Searching and Sorting:-

(DAY:- 5)

Very Easy

QUESTION 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)

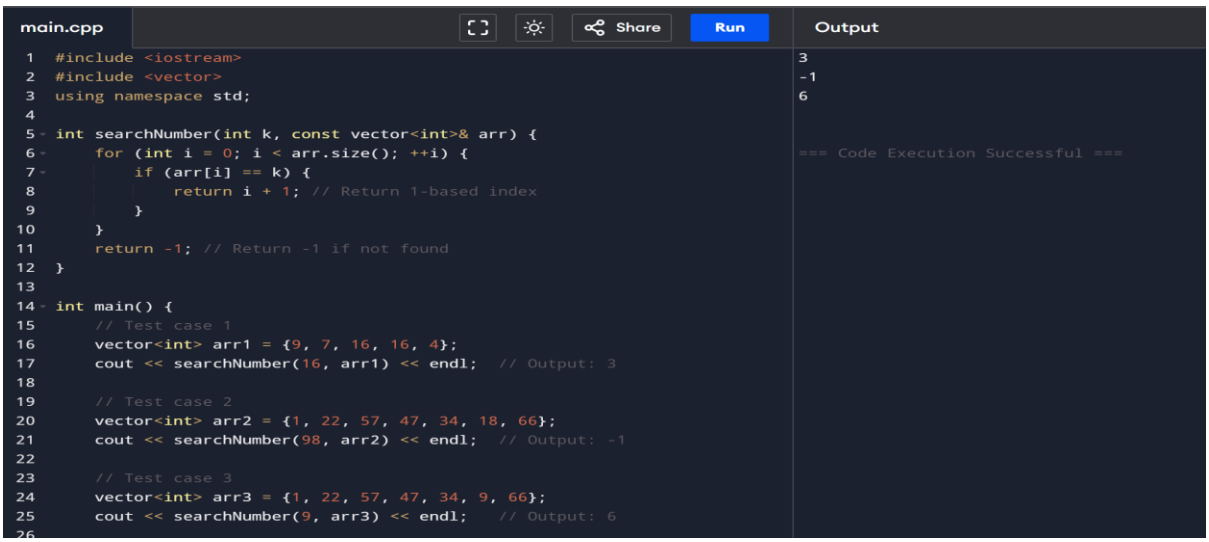
Expected Auxiliary Space: O(1)

Constraints:

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

CODE:-

```
#include <iostream>
#include <vector>
using namespace std;
int searchNumber(int k, const vector<int>& arr) {
    for (int i = 0; i < arr.size(); ++i) {
        if (arr[i] == k) {
            return i + 1;  }    }
    return -1; }
int main() {
    vector<int> arr1 = {9, 7, 16, 16, 4};
    cout << searchNumber(16, arr1) << endl;  // Output: 3
    vector<int> arr2 = {1, 22, 57, 47, 34, 18, 66};
    cout << searchNumber(98, arr2) << endl;  // Output: -1
    vector<int> arr3 = {1, 22, 57, 47, 34, 9, 66};
    cout << searchNumber(9, arr3) << endl;   // Output: 6
    return 0;}
```



QUESTION 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.

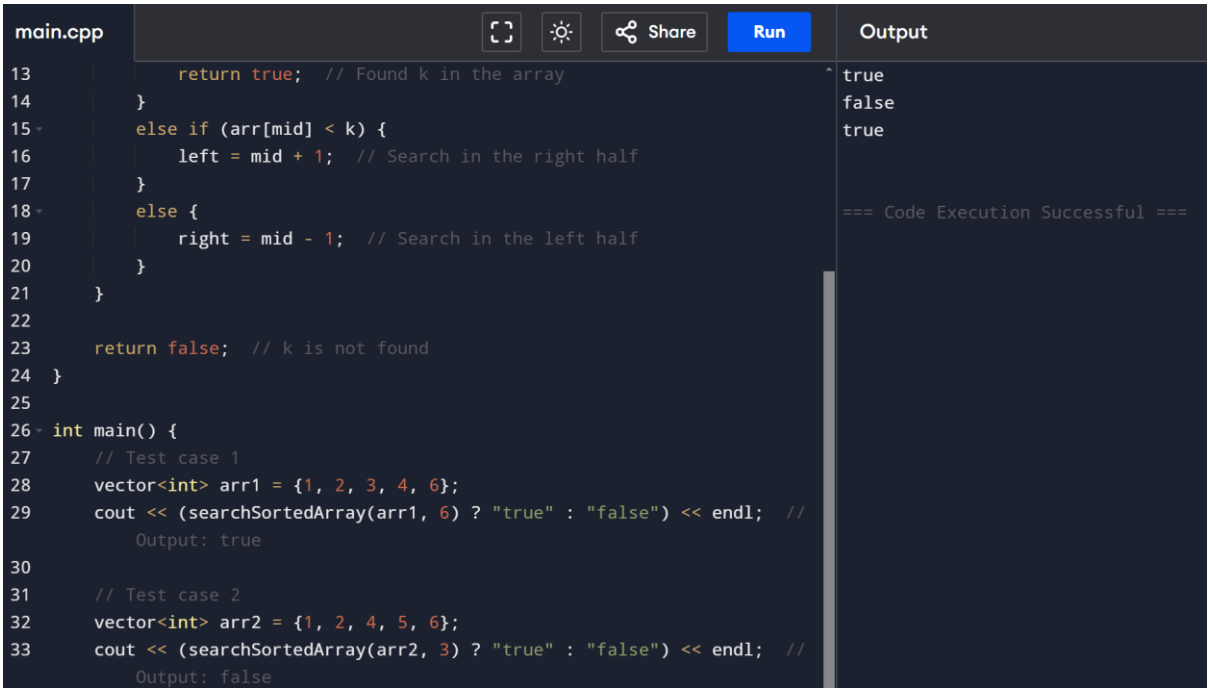
Constraints:

