## 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:**

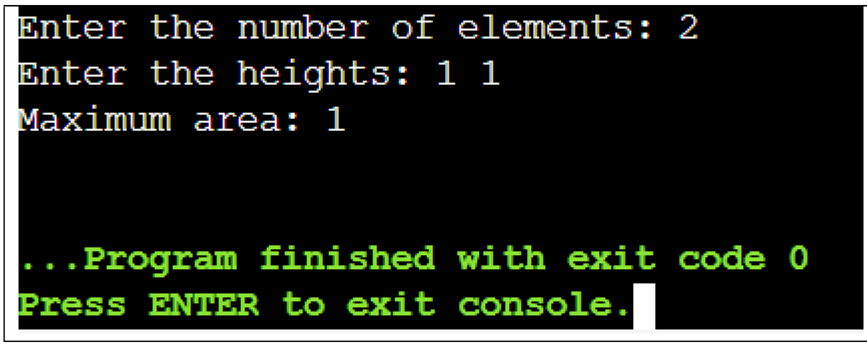
n == height.length
2 <= n <= 105
0 <= height[i] <= 104

CODE :

```
#include <iostream>
```

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

int maxArea(vector<int>& height) {
    int l = 0, r = height.size() - 1;
    int max_area = 0;

    while (l < r) {
        int area = min(height[l], height[r]) * (r - l);
        max_area = max(max_area, area);

        if (height[l] < height[r]) {
            l++;
        } else {
            r--;
        }
    }
    return max_area;
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    vector<int> height(n);

    cout << "Enter the heights: ";
    for (int i = 0; i < n; i++) {
        cin >> height[i];
    }

    cout << "Maximum area: " << maxArea(height) << endl;
    return 0;
}
```

OUTPUT:



```
Enter the number of elements: 2
Enter the heights: 1 1
Maximum area: 1


...Program finished with exit code 0
Press ENTER to exit console.
```

## 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
**Example 2:**

Input: board =
[["8","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: false
Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

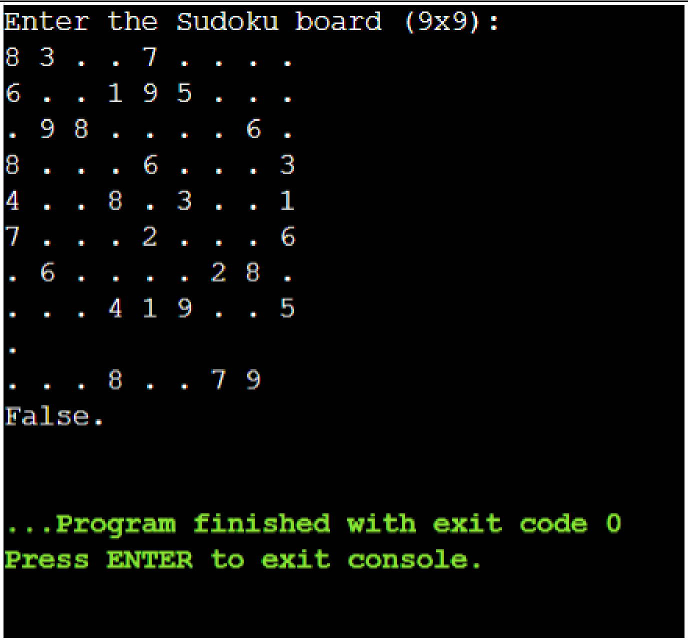
**Constraints:**

board.length == 9
board[i].length == 9
board[i][j] is a digit 1-9 or '.'.

CODE :

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

bool isValidSudoku(vector<vector<char>>& board) {
    vector<unordered_set<char>> rows(9), cols(9), boxes(9);

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            char num = board[i][j];
            if (num == '.') continue;

            int boxIndex = (i / 3) * 3 + (j / 3);
            if (rows[i].count(num) || cols[j].count(num) || boxes[boxIndex].count(num)) {
                return false;
            }

            rows[i].insert(num);
            cols[j].insert(num);
            boxes[boxIndex].insert(num);
        }
    }
    return true;
}
```

OUTPUT:



```
Enter the Sudoku board (9x9):
8 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
False.


...Program finished with exit code 0
Press ENTER to exit console.
```

