



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: SIDDHARTH

UID: 22BCS15779

Branch: BE-CSE

Section/Group: 22BCS\_FL\_IOT-603/B

Semester: 5<sup>th</sup>

DAY-5 [24-12-2024]

### Searching and Sorting QUESTIONS :-

#### Searching

##### 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.

#### Example1:

**Input:**  $k = 16$  ,  $arr = [9, 7, 16, 16, 4]$

**Output:** 3

**Explanation:** The value 16 is found in the given array at positions 3 and 4, with position 3 being the first occurrence.

#### Example2:

**Input:**  $k=98$  ,  $arr = [1, 22, 57, 47, 34, 18, 66]$

**Output:** -1



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

**Example2:**

**Input:**  $k=9$  , arr = [1, 22, 57, 47, 34, 9, 66]

**Output:** 6

**Explanation:**  $k = 98$  isn't found in the given array.

**Expected Time Complexity:**  $O(n)$

**Expected Auxiliary Space:**  $O(1)$

**Constraints:**

- $1 \leq \text{arr.size} \leq 10^6$
- $1 \leq \text{arr}[i] \leq 10^9$
- $1 \leq k \leq 10^6$

Reference:: <https://www.geeksforgeeks.org/problems/searching-a-number0324/1>

**CODE:**

```
#include <iostream>

#include <vector>

using namespace std;

int findFirstOccurrence(int k, vector<int>& arr) {

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

        if (arr[i] == k) {

            return i + 1; // Return 1-based index

        }

    }

    return -1; // Return -1 if k is not found
```

```
}  
  
int main() {  
  
    vector<int> arr1 = {9, 7, 16, 16, 4};  
  
    cout << findFirstOccurrence(16, arr1) << endl; // Output: 3  
  
    vector<int> arr2 = {1, 22, 57, 47, 34, 18, 66};  
  
    cout << findFirstOccurrence(98, arr2) << endl; // Output: -1  
  
    vector<int> arr3 = {1, 22, 57, 47, 34, 9, 66};  
  
    cout << findFirstOccurrence(9, arr3) << endl; // Output: 6  
  
    return 0;  
  
}
```

**OUTPUT:**

```
3  
-1  
6
```

**Easy****2. Minimum Number of Moves to Seat Everyone**

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  $i$ th seat. You are also given the array `students` of length  $n$ , where `students[j]` is the position of the  $j$ th student.

You may perform the following move any number of times:

Increase or decrease the position of the  $i$ th student by 1 (i.e., moving the  $i$ th 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.

### Example 1:

**Input:** seats = [3,1,5], students = [2,7,4]

**Output:** 4

**Explanation:** The students are moved as follows:

- The first student is moved from position 2 to position 1 using 1 move.
- The second student is moved from position 7 to position 5 using 2 moves.
- The third student is moved from position 4 to position 3 using 1 move.

In total,  $1 + 2 + 1 = 4$  moves were used.

### Example 2:

**Input:** seats = [4,1,5,9], students = [1,3,2,6]

**Output:** 7

**Explanation:** The students are moved as follows:

- The first student is not moved.
- The second student is moved from position 3 to position 4 using 1 move.
- The third student is moved from position 2 to position 5 using 3 moves.
- The fourth student is moved from position 6 to position 9 using 3 moves.

In total,  $0 + 1 + 3 + 3 = 7$  moves were used.

o **Reference:** <https://leetcode.com/problems/minimum-number-of-moves-to-seat-everyone/description/>

CODE:

```
#include <iostream>
```

```
#include <vector>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
int minMovesToSeats(vector<int>& seats, vector<int>& students) {
```

```
    // Sort both arrays
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
sort(seats.begin(), seats.end());

sort(students.begin(), students.end());


int moves = 0;

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

    moves += abs(seats[i] - students[i]); // Calculate the moves needed for each
student

}

return moves;

}


int main() {

    vector<int> seats1 = {3, 1, 5};

    vector<int> students1 = {2, 7, 4};

    cout << minMovesToSeats(seats1, students1) << endl; // Output: 4


    vector<int> seats2 = {4, 1, 5, 9};

    vector<int> students2 = {1, 3, 2, 6};

    cout << minMovesToSeats(seats2, students2) << endl; // Output: 7


    return 0;

}
```

OUTPUT:

4  
7

## Medium

### 3.Search in 2D Matrix.

You are given an  $m \times n$  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.

#### Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

**Output:** true

#### Example2:

1	3	5	7
10	11	16	20
23	30	34	60

**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13

**Output:** false

**Constraints:**

$m == \text{matrix.length}$

$n == \text{matrix}[i].\text{length}$

$1 \leq m, n \leq 10^0$

$-10^4 \leq \text{matrix}[i][j], \text{target} \leq 10^4$

Ø **Reference:** <https://leetcode.com/problems/search-a-2d-matrix/description/>

**CODE:**

```
#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 midElement = matrix[mid / n][mid % n]; // Access the element at index
        mid in 2D matrix

        if (midElement == target) {

            return true;

        } else if (midElement < 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}};

    cout << (searchMatrix(matrix1, 3) ? "true" : "false") << endl; // Output: true
```



```
vector<vector<int>> matrix2 = {{1, 3, 5, 7}, {10, 11, 16, 20}, {23, 30, 34, 60}};
```

```
cout << (searchMatrix(matrix2, 13) ? "true" : "false") << endl; // Output:  
false
```

```
return 0;
```

```
}
```

**OUTPUT:**

```
true  
false
```

### 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.

### **Example 1:**

Item	Group	Before
0	-1	
1	-1	6
2	1	5
3	0	6
4	0	3, 6
5	1	
6	0	
7	-1	

**Input:**  $n = 8$ ,  $m = 2$ ,  $group = [-1, -1, 1, 0, 0, 1, 0, -1]$ ,  $beforeItems = [[], [6], [5], [6], [3, 6], [], [], []]$

**Output:**  $[6, 3, 4, 1, 5, 2, 0, 7]$

**Example 2:**

**Input:**  $n = 8$ ,  $m = 2$ ,  $group = [-1, -1, 1, 0, 0, 1, 0, -1]$ ,  $beforeItems = [[], [6], [5], [6], [3], [], [4], []]$

**Output:**  $[]$

**Explanation:** This is the same as example 1 except that 4 needs to be before 6 in the sorted list.

**Constraints:**

- $1 \leq m \leq n \leq 3 * 10^4$
- $group.length == beforeItems.length == n$
- $-1 \leq group[i] \leq m - 1$
- $0 \leq beforeItems[i].length \leq n - 1$
- $0 \leq beforeItems[i][j] \leq n - 1$
- $i \neq beforeItems[i][j]$
- $beforeItems[i]$  does not contain duplicates elements.

**Reference:** <https://leetcode.com/problems/sort-items-by-groups-respecting-dependencies/>



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## CODE:

```
#include <iostream>
```

```
#include <vector>
```

```
#include <deque>
```

```
#include <unordered_map>
```

```
#include <unordered_set>
```

```
using namespace std;
```

```
vector<int> topologicalSort(const unordered_map<int, vector<int>>& graph,  
vector<int>& inDegree, int n) {
```

```
    deque<int> queue;
```

```
    vector<int> result;
```

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

```
        if (inDegree[i] == 0) {
```

```
            queue.push_back(i);
```

```
        }
```

```
    }
```

```
    while (!queue.empty()) {
```

```
        int node = queue.front();
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
queue.pop_front();
```

```
result.push_back(node);
```

```
for (int neighbor : graph.at(node)) {
```

```
    if (--inDegree[neighbor] == 0) {
```

```
        queue.push_back(neighbor);
```

```
    }
```

```
}
```

```
}
```

```
if (result.size() != n) {
```

```
    return {}; // Cycle detected
```

```
}
```

```
return result;
```

```
}
```

```
vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>&  
beforeItems) {
```

```
    // Step 1: Create item graph and group graph
```

```
    unordered_map<int, vector<int>> itemGraph;
```

```
    vector<int> itemInDegree(n, 0);
```

```
unordered_map<int, vector<int>> groupGraph;  
  
vector<int> groupInDegree(m, 0);
```

