[0][0] == 1 || grid[n-1][m-1] == 1)

return false;

vector<vector<bool>> visited(n, vector<bool>(m, false));

queue<pair<int, int>> q;

q.push({0, 0});

visited[0][0] = true;

int dx[] = {1, -1, 0, 0};

int dy[] = {0, 0, 1, -1};

while (!q.empty()) {

auto [x, y] = q.front();

q.pop();

// If we reached destination

if (x == n - 1 && y == m - 1)

return true;

// Explore 4 directions

for (int i = 0; i < 4; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

if (nx >= 0 && ny >= 0 && nx < n && ny < m &&

grid[nx][ny] == 0 && !visited[nx][ny]) {

visited[nx][ny] = true;

q.push({nx, ny});

}

}

}

return false;

}

int main() {

int n, m;

cin >> n >> m;

vector<vector<int>> grid(n, vector<int>(m));

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

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

cin >> grid[i][j];

}

}

if (canReach(grid))

cout << "true" << endl;

else

cout << "false" << endl;

return 0;

}

# PROBLEM STATEMENT - The Rainwater Pond

#include <iostream>

#include <vector>

using namespace std;

int n, m;

// Directions: 8 possible moves (vertical, horizontal, diagonal)

int dx[] = {-1, -1, -1, 0, 0, 1, 1, 1};

int dy[] = {-1, 0, 1, -1, 1, -1, 0, 1};

void dfs(vector<vector<int>>& grid, int x, int y) {

if (x < 0 || y < 0 || x >= n || y >= m || grid[x][y] == 0)

return;

grid[x][y] = 0; // mark as visited

for (int i = 0; i < 8; i++) {

dfs(grid, x + dx[i], y + dy[i]);

}

}

int countPonds(vector<vector<int>>& grid) {

int count = 0;

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

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

if (grid[i][j] == 1) {

count++;

dfs(grid, i, j); // mark all connected water cells

}

}

}

return count;

}

int main() {

cin >> n >> m;

vector<vector<int>> grid(n, vector<int>(m));

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

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

cin >> grid[i][j];

}

}

cout << countPonds(grid) << endl;

return 0;

}

# PROBLEM STATEMENT - Tower of Temples (Hanoi)

#include <iostream>

using namespace std;

int hanoi(int n, char source, char destination, char helper) {

if (n == 0) return 0;

int moves = 0;

// Move n-1 disks from source to helper

moves += hanoi(n - 1, source, helper, destination);

// Move nth disk from source to destination

cout << "Move disk " << n << " from " << source << " to " << destination << endl;

moves++;

// Move n-1 disks from helper to destination

moves += hanoi(n - 1, helper, destination, source);

return moves;

}

int main() {

int n;

cin >> n;

int totalMoves = hanoi(n, 'A', 'C', 'B'); // A = source, C = destination, B = helper

cout << "Total moves: " << totalMoves << endl;

return 0;

}

# PROBLEM STATEMENT - The Magical Staircase

#include <iostream>

using namespace std;

int countWays(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

if (n == 2) return 2;

int a = 1; // ways to reach step 1

int b = 2; // ways to reach step 2

int c;

for (int i = 3; i <= n; i++) {

c = a + b;

a = b;

b = c;

}

return c;

}

int main() {

int n;

cin >> n;

cout << countWays(n) << endl;

return 0;

}

# PROBLEM STATEMENT - The Sorcerer’s Spell

#include <iostream>

using namespace std;

string reverseString(string s) {

if (s.length() <= 1)

return s;

// Reverse the substring excluding first character

return reverseString(s.substr(1)) + s[0];

}

int main() {

string str;

cin >> str;

string reversed = reverseString(str);

cout << reversed << endl;

return 0;

}

# PROBLEM STATEMENT - The Dragon’s Roar

#include <iostream>

using namespace std;

void printNumbers(int n) {

if (n == 0)

return;

printNumbers(n - 1); // print 1 to n-1

cout << n << " ";

}

int main() {

int n;

cin >> n;

printNumbers(n);

cout << endl;

return 0;

}

# PROBLEM STATEMENT - The Hidden Chamber

#include <iostream>

#include <vector>

using namespace std;

int sumArray(vector<int>& arr, int n) {

if (n == 0)

return 0;

return arr[n-1] + sumArray(arr, n-1);

}

int main() {

int n;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

}

int total = sumArray(arr, n);

cout << total << endl;

return 0;