**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.
Example 2:

Input: nums = [2,3,0,1,4]
Output: 2

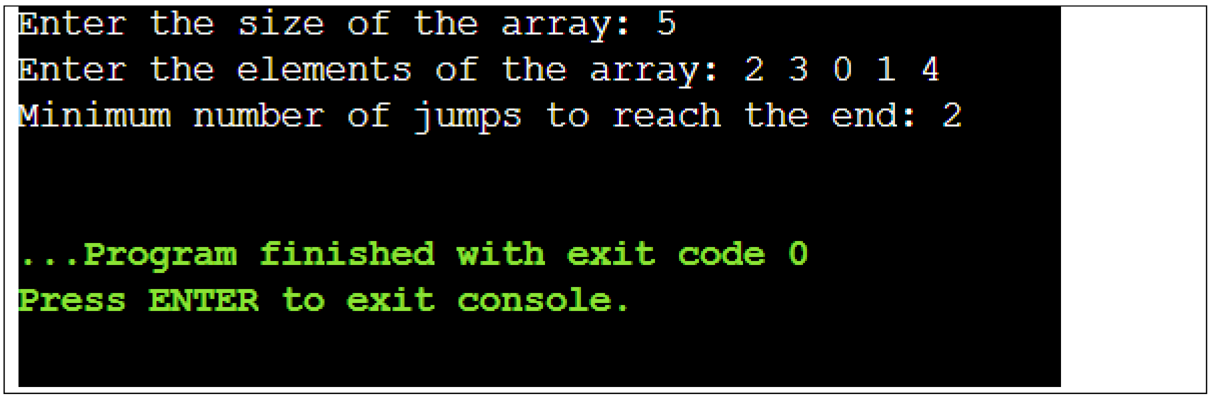
**Constraints:**

1 <= nums.length <= 104
0 <= nums[i] <= 1000
It's guaranteed that you can reach nums[n - 1].

CODE :

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

int jump(vector<int>& nums) {
    int jumps = 0, current_end = 0, farthest = 0;

    for (int i = 0; i < nums.size() - 1; i++) {
        farthest = max(farthest, i + nums[i]);

        if (i == current_end) {
            jumps++;
            current_end = farthest;
        }
    }
    return jumps;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    int result = jump(nums);
    cout << "Minimum number of jumps to reach the end: " << result << endl;
    return 0;
}
```

OUTPUT:



```
Enter the size of the array: 5
Enter the elements of the array: 2 3 0 1 4
Minimum number of jumps to reach the end: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

## 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.

**Example 2:**

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

**Constraints:**

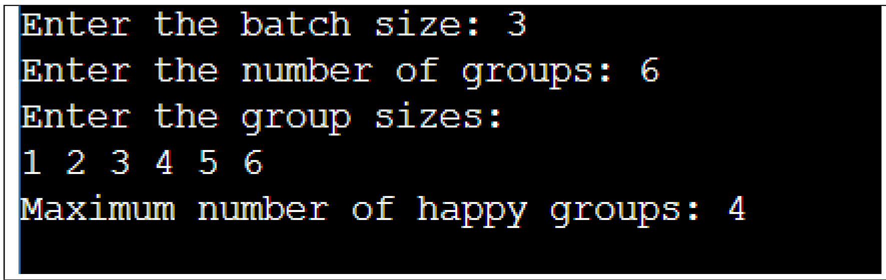
1 <= batchSize <= 9
1 <= groups.length <= 30
1 <= groups[i] <= 109

CODE:

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

int maxHappyGroups(int batchSize, vector<int>& groups) {
    vector<int> remainderCount(batchSize, 0);
    int happyGroups = 0;

    for (int group : groups) {
        int remainder = group % batchSize;
        if (remainder == 0) {
            happyGroups++;
        } else {
            remainderCount[remainder]++;
        }
    }

    for (int i = 1; i <= batchSize / 2; i++) {
        if (i == batchSize - i) {
            happyGroups += remainderCount[i] / 2;
        } else {
            int minPairs = min(remainderCount[i], remainderCount[batchSize - i]);
            happyGroups += minPairs;
            remainderCount[i] -= minPairs;
            remainderCount[batchSize - i] -= minPairs;
        }
    }

