



## DOMAIN WINTER WINNING CAMP

**Student Name:** Hamir Chauhan

**UID:** 22BCS10516

**Branch:** CSE

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

**Semester:** 5th

### 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

#### 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:**

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

**CODE:**

```
def find_position(k, arr):  
    try:  
        return arr.index(k) + 1  
    except ValueError:  
        return -1  
  
print(find_position(16, [9, 7, 16, 16, 4]))
```

Output

3

=== Code Execution Successful ===

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.

**CODE:**



```
def min_moves_to_seat(seats, students):  
  
    seats.sort()  
  
    students.sort()  
  
    return sum(abs(seat - student) for seat, student in zip(seats, students))  
  
print(min_moves_to_seat([3, 1, 5], [2, 7, 4]))
```

## Output

4

=== Code Execution Successful ===

## 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:**



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

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

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

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

## CODE:

```
def search_matrix(matrix, target):
    rows, cols = len(matrix), len(matrix[0])
    left, right = 0, rows * cols - 1

    while left <= right:
        mid = (left + right) // 2
        mid_val = matrix[mid // cols][mid % cols]
        if mid_val == target:
            return True
        elif mid_val < target:
            left = mid + 1
        else:
            right = mid - 1

    return False
print(search_matrix([[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], 3))
```

## Output

True

=== Code Execution Successful ===

## Hard

### 4.Sort Items by Groups Respecting Dependencies



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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.

**CODE:**





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

from collections import defaultdict, deque

```
def sort_items(n, m, group, beforeItems):
    group_to_items = defaultdict(list)
    item_graph = defaultdict(list)
    group_graph = defaultdict(list)
    item_indegree = [0] * n
    group_indegree = [0] * m
    for i, g in enumerate(group):
        if g == -1:
            group[i] = m
            m += 1
        group_to_items[group[i]].append(i)
    for i, items in enumerate(beforeItems):
        for item in items:
            item_graph[item].append(i)
            item_indegree[i] += 1
            if group[i] != group[item]:
                group_graph[group[item]].append(group[i])
                group_indegree[group[i]] += 1

def topological_sort(graph, indegree, nodes):
    queue = deque([node for node in nodes if indegree[node] == 0])
    order = []
    while queue:
        node = queue.popleft()
        order.append(node)
        for neighbor in graph[node]:
            indegree[neighbor] -= 1
            if indegree[neighbor] == 0:
                queue.append(neighbor)
    return order if len(order) == len(nodes) else []

group_order = topological_sort(group_graph, group_indegree, list(range(m)))
if not group_order:
    return []
result = []
for g in group_order:
    items = group_to_items[g]
    item_order = topological_sort(item_graph, item_indegree, items)
```



```
if not item_order:  
    return []  
result.extend(item_order)  
return result
```

```
print(sort_items(8, 2, [-1, -1, 1, 0, 0, 1, 0, -1], [[], [6], [5], [6], [3, 6], [], [], []]))
```

## Very Hard

### 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.

#### Example 1:

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

**Output:** 1

#### Example 2:

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

**Output:** 0

#### Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5000$



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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

## CODE:

```
def find_min(nums):  
    left, right = 0, len(nums) - 1  
    while left < right:  
        mid = (left + right) // 2  
        if nums[mid] > nums[right]:  
            left = mid + 1  
        elif nums[mid] < nums[right]:  
            right = mid  
        else:  
            right -= 1  
    return nums[left]
```

```
print(find_min([1, 3, 5]))
```

## Output

1

=== Code Execution Successful ===