DOMAIN WINTER CAMP

DAY - 5

Student Name: Priyanshu Anna **UID:** 22BCS12467

Branch: BE-CSE **Section:** 22BCS_FL_IOT-603-A

Searching and Sorting QUESTIONS :-

Very Easy

1.Searching a Number

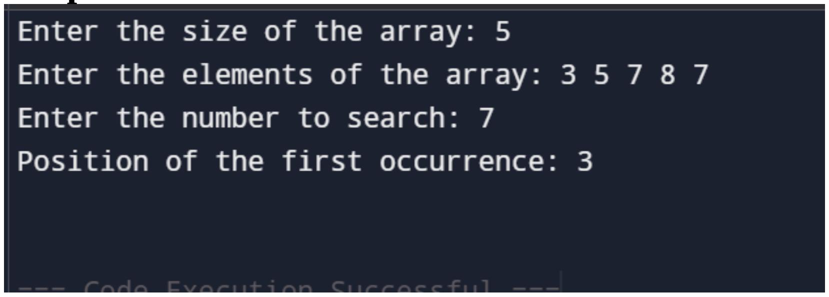
Given an integer k and array arr. Your task is to return the position of the first occurrence of k in the given array and if element k is not present in the array then return -1.

Note: 1-based indexing is followed here.

Solution:

```
#include <iostream>
#include <vector>
using namespace std;
int searchNumber(int k, const vector<int>& arr) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == k) {
            return i + 1; // 1-based indexing
        }
    }
    return -1; // If not found
}

int main() {
    int k, n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the number to search: ";
    cin >> k;
    int result = searchNumber(k, arr);
    cout << "Position of the first occurrence: " << result << endl;

    return 0;
}
```

Output:



```
Enter the size of the array: 5
Enter the elements of the array: 3 5 7 8 7
Enter the number to search: 7
Position of the first occurrence: 3



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

Easy

2. Minimum Number of Moves to Seat Everyone

There are n availabe seats and n students standing in a room. You are given an array seats of length n, where seats[i] is the position of the ith seat. You are also given the array students of length n, where students[j] is the position of the jth student.

You may perform the following move any number of times:

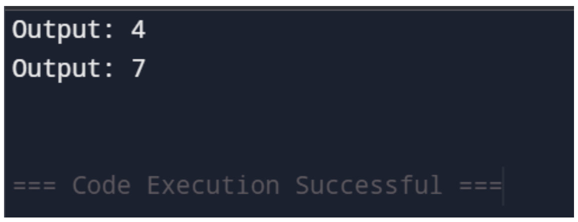
Increase or decrease the position of the ith student by 1 (i.e., moving the ith student from position x to x + 1 or x - 1)
Return the minimum number of moves required to move each student to a seat such that no two students are in the same seat.

Note that there may be multiple seats or students in the same position at the beginning.

Input:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath> // for abs

using namespace std;

int minMovesToSeat(vector<int>& seats, vector<int>& students) {
    // Sort both vectors
    sort(seats.begin(), seats.end());
    sort(students.begin(), students.end());

    int moves = 0;
    // Calculate the total moves
    for (int i = 0; i < seats.size(); i++) {
        moves += abs(seats[i] - students[i]);
    }

    return moves;
}

int main() {
    // Example 1
    vector<int> seats1 = {3, 1, 5};
    vector<int> students1 = {2, 7, 4};
    cout << "Output: " << minMovesToSeat(seats1, students1) << endl;

    // Example 2
    vector<int> seats2 = {4, 1, 5, 9};
    vector<int> students2 = {1, 3, 2, 6};
    cout << "Output: " << minMovesToSeat(seats2, students2) << endl;

    return 0;
}
```