    int remaining = 0;
    for (int i = 1; i < batchSize; i++) {
        remaining += remainderCount[i] * i;
    }
    happyGroups += (remaining + batchSize - 1) / batchSize;

    return happyGroups;
}
```

```
int main() {
    int batchSize, n;
    cout << "Enter the batch size: ";
    cin >> batchSize;
    cout << "Enter the number of groups: ";
    cin >> n;

    vector<int> groups(n);
    cout << "Enter the group sizes:\n";
    for (int i = 0; i < n; i++) {
        cin >> groups[i];
    }

    int result = maxHappyGroups(batchSize, groups);
    cout << "Maximum number of happy groups: " << result << endl;

    return 0;
}
```

OUTPUT:



```
Enter the batch size: 3
Enter the number of groups: 6
Enter the group sizes:
1 2 3 4 5 6
Maximum number of happy groups: 4
```

## 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.

**Example 2:**

Input: grid = [[1,0,0,0,0,0,1],[2,0,0,0,0,3,0],[2,0,9,0,0,0,0],[0,3,0,5,4,0,0],[1,0,2,3,0,0,6]]
Output: 28
Explanation: Path of robot #1 and #2 are described in color green and blue respectively.
Cherries taken by Robot #1, (1 + 9 + 5 + 2) = 17.
Cherries taken by Robot #2, (1 + 3 + 4 + 3) = 11.
Total of cherries: 17 + 11 = 28.

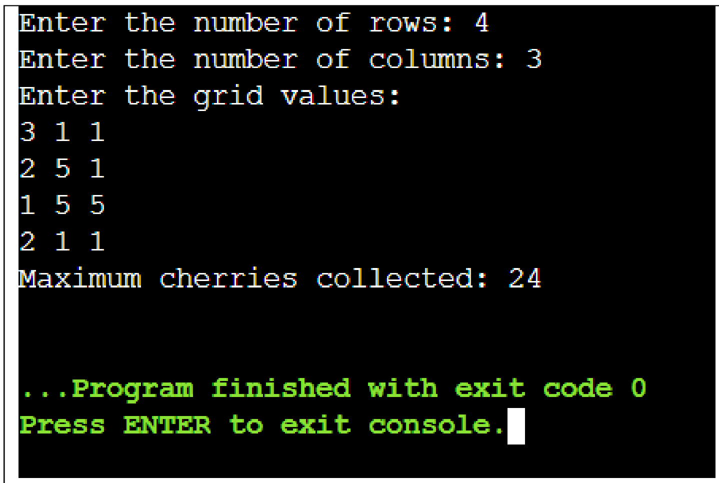
**Constraints:**

rows == grid.length
cols == grid[i].length
2 <= rows, cols <= 70
0 <= grid[i][j] <= 100

CODE :

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

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

    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;
                for (int d1 = -1; d1 <= 1; ++d1) {
                    for (int d2 = -1; d2 <= 1; ++d2) {
                        int newJ1 = j1 + d1;
                        int newJ2 = j2 + d2;
                        if (newJ1 >= 0 && newJ1 < cols && newJ2 >= 0 && newJ2 < cols) {
                            maxCherries = max(maxCherries, dp[i + 1][newJ1][newJ2]);
                        }
                    }
                }
                dp[i][j1][j2] = grid[i][j1] + grid[i][j2] + maxCherries;
            }
        }
    }

    return dp[0][0][cols - 1];
}

int main() {
    int rows, cols;
    cout << "Enter the number of rows: ";
    cin >> rows;
    cout << "Enter the number of columns: ";
    cin >> cols;

    vector<vector<int>> grid(rows, vector<int>(cols));
    cout << "Enter the grid values:\n";
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cin >> grid[i][j];
        }
    }

    cout << "Maximum cherries collected: " << cherryPickup(grid) << endl;
    return 0;
}
```

OUTPUT:



```
Enter the number of rows: 4
Enter the number of columns: 3
Enter the grid values:
3 1 1
2 5 1
1 5 5
2 1 1
Maximum cherries collected: 24


...Program finished with exit code 0
Press ENTER to exit console.
```

