



## DOMAIN WINTER WINNING CAMP ASSIGNMENT

**Student Name:** Arush Tripathi  
**Branch:** BE-CSE::CS201  
**Semester:** 5<sup>th</sup>

**UID:** 22BCS11993  
**Section/Group:** 22BCS\_FL\_IOT-603/A

### ➤ DAY-5 [24-12-2024]

#### 1. Searching a Number

(*Very Easy*)

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.]*

##### Example 1:

**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.

##### Example 2:

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

**Output:** -1

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

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

##### Constraints:

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

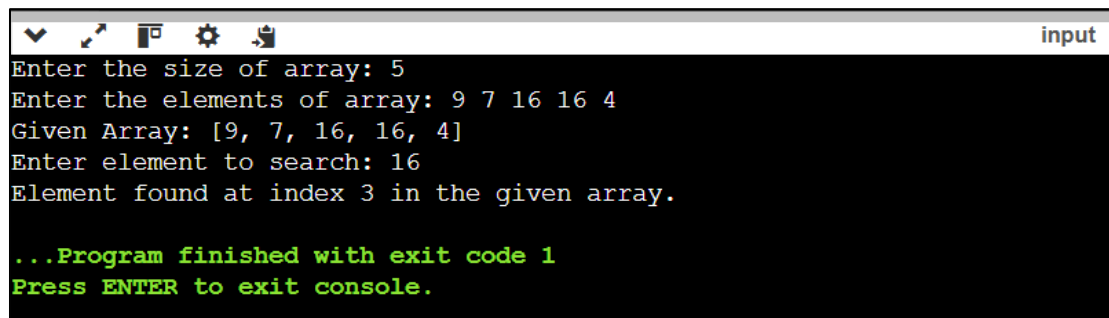
## Implementation/Code:

```
#include <iostream>                                     //Programming in C++
#include <cmath>
using namespace std;

int main()
{
    int n,k;
    cout<<"Enter the size of array: ";
    cin>>n;
    if(n<1||n>pow(10,6))
    {
        cout<<"Invalid Input.";
        return 0;
    }
    int arr[n];
    cout<<"Enter the elements of array: ";
    for(int i=1;i<=n;i++)
    {
        cin>>arr[i];
        if(arr[i]<1||arr[i]>pow(10,9))
        {
            cout<<"Invalid Input.";
            return 0;
        }
    }
    cout<<"Given Array: [";
    for(int i=1;i<n;i++)
    {
        cout<<arr[i]<<" ";
    }
    cout<<arr[n]<<"]"<<endl;
    cout<<"Enter element to search: ";
    cin>>k;
    if(k<1||k>pow(10,6))
    {
        cout<<"Invalid Input.";
```

```
        return 0;
    }
    for(int i=1;i<=n;i++)
    {
        if(k==arr[i])
        {
            cout<<"Element found at index "<<i<<" in the given array.";
            return 1;
        }
    }
    cout<<"-1 (indicates element not found)";
    return 0;
}
```

## Output:



```
input
Enter the size of array: 5
Enter the elements of array: 9 7 16 16 4
Given Array: [9, 7, 16, 16, 4]
Enter element to search: 16
Element found at index 3 in the given array.

...Program finished with exit code 1
Press ENTER to exit console.
```

## 2. Minimum Number of Moves to Seat Everyone

(Easy)

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

You may perform the following move any number of times:

Increase or decrease the position of the  $i^{\text{th}}$  student by 1 (i.e., moving the  $i^{\text{th}}$  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

### **Implementation/Code:**


```
#include <iostream> //Programming in C++
#include <vector>
#include <algorithm>
using namespace std;

int minMovesToSeat(vector<int>& seats, vector<int>& students)
{
    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]);
    }
}
```

```
        return moves;
    }

    int main()
    {
        vector<int> seats = {3, 1, 5};
        vector<int> students = {2, 7, 4};
        cout << minMovesToSeat(seats, students) << endl; // Output: 4
        vector<int> seats2 = {4,1,5,9};
        vector<int> students2 = {1,3,2,6};
        cout << minMovesToSeat(seats2, students2) << endl; // Output: 7
        return 0;
    }
```

## Output:



```
input
4
7
...Program finished with exit code 0
Press ENTER to exit console.
```

### 3. Search in 2D Matrix

*(Medium)*

You are given an  $m \times n$  integer 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 == matrix.length

n == matrix[i].length

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

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

## Implementation/Code:

```
#include <iostream>                                     //Programming in C++
#include <vector>

using namespace std;

bool searchMatrix(vector<vector<int>>& matrix, int target)
{
    if (matrix.empty() || matrix[0].empty()) return false;
    int m = matrix.size(), 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];
        if (midValue == target) return true;
        else if (midValue < target) left = mid + 1;
        else right = mid - 1;
    }
    return false;
}

int main()
{
    vector<vector<int>> matrix = {{1, 3, 5, 7}, {10, 11, 16, 20}, {23, 30, 34, 60}};
    int target = 3;
    cout << (searchMatrix(matrix, target) ? "true" : "false") << endl; // Output: true

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

## Output:

```
input
true
false

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

## 4. Sort Items by Groups Respecting Dependencies (Hard)

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:**  $n = 8$ ,  $m = 2$ ,  $\text{group} = [-1, -1, 1, 0, 0, 1, 0, -1]$ ,  $\text{beforeItems} = [ [], [6], [5], [6], [3, 6], [], [], [] ]$   
**Output:**  $[6, 3, 4, 1, 5, 2, 0, 7]$

### Example 2:

**Input:**  $n = 8$ ,  $m = 2$ ,  $\text{group} = [-1, -1, 1, 0, 0, 1, 0, -1]$ ,  $\text{beforeItems} = [ [], [6], [5], [6], [3], [], [4], [] ]$   
**Output:**  $[]$   
**Explanation:** This is the same as example 1 except that 4 needs to be before 6 in the sorted list.