}

# PROBLEM STATEMENT - The Ancient Scroll

#include <iostream>

#include <vector>

using namespace std;

int searchScroll(vector<int>& arr, int key) {

for (int i = 0; i < arr.size(); i++) {

if (arr[i] == key)

return i; // return index if found

}

return -1; // not found

}

int main() {

int n, key;

cin >> n; // number of scrolls

vector<int> arr(n);

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

cin >> arr[i];

cin >> key; // scroll ID to search

int index = searchScroll(arr, key);

cout << index << endl;

return 0;

}

# PROBLEM STATEMENT - The Farmer’s Basket

#include <iostream>

#include <vector>

using namespace std;

int findFruit(vector<int>& arr, int key) {

for (int i = 0; i < arr.size(); i++) {

if (arr[i] == key)

return i; // fruit found

}

return -1; // fruit not found

}

int main() {

int n, key;

cin >> n; // number of fruits

vector<int> arr(n);

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

cin >> arr[i];

cin >> key; // fruit to search

cout << findFruit(arr, key) << endl;

return 0;

}

# PROBLEM STATEMENT - The Secret Door

#include <iostream>

#include <vector>

using namespace std;

int binarySearch(vector<int>& arr, int key) {

int low = 0, high = arr.size() - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == key)

return mid; // found key

else if (arr[mid] < key)

low = mid + 1; // search right half

else

high = mid - 1; // search left half

}

return -1; // key not found

}

int main() {

int n, key;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> key;

int index = binarySearch(arr, key);

cout << index << endl;

return 0;

}

# PROBLEM STATEMENT - The Archer’s Range

#include <iostream>

#include <vector>

using namespace std;

int firstOccurrence(vector<int>& arr, int key) {

int low = 0, high = arr.size() - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == key) {

result = mid; // possible first occurrence

high = mid - 1; // search left half

} else if (arr[mid] < key) {

low = mid + 1; // search right half

} else {

high = mid - 1; // search left half

}

}

return result;

}

int main() {

int n, key;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> key;

cout << firstOccurrence(arr, key) << endl;

return 0;

}

# PROBLEM STATEMENT - The Treasure Chest

#include <iostream>

#include <vector>

using namespace std;

int lastOccurrence(vector<int>& arr, int key) {

int low = 0, high = arr.size() - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == key) {

result = mid; // possible last occurrence

low = mid + 1; // search right half

} else if (arr[mid] < key) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

int main() {

int n, key;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> key;

cout << lastOccurrence(arr, key) << endl;

return 0;

}

# PROBLEM STATEMENT - The first index where the element is greater than or equal to the target

#include <iostream>

#include <vector>

using namespace std;

int lowerBound(vector<int>& arr, int target) {

int low = 0, high = arr.size() - 1;

int n = arr.size();

int result = n; // default if target > all elements

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] >= target) {

result = mid; // possible lower bound

high = mid - 1; // search left half

} else {

low = mid + 1; // search right half

}

}

return result;

}

int main() {

int n, target;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> target;

cout << lowerBound(arr, target) << endl;

return 0;

}

# PROBLEM STATEMENT - The first index where the element is strictly greater than the target

#include <iostream>

#include <vector>

using namespace std;

int upperBound(vector<int>& arr, int target) {

int low = 0, high = arr.size() - 1;

int n = arr.size();

int result = n; // default if all elements ≤ target

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] > target) {

result = mid; // possible upper bound

high = mid - 1; // search left half

} else {

low = mid + 1; // search right half

}

}

return result;

}

int main() {

int n, target;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> target;

cout << upperBound(arr, target) << endl;

return 0;

}

# PROBLEM STATEMENT - The smallest element ≥ target (actual value, not index)

#include <iostream>

#include <vector>

using namespace std;

int findCeil(vector<int>& arr, int target) {

int low = 0, high = arr.size() - 1;

int n = arr.size();

int result = -1; // default if no element ≥ target

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] >= target) {

result = arr[mid]; // possible ceil

high = mid - 1; // search left half

} else {

low = mid + 1; // search right half

}

}

return result;

}

int main() {

int n, target;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> target;

cout << findCeil(arr, target) << endl;

return 0;

}

# PROBLEM STATEMENT - The largest element ≤ target

#include <iostream>

#include <vector>

using namespace std;

int findFloor(vector<int>& arr, int target) {

int low = 0, high = arr.size() - 1;

int result = -1; // default if no element ≤ target

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] <= target) {

result = arr[mid]; // possible floor

low = mid + 1; // search right half

} else {

high = mid - 1; // search left half

}

}

return result;