**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).

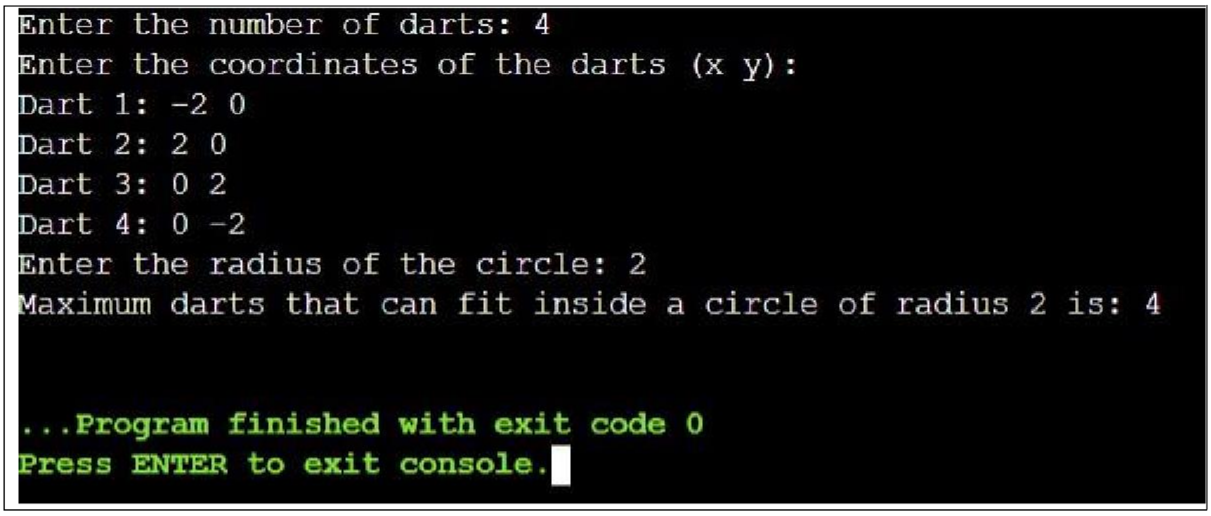
**Constraints:**

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>
using namespace std;

int maxDartsInsideCircle(vector<vector<int>>& darts, int r) {
    int maxDarts = 0;
    for (int i = 0; i < darts.size(); ++i) {
        for (int j = i + 1; j < darts.size(); ++j) {
            int count = 0;
            double centerX = (darts[i][0] + darts[j][0]) / 2.0;
            double centerY = (darts[i][1] + darts[j][1]) / 2.0;
            for (int k = 0; k < darts.size(); ++k) {
                double distance = sqrt(pow(darts[k][0] - centerX, 2) + pow(darts[k][1] - centerY,
2));
                if (distance <= r) {
                    count++;
                }
            }
            maxDarts = max(maxDarts, count);
        }
    }
    return maxDarts;
}

int main() {
    int n, radius;
    cout << "Enter the number of darts: ";
    cin >> n;

    vector<vector<int>> darts(n, vector<int>(2));
    cout << "Enter the coordinates of the darts (x y):" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Dart " << i + 1 << ": ";
        cin >> darts[i][0] >> darts[i][1];
    }

    cout << "Enter the radius of the circle: ";
    cin >> radius;

    int result = maxDartsInsideCircle(darts, radius);
    cout << "Maximum darts that can fit inside a circle of radius " << radius << " is: " <<
result << endl;

    return 0;
}
```

OUTPUT:



```
Enter the number of darts: 4
Enter the coordinates of the darts (x y):
Dart 1: -2 0
Dart 2: 2 0
Dart 3: 0 2
Dart 4: 0 -2
Enter the radius of the circle: 2
Maximum darts that can fit inside a circle of radius 2 is: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

## 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.

**Example 2:**

Input: jobs = [1,2,4,7,8], k = 2

Output: 11
Explanation: Assign the jobs the following way:
Worker 1: 1, 2, 8 (working time = 1 + 2 + 8 = 11)
Worker 2: 4, 7 (working time = 4 + 7 = 11)
The maximum working time is 11.