// Step 2: Process the beforeItems to build the graph

```
for (int i = 0; i < n; ++i) {  
    for (int before : beforeItems[i]) {  
        itemGraph[before].push_back(i);  
        itemInDegree[i]++;  
        if (group[i] != group[before]) {  
            groupGraph[group[before]].push_back(group[i]);  
            groupInDegree[group[i]]++;  
        }  
    }  
}  
}
```

// Step 3: Topological sort for items

```
vector<int> itemOrder = topologicalSort(itemGraph, itemInDegree, n);  
  
if (itemOrder.empty()) {  
    return {}; // No valid ordering for items  
}
```

// Step 4: Group items by their group

```
unordered_map<int, vector<int>> groupItems;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
for (int item : itemOrder) {  
    if (group[item] == -1) {  
        groupItems[-1].push_back(item);  
    } else {  
        groupItems[group[item]].push_back(item);  
    }  
}  
  
// Step 5: Topological sort for groups  
vector<int> groupOrder = topologicalSort(groupGraph, groupInDegree, m);  
if (groupOrder.empty()) {  
    return {}; // No valid ordering for groups  
}  
  
// Step 6: Collect items in sorted order according to group order  
vector<int> result;  
for (int g : groupOrder) {  
    for (int item : groupItems[g]) {  
        result.push_back(item);  
    }  
}  
  
return result;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}
```

```
int main() {
```

```
    // Example 1
```

```
    int n1 = 8, m1 = 2;
```

```
    vector<int> group1 = {-1, -1, 1, 0, 0, 1, 0, -1};
```

```
    vector<vector<int>> beforeItems1 = { {}, {6}, {5}, {6}, {3, 6}, {}, {}, {} };
```

```
    vector<int> result1 = sortItems(n1, m1, group1, beforeItems1);
```

```
    if (!result1.empty()) {
```

```
        for (int item : result1) {
```

```
            cout << item << " ";
```

```
        }
```

```
    } else {
```

```
        cout << "[]";
```

```
    }
```

```
    cout << endl;
```

```
    // Example 2
```

```
    int n2 = 8, m2 = 2;
```

```
    vector<int> group2 = {-1, -1, 1, 0, 0, 1, 0, -1};
```

```
    vector<vector<int>> beforeItems2 = { {}, {6}, {5}, {6}, {3}, {}, {4}, {} };
```

```
vector<int> result2 = sortItems(n2, m2, group2, beforeItems2);
```

```
if (!result2.empty()) {  
    for (int item : result2) {  
        cout << item << " ";  
    }  
} else {  
    cout << "[]";  
}  
cout << endl;
```

```
return 0;
```

```
}
```

OUTPUT:

```
n = 8, m = 2  
group = {-1, -1, 1, 0, 0, 1, 0, -1}  
beforeItems = { {}, {6}, {5}, {6}, {3, 6}, {}, {}, {} }
```

```
6 3 4 1 5 2 0 7
```

Very Hard

**5.Find Minimum in Rotated Sorted Array II.**





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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.

## Example 1:

**Input:** `nums = [1,3,5]`

**Output:** 1

## Example 2:

**Input:** `nums = [2,2,2,0,1]`

**Output:** 0

## Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5000$
- $-5000 \leq \text{nums}[i] \leq 5000$
- `nums` is sorted and rotated between 1 and  $n$  times.

Reference:-

<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-ii/description/>

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;

// Compare mid with the rightmost element
if (nums[mid] < nums[right]) {
    // Minimum lies to the left of mid, including mid
    right = mid;
} else if (nums[mid] > nums[right]) {
    // Minimum lies to the right of mid
    left = mid + 1;
} else {
    // nums[mid] == nums[right], reduce the search space
    right--;
}

return nums[left];
}

int main() {
    vector<int> nums1 = {1, 3, 5};
    cout << "Minimum in nums1: " << findMin(nums1) << endl; // Output: 1

    vector<int> nums2 = {2, 2, 2, 0, 1};
    cout << "Minimum in nums2: " << findMin(nums2) << endl; // Output: 0

    return 0;
}
```

OUTPUT:

```
Minimum in nums1: 1
Minimum in nums2: 0
```