Output:



```
Output: 4
Output: 7


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

<u>Medium</u>

3.Search in 2D Matrix.

You are given an m x n integer matrix matrix with the following two properties:

Each row is sorted in non-decreasing order.

The first integer of each row is greater than the last integer of the previous row.

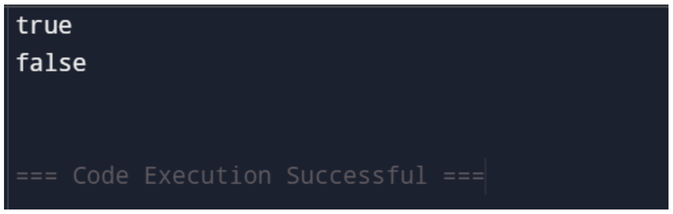
Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m * n)) time complexity.

Input:

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

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size();
    int n = matrix[0].size();
    int left = 0, right = m * n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        int midValue = matrix[mid / n][mid % n]; // Map 1D index to 2D
matrix
        if (midValue == target) return true;
        else if (midValue < target) left = mid + 1;
        else right = mid - 1;
    }
    return false;
}

int main() {
    vector<vector<int>> matrix1 = {
        {1, 3, 5, 7},
        {10, 11, 16, 20},
        {23, 30, 34, 60}
    };
    int target1 = 3;
    cout << (searchMatrix(matrix1, target1) ? "true" : "false") << endl;
    vector<vector<int>> matrix2 = {
        {1, 3, 5, 7},
        {10, 11, 16, 20},
        {23, 30, 34, 60}
    };
    int target2 = 13;
    cout << (searchMatrix(matrix2, target2) ? "true" : "false") << endl;
    return 0;
}
```

Output:



```
true
false


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

__**Hard**__

4.Sort Items by Groups Respecting Dependencies

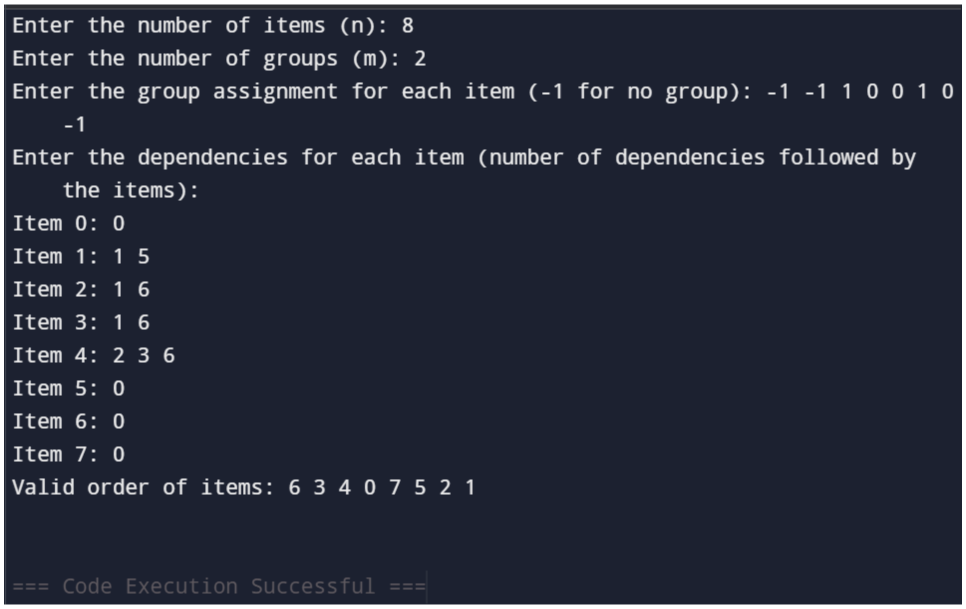
There are n items each belonging to zero or one of m groups where group[i] is the group that the i-th item belongs to and it's equal to -1 if the i-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.
Return a sorted list of the items such that:
The items that belong to the same group are next to each other in the sorted list.
There are some relations between these items where beforeItems[i] is a list containing all the items that should come before the i-th item in the sorted array (to the left of the i-th item).
Return any solution if there is more than one solution and return an empty list if there is no solution.

Solution:

