DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name:Saina **UID: 22BCS15840**
Branch: BE-CSE **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

DAY-5 [24-12-2024]

Searching and Sorting :-

VERY EASY:

1.Searching a Number

Q1: 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.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;

int findFirstOccurrence(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; // Element not found}

int main() {
```

```
// Test case 1
int k1 = 16;
vector<int> arr1 = {9, 7, 16, 16, 4};
cout << "Output for Test Case 1: " << findFirstOccurrence(k1, arr1) << endl;
// Test case 2
int k2 = 98;
vector<int> arr2 = {1, 22, 57, 47, 34, 18, 66};
cout << "Output for Test Case 2: " << findFirstOccurrence(k2, arr2) << endl;
// Test case 3
int k3 = 9;
vector<int> arr3 = {1, 22, 57, 47, 34, 9, 66};
cout << "Output for Test Case 3: " << findFirstOccurrence(k3, arr3) << endl;
return 0;}
```

OUTPUT:

Output

```
Output for Test Case 1: 3
Output for Test Case 2: -1
Output for Test Case 3: 6
```

EASY:

Minimum Number of Moves to Seat Everyone

Q2: There are n available 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.

CODE:

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

int minMovesToSeat(vector<int>& seats, vector<int>& students) {
    // Sort both arrays to match each student with the closest seat
    sort(seats.begin(), seats.end());
    sort(students.begin(), students.end());

    int totalMoves = 0;

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

    return totalMoves;
}

int main() {
    // Example 1
    vector<int> seats1 = {3, 1, 5};
```

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

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

    return 0;
}
```

OUTPUT:

Output

```
Output for Example 1: 4
Output for Example 2: 7
```

Medium

Search in 2D Matrix.

Q3. 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.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;

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

    int m = matrix.size();     // Number of rows

    int n = matrix[0].size(); // Number of columns

    int left = 0, right = m * n - 1;
```

```
    // Binary search on the virtual 1D representation of the matrix
    while (left <= right) {
        int mid = left + (right - left) / 2;
        int midValue = matrix[mid / n][mid % n]; // Convert 1D index to 2D index
        if (midValue == target) {
            return true; // Target found
        } else if (midValue < target) {
            left = mid + 1;
        } else {
            right = mid - 1;}}
    return false; // Target not found}
int main() {
    // Example 1
    vector<vector<int>> matrix1 = {{1, 3, 5, 7}, {10, 11, 16, 20}, {23, 30, 34, 60}};
    int target1 = 3;
    cout << "Output for Example 1: " << (searchMatrix(matrix1, target1) ? "true" : "false")
<< endl;
    // Example 2
    vector<vector<int>> matrix2 = {{1, 3, 5, 7}, {10, 11, 16, 20}, {23, 30, 34, 60}};
    int target2 = 13;
    cout << "Output for Example 2: " << (searchMatrix(matrix2, target2) ? "true" : "false")
<< endl;
    return 0;}
```

OUTPUT:

Output

```
Output for Example 1: true
Output for Example 2: false
```

<u>Hard</u>

Sort Items by Groups Respecting Dependencies

Q4: 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.

CODE:

```
class Solution:

    def sortItems(self, n: int, m: int, group: List[int], beforeItems: List[List[int]]) -> List[int]:

        groupId = m

        for i in range(n):

            if group[i] == -1:

                group[i] = groupId

                groupId += 1


        itemGraph = defaultdict(list)

        itemIndegree = [0] * n
```

```
groupGraph = defaultdict(list)  # Initialize groupGraph
groupIndegree = [0] * groupId

for i in range(n):
    for prev in beforeItems[i]:
        itemGraph[prev].append(i)
        itemIndegree[i] += 1
        if group[i] != group[prev]:
            groupGraph[group[prev]].append(group[i])
            groupIndegree[group[i]] += 1

itemOrder = self.topologicalSort(itemGraph, itemIndegree)
groupOrder = self.topologicalSort(groupGraph, groupIndegree)

if not itemOrder or not groupOrder:
    return []

orderedGroups = defaultdict(list)
for item in itemOrder:
    orderedGroups[group[item]].append(item)

answerList = []
for groupIndex in groupOrder:
```

```
        answerList.extend(orderedGroups[groupIndex])

    return answerList

def topologicalSort(self, graph: Dict[int, List[int]], indegree: List[int]) -> List[int]:
    visited = []
    stk = []
    for i in range(len(indegree)):
        if indegree[i] == 0:
            stk.append(i)

    while stk:
        curr = stk.pop()
        visited.append(curr)

        for n in graph[curr]:
            indegree[n] -= 1
            if indegree[n] == 0:
                stk.append(n)

    return visited if len(visited) == len(graph) else []
```

OUTPUT:

```
n =
8
```

```
m =
2
```

```
group =
[-1,-1,1,0,0,1,0,-1]
```

```
beforeItems =
[[],[6],[5],[6],[3,6],[],[],[]]
```

<u>Very Hard</u>

Find Minimum in Rotated Sorted Array II.

Q5: 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.

CODE:

```
#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 must be in the right half

            left = mid + 1;

        } else if (nums[mid] < nums[right]) {

            // Minimum must be in the left half

            right = mid;

        } else {

            // nums[mid] == nums[right], reduce the search space

            right--;}}

    return nums[left];}

int main() {

    // Example 1
```

```
vector<int> nums1 = {1, 3, 5};

cout << "Minimum in Example 1: " << findMin(nums1) << endl;

// Example 2

vector<int> nums2 = {2, 2, 2, 0, 1};

cout << "Minimum in Example 2: " << findMin(nums2) << endl;

return 0;}
```

OUTPUT:

Output

```
Minimum in Example 1: 1
Minimum in Example 2: 0
```