### Constraints:

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

### **Implementation/Code:**

```
#include <iostream> //Programming in C++
#include <vector>
#include <queue>
#include <unordered_map>
#include <numeric>
using namespace std;

vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>&
beforeItems) {
    int idx = m;
    vector<vector<int>> groupItems(n + m);
    vector<int> itemDegree(n);
    vector<int> groupDegree(n + m);
    vector<vector<int>> itemGraph(n);
```

```
vector<vector<int>> groupGraph(n + m);
for (int i = 0; i < n; ++i) {
    if (group[i] == -1) {
        group[i] = idx++;
    }
    groupItems[group[i]].push_back(i);
}
for (int i = 0; i < n; ++i) {
    for (int j : beforeItems[i]) {
        if (group[i] == group[j]) {
            ++itemDegree[i];
            itemGraph[j].push_back(i);
        } else {
            ++groupDegree[group[i]];
            groupGraph[group[j]].push_back(group[i]);
        }
    }
}

vector<int> items(n + m);
iota(items.begin(), items.end(), 0);
auto topoSort = [](vector<vector<int>>& graph, vector<int>& degree, vector<int>&
items) -> vector<int> {
    queue<int> q;
    for (int& i : items) {
        if (degree[i] == 0) {
            q.push(i);
        }
    }
    vector<int> ans;
    while (!q.empty()) {
        int i = q.front();
        q.pop();
        ans.push_back(i);
        for (int& j : graph[i]) {
            if (--degree[j] == 0) {
                q.push(j);
            }
        }
    }
}
```

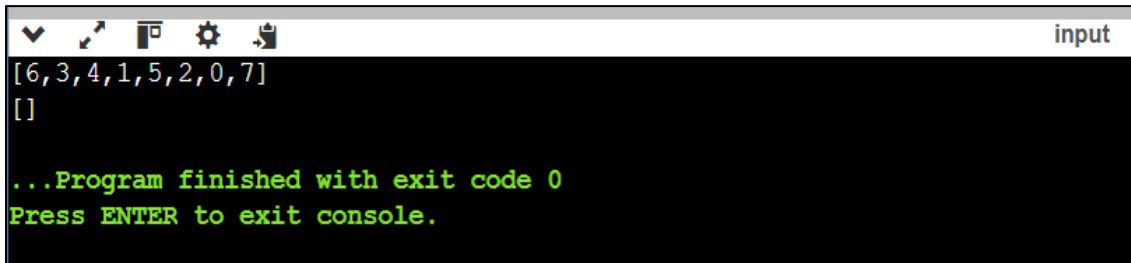
```
        }
    }
    return ans.size() == items.size() ? ans : vector<int>();
};
auto groupOrder = topoSort(groupGraph, groupDegree, items);
if (groupOrder.empty()) {
    return vector<int>();
}

vector<int> ans;
for (int& gi : groupOrder) {
    items = groupItems[gi];
    auto itemOrder = topoSort(itemGraph, itemDegree, items);
    if (items.size() != itemOrder.size()) {
        return vector<int>();
    }
    ans.insert(ans.end(), itemOrder.begin(), itemOrder.end());
}
return ans;
}

int main()
{
    int n = 8, m = 2;
    vector<int> group = {-1, -1, 1, 0, 0, 1, 0, -1};
    vector<vector<int>> beforeItems = {{}, {6}, {5}, {6}, {3, 6}, {}, {}, {}};
    vector<int> result = sortItems(n, m, group, beforeItems);
    cout<<"[";
    for (int x : result) cout << x << " ";
    cout <<"]"<<endl;

    vector<vector<int>> beforeItems2 = {{}, {6}, {5}, {6}, {3}, {}, {4}, {}};
    vector<int> result2 = sortItems(n, m, group, beforeItems2);
    cout<<"[";
    for (int x : result2) cout << x << " ";
    cout <<"]"<<endl;
    return 0;
}
```

**Output:**



```
input
[6,3,4,1,5,2,0,7]
[]

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

## 5. Find Minimum in Rotated Sorted Array II (Very Hard)

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.

## Implementation/Code:

```
#include <iostream>                                     //Programming in C++
#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]) {
            left = mid + 1;
        } else if (nums[mid] < nums[right]) {
            right = mid;
        } else {
            right--;
        }
    }
    return nums[left];
}

int main() {
    vector<int> nums = {1,3,5};
    cout << findMin(nums) << endl; // Output: 1
    vector<int> nums2 = {2, 2, 2, 0, 1};
    cout << findMin(nums2) << endl; // Output: 0
    return 0;
}
```

## Output:



```
input
1
0

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