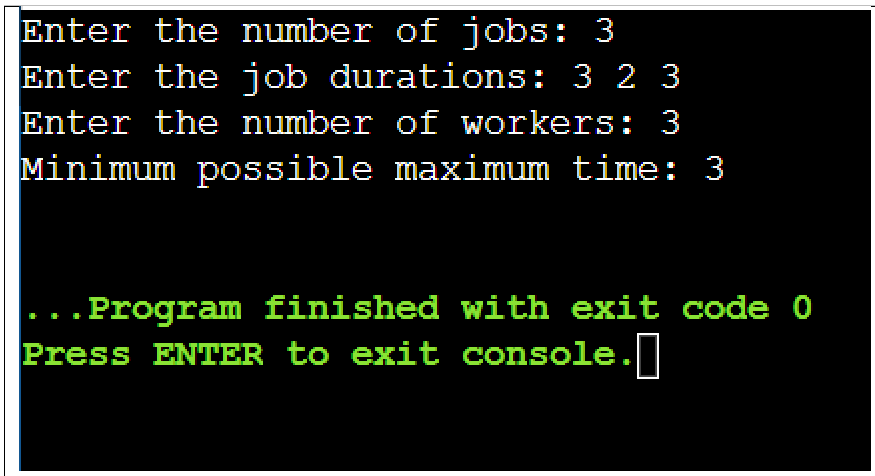
**Constraints:**

1 <= k <= jobs.length <= 12
1 <= jobs[i] <= 107

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;

bool canAssignJobs(const vector<int>& jobs, int k, int maxTime) {
    int workers = 1, currentTime = 0;

    for (int job : jobs) {
        if (currentTime + job <= maxTime) {
            currentTime += job;
        } else {
            workers++;
            if (workers > k || job > maxTime) {
                return false;
            }
            currentTime = job;
        }
    }
    return true;
}

int minTimeToFinishJobs(vector<int>& jobs, int k) {
    int left = *max_element(jobs.begin(), jobs.end());
    int right = accumulate(jobs.begin(), jobs.end(), 0);

    while (left < right) {
        int mid = left + (right - left) / 2;
        if (canAssignJobs(jobs, k, mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}

int main() {
    int n, k;
    cout << "Enter the number of jobs: ";
    cin >> n;

    vector<int> jobs(n);
    cout << "Enter the job durations: ";
    for (int i = 0; i < n; i++) {
        cin >> jobs[i];
    }

    cout << "Enter the number of workers: ";
    cin >> k;

    cout << "Minimum possible maximum time: " << minTimeToFinishJobs(jobs, k) <<
endl;
    return 0;
}
```

OUTPUT:



```
Enter the number of jobs: 3
Enter the job durations: 3 2 3
Enter the number of workers: 3
Minimum possible maximum time: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

## **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.

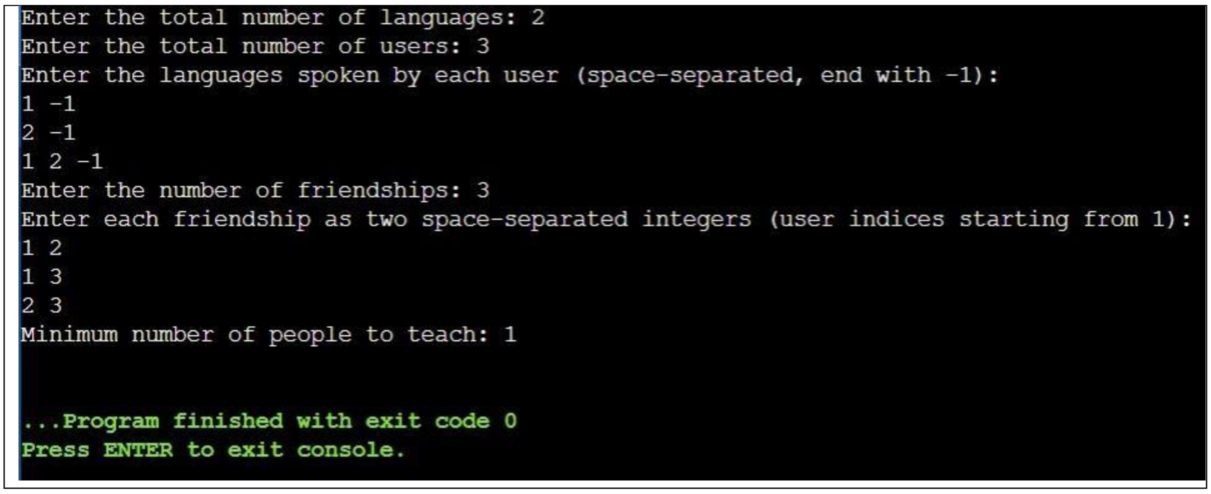
**Constraints:**

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 <iostream>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <algorithm>

