Searching and Sorting QUESTIONS :-

MEGHA SHREE

UID - 22BCS10381

Very Easy (Questions 1–5)

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 <= arr.size <= 10^6
- 1 <= arr[i] <= 10^9
- 1 <= k <= 10^6

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

int findPosition(int k, vector<int>& arr) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == k) {
            return i + 1;
        }
    }
    return -1;
}
```

```
int main() {
    int k = 16;
    vector<int> arr = {9, 7, 16, 16, 4};
    cout << findPosition(k, arr) << endl;

    k = 98;
    arr = {1, 22, 57, 47, 34, 18, 66};
    cout << findPosition(k, arr) << endl;

    return 0;
}
```

2.Sorted array Search.

Given an array, arr[] sorted in ascending order and an integer k. Return true if k is present in the array, otherwise, false.

Example 1:

Input: arr[] = [1,2,3,4,6], k=6

Output: true

Explanation: Since, 6 is present in the array at index4 (0-based indexing), Output is true.

Example 2:

Input: arr[] = [1, 2, 4, 5, 6], k = 3

Output: false

Example 3:

Input: arr[] = [1, 2, 4, 5, 6], k = 6

Output: true

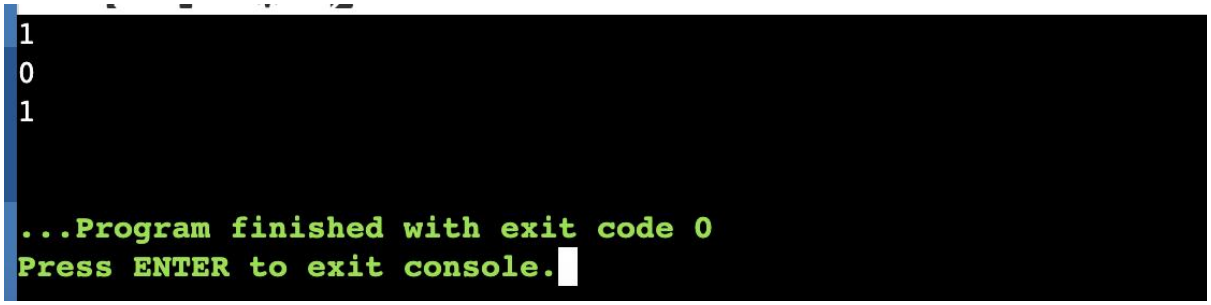
Exlpanation: Since, 3 is not present in the array, output is false.

Constraints:

- 1 <= arr.size() <= 10^6
- 1 <= k <= 106
- 1 <= arr[i] <= 10^6

```
#include <iostream>

#include <vector>

using namespace std;


bool isPresent(int k, vector<int>& arr) {

    int left = 0, right = arr.size() - 1;

    while (left <= right) {

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

        if (arr[mid] == k) {

            return true;

        } else if (arr[mid] < k) {
```

```
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return false;
}

int main() {
    vector<int> arr1 = {1, 2, 3, 4, 6};
    cout << isPresent(6, arr1) << endl;

    vector<int> arr2 = {1, 2, 4, 5, 6};
    cout << isPresent(3, arr2) << endl;

    vector<int> arr3 = {1, 2, 4, 5, 6};
    cout << isPresent(6, arr3) << endl;