}

int main() {

int n, target;

cin >> n;

vector<int> arr(n);

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

cin >> arr[i];

cin >> target;

cout << findFloor(arr, target) << endl;

return 0;

}

# PROBLEM STATEMENT - The Treasure Map (Linear Search)

#include <iostream>

#include <vector>

using namespace std;

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

int n = matrix.size();

int m = matrix[0].size();

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

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

if (matrix[i][j] == target)

return true; // treasure found

}

}

return false; // treasure not found

}

int main() {

int n, m, target;

cin >> n >> m;

vector<vector<int>> matrix(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

cin >> target;

if (findTreasure(matrix, target))

cout << "Yes" << endl;

else

cout << "No" << endl;

return 0;

}

# PROBLEM STATEMENT - The Magical Scrolls (Linear Search Return Index)

#include <iostream>

#include <vector>

using namespace std;

pair<int,int> findScroll(vector<vector<int>>& matrix, int target) {

int n = matrix.size();

int m = matrix[0].size();

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

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

if (matrix[i][j] == target)

return {i, j}; // row, column

}

}

return {-1, -1}; // not found

}

int main() {

int n, m, target;

cin >> n >> m;

vector<vector<int>> matrix(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

cin >> target;

pair<int,int> pos = findScroll(matrix, target);

cout << "(" << pos.first << "," << pos.second << ")" << endl;

return 0;

}

# PROBLEM STATEMENT - The Battle Formation (Binary Search - Flattened)

#include <iostream>

#include <vector>

using namespace std;

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

int n = matrix.size();

int m = matrix[0].size();

int low = 0, high = n \* m - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

int row = mid / m;

int col = mid % m;

if (matrix[row][col] == target)

return true;

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

low = mid + 1;

else

high = mid - 1;

}

return false;

}

int main() {

int n, m, target;

cin >> n >> m;

vector<vector<int>> matrix(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

cin >> target;

if (searchSoldier(matrix, target))

cout << "True" << endl;

else

cout << "False" << endl;

return 0;

}

# PROBLEM STATEMENT - The Queen’s Jewels (Binary Search First Occurrence)

#include <iostream>

#include <vector>

using namespace std;

// Binary search to find first occurrence in a row

int firstOccurrenceInRow(vector<int>& row, int target) {

int low = 0, high = row.size() - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (row[mid] == target) {

result = mid;

high = mid - 1; // search left half for first occurrence

} else if (row[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

pair<int,int> findJewel(vector<vector<int>>& matrix, int target) {

int n = matrix.size();

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

int col = firstOccurrenceInRow(matrix[i], target);

if (col != -1)

return {i, col}; // first occurrence found

}

return {-1, -1}; // not found

}

int main() {

int n, m, target;

cin >> n >> m;

vector<vector<int>> matrix(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

cin >> target;

pair<int,int> pos = findJewel(matrix, target);

cout << "(" << pos.first << "," << pos.second << ")" << endl;

return 0;

}

# PROBLEM STATEMENT - The Hidden Scrolls (Staircase Search)

#include <iostream>

#include <vector>

using namespace std;

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

int n = matrix.size();

int m = matrix[0].size();

int row = 0, col = m - 1;

while (row < n && col >= 0) {

if (matrix[row][col] == target)

return true;

else if (matrix[row][col] > target)

col--; // move left

else

row++; // move down

}

return false; // target not found

}

int main() {

int n, m, target;

cin >> n >> m;

vector<vector<int>> matrix(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

cin >> target;

if (findScrollStaircase(matrix, target))

cout << "True" << endl;

else

cout << "False" << endl;

return 0;

}

# PROBLEM STATEMENT - The Magic Portal (Binary Search 2D)

#include <iostream>

#include <vector>

using namespace std;

string activatePortal(vector<vector<int>>& matrix, int target) {

int n = matrix.size();

int m = matrix[0].size();

int row = 0, col = m - 1;

while (row < n && col >= 0) {

if (matrix[row][col] == target)

return "Activated";

else if (matrix[row][col] > target)

col--; // move left

else

row++; // move down

}

return "Failed";

}

int main() {

int n, m, target;

cin >> n >> m;

vector<vector<int>> matrix(n, vector<int>(m));

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

for (int j = 0; j < m; j++)

cin >> matrix[i][j];

cin >> target;

cout << activatePortal(matrix, target) << endl;

return 0;

}