using namespace std;

class UnionFind {
public:
    vector<int> parent;
    UnionFind(int n) : parent(n) {
        for (int i = 0; i < n; ++i) parent[i] = i;
    }

    int find(int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    }

    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) parent[rootX] = rootY;
    }
};

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

    for (auto& f : friendships) {
        int u = f[0] - 1, v = f[1] - 1;
        uf.unite(u, v);
    }

    unordered_map<int, unordered_set<int>> groupLanguages;
    for (int i = 0; i < m; ++i) {
        int root = uf.find(i);
        for (int lang : languages[i]) {
            groupLanguages[root].insert(lang);
        }
    }

    int usersToTeach = 0;
    for (auto& group : groupLanguages) {
        if (group.second.empty()) continue;
        int minTeach = m;
        for (int lang = 1; lang <= n; ++lang) {
            int teachCount = 0;
            for (int i = 0; i < m; ++i) {
                if (uf.find(i) == group.first && find(languages[i].begin(), languages[i].end(),
lang) == languages[i].end()) {
                    teachCount++;
                }
            }
            minTeach = min(minTeach, teachCount);
        }
        usersToTeach += minTeach;
    }

    return usersToTeach;
}

int main() {
    int n, m, f;
    cout << "Enter the total number of languages: ";
    cin >> n;
    cout << "Enter the total number of users: ";
    cin >> m;

    vector<vector<int>> languages(m);
    cout << "Enter the languages spoken by each user (space-separated, end with -1):\n";
    for (int i = 0; i < m; ++i) {
        int lang;
        while (cin >> lang && lang != -1) {
            languages[i].push_back(lang);
        }
    }

    cout << "Enter the number of friendships: ";
    cin >> f;
    vector<vector<int>> friendships(f, vector<int>(2));
    cout << "Enter each friendship as two space-separated integers (user indices starting
from 1):\n";
    for (int i = 0; i < f; ++i) {
        cin >> friendships[i][0] >> friendships[i][1];
    }

    int result = minPeopleToTeach(n, languages, friendships);
    cout << "Minimum number of people to teach: " << result << endl;

    return 0;
}
```

OUTPUT:



```
Enter the total number of languages: 2
Enter the total number of users: 3
Enter the languages spoken by each user (space-separated, end with -1):
1 -1
2 -1
1 2 -1
Enter the number of friendships: 3
Enter each friendship as two space-separated integers (user indices starting from 1):
1 2
1 3
2 3
Minimum number of people to teach: 1