    return 0;
}
```



3. Find Target Indices After Sorting Array.

You are given a 0-indexed integer array nums and a target element target.

A target index is an index i such that nums[i] == target.

Return a list of the target indices of nums after sorting nums in non-decreasing order. If there are no target indices, return an empty list. The returned list must be sorted in increasing order.

Example 1:

Input: nums = [1,2,5,2,3], target = 2
Output: [1,2]
Explanation: After sorting, nums is [1,2,2,3,5].
The indices where nums[i] == 2 are 1 and 2.

Example 2:

Input: nums = [1,2,5,2,3], target = 3
Output: [3]
Explanation: After sorting, nums is [1,2,2,3,5].
The index where nums[i] == 3 is 3.

Example 3:

Input: nums = [1,2,5,2,3], target = 5

Output: [4]
Explanation: After sorting, nums is [1,2,2,3,5].
The index where nums[i] == 5 is 4.

Constraints:

1 <= nums.length <= 100
1 <= nums[i], target <= 100

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


vector<int> targetIndices(vector<int>& nums, int target) {

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

   vector<int> result;

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

      if (nums[i] == target) {

         result.push_back(i);

      }

   }

   return result;

}


int main() {

   vector<int> nums1 = {1, 2, 5, 2, 3};
```

```
    int target1 = 2;
    vector<int> result1 = targetIndices(nums1, target1);
    for (int i : result1) cout << i << " ";
    cout << endl;

    vector<int> nums2 = {1, 2, 5, 2, 3};
    int target2 = 3;
    vector<int> result2 = targetIndices(nums2, target2);
    for (int i : result2) cout << i << " ";
    cout << endl;

    vector<int> nums3 = {1, 2, 5, 2, 3};
    int target3 = 5;
    vector<int> result3 = targetIndices(nums3, target3);
    for (int i : result3) cout << i << " ";
    cout << endl;

    return 0;
}
```

```
1 2
3
4


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

4. **Search Insert Position.**

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5

Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2

Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7

Output: 4

Constraints:

$1 <= nums.length <= 10^4$

$-104 <= nums[i] <= 10^4$

nums contains distinct values sorted in ascending order.

$-104 <= target <= 10^4$

Constraints:

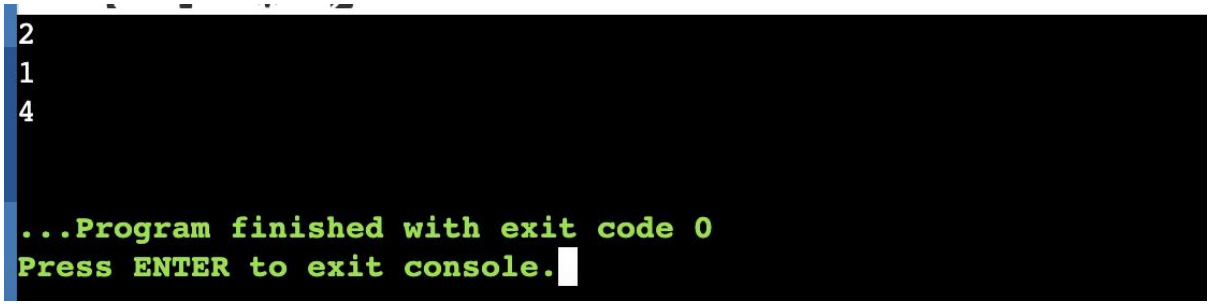
n == seats.length == students.length

1 <= n <= 100

1 <= seats[i], students[j] <= 100

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

int searchInsert(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return left;
}

int main() {
    vector<int> nums1 = {1, 3, 5, 6};
    cout << searchInsert(nums1, 5) << endl;
```

```
    vector<int> nums2 = {1, 3, 5, 6};
    cout << searchInsert(nums2, 2) << endl;

    vector<int> nums3 = {1, 3, 5, 6};
    cout << searchInsert(nums3, 7) << endl;

    return 0;
}
```



5.Relative Sort Array.

Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.

Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that do not appear in arr2 should be placed at the end of arr1 in ascending order.

Example 1:

Input: arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

Output: [2,2,2,1,4,3,3,9,6,7,19]

Example 2:

Input: arr1 = [28,6,22,8,44,17], arr2 = [22,28,8,6]

Output: [22,28,8,6,17,44]

Constraints:-

- 1 <= arr1.length, arr2.length <= 1000
- 0 <= arr1[i], arr2[i] <= 1000
- All the elements of arr2 are distinct.
- Each arr2[i] is in arr1.

```
#include <iostream>

#include <vector>

#include <unordered_map>

#include <algorithm>

using namespace std;


vector<int> relativeSortArray(vector<int>& arr1, vector<int>&
arr2) {

    unordered_map<int, int> rank;

    for (int i = 0; i < arr2.size(); i++) {

        rank[arr2[i]] = i;

    }

    sort(arr1.begin(), arr1.end(), [&](int a, int b) {

        if (rank.count(a) && rank.count(b)) {

            return rank[a] < rank[b];

        } else if (rank.count(a)) {

            return true;

        } else if (rank.count(b)) {
```

```
            return false;
        } else {
            return a < b;
        }
    });
    return arr1;
}

int main() {
    vector<int> arr1 = {2, 3, 1, 3, 2, 4, 6, 7, 9, 2, 19};
    vector<int> arr2 = {2, 1, 4, 3, 9, 6};
    vector<int> result = relativeSortArray(arr1, arr2);
    for (int x : result) cout << x << " ";
    cout << endl;
    return 0;
}
```