```
#include <iostream>
```

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

// Helper function to perform topological sort
vector<int> topologicalSort(int n, unordered_map<int, vector<int>>& graph,
vector<int>& indegree) {
    vector<int> result;
    queue<int> q;

    // Add nodes with zero indegree
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) q.push(i);
    }

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        result.push_back(node);

        for (int neighbor : graph[node]) {
            indegree[neighbor]--;
            if (indegree[neighbor] == 0) q.push(neighbor);
        }
    }

    // If the result doesn't include all nodes, there's a cycle
    if (result.size() != n) return {};
    return result;
}

vector<int> sortItems(int n, int m, vector<int>& group,
vector<vector<int>>& beforeItems) {
    // Assign unique group IDs for items with group[i] == -1
    int groupId = m;
    for (int i = 0; i < n; i++) {
        if (group[i] == -1) group[i] = groupId++;
    }

    // Build item and group graphs
    unordered_map<int, vector<int>> itemGraph, groupGraph;
    vector<int> itemIndegree(n, 0), groupIndegree(groupId, 0);
    for (int i = 0; i < n; i++) {
        for (int prev : beforeItems[i]) {
            // Build item graph
            itemGraph[prev].push_back(i);
            itemIndegree[i]++;

            // Build group graph if items belong to different groups
            if (group[prev] != group[i]) {
                groupGraph[group[prev]].push_back(group[i]);
                groupIndegree[group[i]]++;
            }
        }
    }

    // Perform topological sort on items and groups
    vector<int> sortedItems = topologicalSort(n, itemGraph, itemIndegree);
    vector<int>    sortedGroups    =    topologicalSort(groupId,    groupGraph,
groupIndegree);

    if (sortedItems.empty() || sortedGroups.empty()) return {};

    // Group items based on sorted groups
    unordered_map<int, vector<int>> groupedItems;
    for (int item : sortedItems) {
        groupedItems[group[item]].push_back(item);
    }

    // Combine sorted groups and their items
    vector<int> result;
    for (int grp : sortedGroups) {
        result.insert(result.end(),                              groupedItems[grp].begin(),
groupedItems[grp].end());
    }

    return result;
}

int main() {
    int n, m;
    cout << "Enter the number of items (n): ";
    cin >> n;
    cout << "Enter the number of groups (m): ";
    cin >> m;

    vector<int> group(n);
    cout << "Enter the group assignment for each item (-1 for no group): ";
    for (int i = 0; i < n; i++) {
        cin >> group[i];
    }

    vector<vector<int>> beforeItems(n);
    cout << "Enter the dependencies for each item (number of dependencies
followed by the items):" << endl;
    for (int i = 0; i < n; i++) {
        int count;
        cout << "Item " << i << ": ";
        cin >> count;
        beforeItems[i].resize(count);
        for (int j = 0; j < count; j++) {
            cin >> beforeItems[i][j];
        }
    }

    vector<int> result = sortItems(n, m, group, beforeItems);

    if (result.empty()) {
        cout << "No valid order exists." << endl;
    } else {
        cout << "Valid order of items: ";
        for (int item : result) {
            cout << item << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:



```
Enter the number of items (n): 8
Enter the number of groups (m): 2
Enter the group assignment for each item (-1 for no group): -1 -1 1 0 0 1 0
    -1
Enter the dependencies for each item (number of dependencies followed by
    the items):
Item 0: 0
Item 1: 1 5
Item 2: 1 6
Item 3: 1 6
Item 4: 2 3 6
Item 5: 0
Item 6: 0
Item 7: 0
Valid order of items: 6 3 4 0 7 5 2 1


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

<u>Very Hard</u>

5.Find Minimum in Rotated Sorted Array II.

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,4,4,5,6,7] might become:

[4,5,6,7,0,1,4] if it was rotated 4 times.
[0,1,4,4,5,6,7] if it was rotated 7 times.
Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

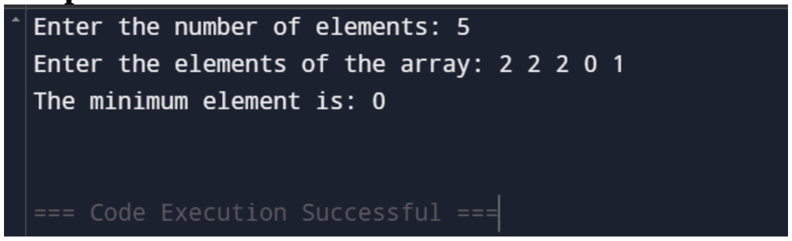
Given the sorted rotated array nums that may contain duplicates, return the minimum element of this array.

You must decrease the overall operation steps as much as possible.

Input:

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

int findMin(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;

    while (left < right) {
        int mid = left + (right - left) / 2;

        if (nums[mid] > nums[right]) {
            // Minimum lies to the right of mid
            left = mid + 1;
        } else if (nums[mid] < nums[right]) {
            // Minimum lies to the left of or at mid
            right = mid;
        } else {
            // nums[mid] == nums[right], reduce the search space
            right--;
        }
    }
    return nums[left];
}

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

    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> num[i];
    }

    int result = findMin(nums);
    cout << "The minimum element is: " << result << endl;

    return 0;
}
```

Output:



```
Enter the number of elements: 5
Enter the elements of the array: 2 2 2 0 1
The minimum element is: 0


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