...Program finished with exit code 0
Press ENTER to exit console.
```

## 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.
**Example 2:**

Input: queries = [[1,1],[2,2],[3,3],[4,4],[5,5]]
Output: [1,2,3,10,5]

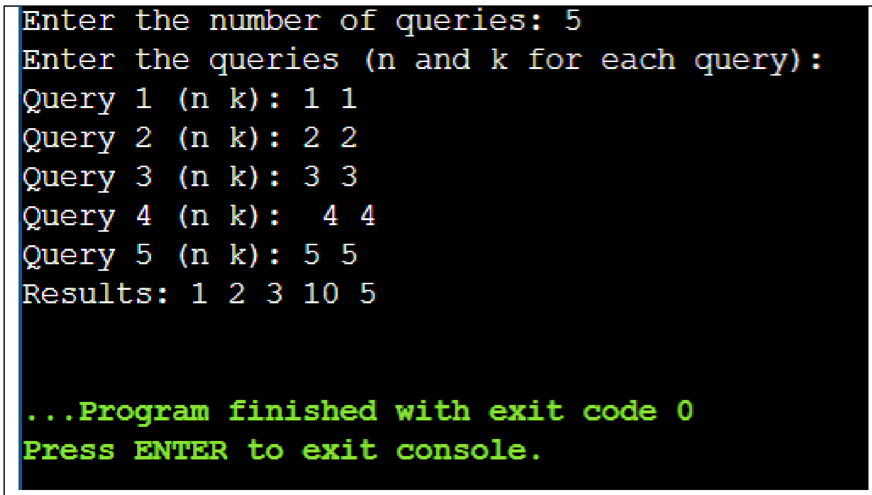
**Constraints**:

1 <= queries.length <= 104
1 <= ni, ki <= 104

Code :

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

const int MOD = 1e9 + 7;

vector<int> factorial(10001), invFactorial(10001);

int modularPower(int base, int exp, int mod) {
    int result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (1LL * result * base) % mod;
        }
        base = (1LL * base * base) % mod;
        exp /= 2;
    }
    return result;
}

void precomputeFactorials() {
    factorial[0] = 1;
    for (int i = 1; i <= 10000; ++i) {
        factorial[i] = (1LL * factorial[i - 1] * i) % MOD;
    }
    for (int i = 0; i <= 10000; ++i) {
        invFactorial[i] = modularPower(factorial[i], MOD - 2, MOD);
    }
}

int combinations(int n, int r) {
    if (r > n) return 0;
    return (1LL * factorial[n] * invFactorial[r] % MOD * invFactorial[n - r] % MOD) %
MOD;
}

unordered_map<int, int> primeFactorize(int num) {
    unordered_map<int, int> factors;
    for (int i = 2; i * i <= num; ++i) {
        while (num % i == 0) {
            factors[i]++;
            num /= i;
        }
    }
    if (num > 1) {
        factors[num]++;
    }
    return factors;
}

int countWaysToMakeArray(int n, int k) {
    unordered_map<int, int> primeFactors = primeFactorize(k);
    int result = 1;

    for (auto it = primeFactors.begin(); it != primeFactors.end(); ++it) {
        int prime = it->first;
        int power = it->second;
        result = (1LL * result * combinations(n + power - 1, power)) % MOD;
    }

    return result;
}

vector<int> countWays(vector<vector<int>>& queries) {
    vector<int> result;
    for (auto& query : queries) {
        int n = query[0], k = query[1];
        result.push_back(countWaysToMakeArray(n,  k));
    }
    return result;
}

int main() {
    precomputeFactorials();

    int q;
    cout << "Enter the number of queries: ";
    cin >> q;

    vector<vector<int>> queries(q, vector<int>(2));
    cout << "Enter the queries (n and k for each query):" << endl;
    for (int i = 0; i < q; ++i) {
        cout << "Query " << i + 1 << " (n k): ";
        cin >> queries[i][0] >> queries[i][1];
    }

    vector<int> result = countWays(queries);

    cout << "Results: ";
    for (const int res : result) {
        cout << res << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:



```
Enter the number of queries: 5
Enter the queries (n and k for each query):
Query 1 (n k): 1 1
Query 2 (n k): 2 2
Query 3 (n k): 3 3
Query 4 (n k):  4 4
Query 5 (n k): 5 5
Results: 1 2 3 10 5


...Program finished with exit code 0
Press ENTER to exit console.
```

## Q4 Maximum Twin Sum of a Linked List

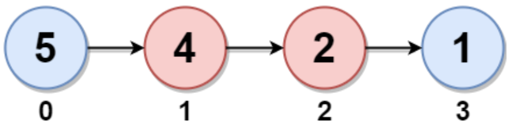
In a linked list of size n, where n is **even**, the ith node (**0-indexed**) of the linked list is known as the **twin** of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1.

- For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.

The **twin sum** is defined as the sum of a node and its twin.

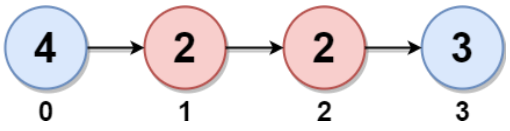
Given the head of a linked list with even length, return *the **maximum twin sum** of the linked list.*

**Example 1:**



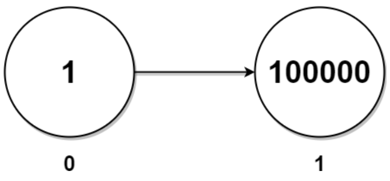
**Input:** head = [5,4,2,1]
**Output:** 6
**Explanation:**
Nodes 0 and 1 are the twins of nodes 3 and 2, respectively. All have twin sum = 6.
There are no other nodes with twins in the linked list.
Thus, the maximum twin sum of the linked list is 6.

**Example 2:**



**Input:** head = [4,2,2,3]
**Output:** 7
**Explanation:**
The nodes with twins present in this linked list are:
- Node 0 is the twin of node 3 having a twin sum of 4 + 3 = 7.
- Node 1 is the twin of node 2 having a twin sum of 2 + 2 = 4.
Thus, the maximum twin sum of the linked list is max(7, 4) = 7.

**Example 3:**



**Input:** head = [1,100000]
**Output:** 100001
**Explanation:**
There is only one node with a twin in the linked list having twin sum of 1 + 100000 = 100001.

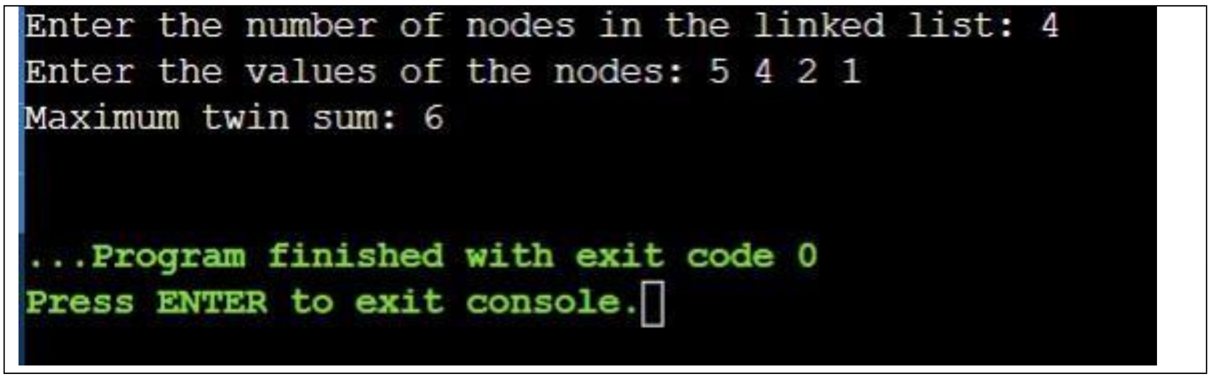
**Constraints:**

- The number of nodes in the list is an **even** integer in the range [2, $10^5$].
- 1 <= Node.val <= $10^5$

Code :

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr) {
        ListNode* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;
    }
    return prev;
}

int maxTwinSum(ListNode* head) {
    vector<int> vals;
    ListNode* slow = head, *fast = head;

    while (fast && fast->next) {
        vals.push_back(slow->val);
        slow = slow->next;
        fast = fast->next->next;
    }
    slow = reverseList(slow);

    int maxSum = 0;
    for (int i = 0; slow; ++i) {
        maxSum = max(maxSum, vals[i] + slow->val);
        slow = slow->next;
    }

    return maxSum;
}

int main() {
    int n;
    cout << "Enter the number of nodes in the linked list: ";
    cin >> n;

    if (n <= 0) {
        cout << "The list is empty. Maximum twin sum is 0." << endl;
        return 0;
    }

    cout << "Enter the values of the nodes: ";
    int val;
    cin >> val;

    ListNode* head = new ListNode(val);
    ListNode* current = head;

    for (int i = 1; i < n; ++i) {
        cin >> val;
        current->next = new ListNode(val);
        current = current->next;
    }

    cout << "Maximum twin sum: " << maxTwinSum(head) << endl;
    return 0;
}
```

OUTPUT:



```
Enter the number of nodes in the linked list: 4
Enter the values of the nodes: 5 4 2 1
Maximum twin sum: 6


...Program finished with exit code 0
Press ENTER to exit console.
```