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

CODE:-

```
#include <iostream>

#include <vector>

using namespace std;

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

    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 << (searchSortedArray(arr1, 6) ? "true" : "false") << endl;
    vector<int> arr2 = {1, 2, 4, 5, 6};
    cout << (searchSortedArray(arr2, 3) ? "true" : "false") << endl;
    vector<int> arr3 = {1, 2, 4, 5, 6};
    cout << (searchSortedArray(arr3, 6) ? "true" : "false") << endl;
    return 0;
}
```



QUESTION 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.

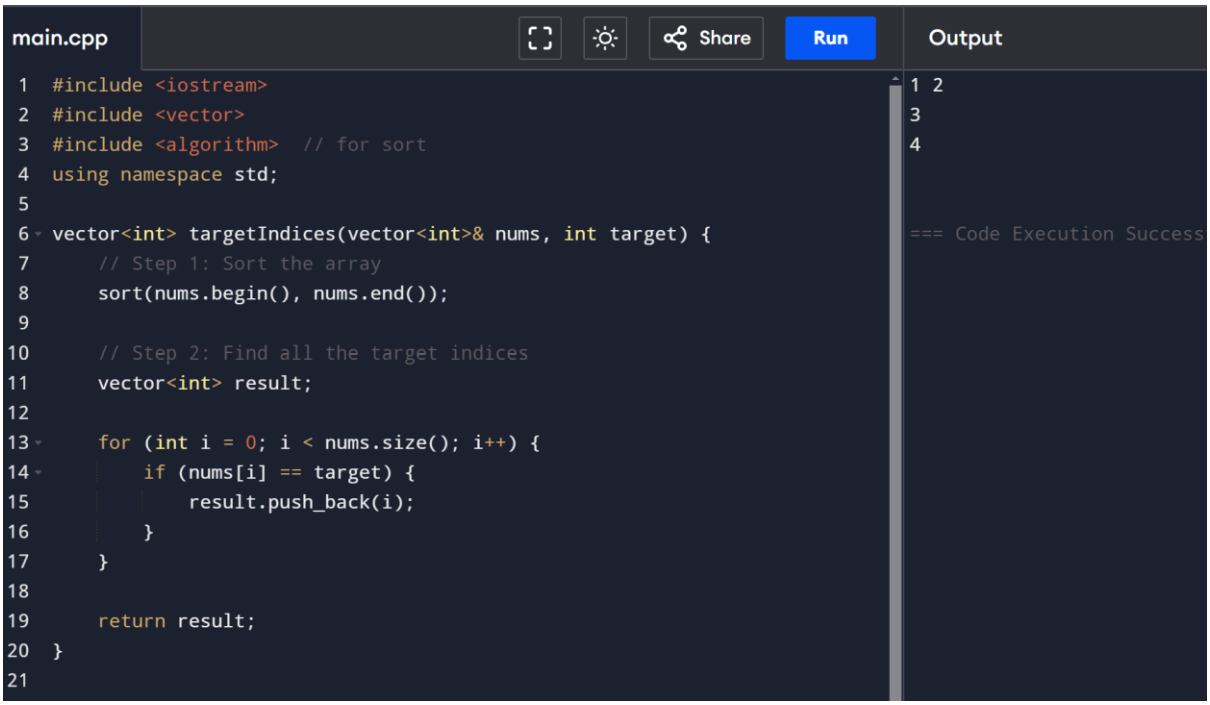
Example 3:
Input: nums = [1,2,5,2,3], target = 5
Output: [4]
Explanation: After sorting, nums is [1,2,2,3,5].
The index where nums[i] == 5 is 4.

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

CODE:-

```
#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 idx : result1) {
    cout << idx << " ";  }
    cout << endl;
    vector<int> nums2 = {1, 2, 5, 2, 3};
    int target2 = 3;
    vector<int> result2 = targetIndices(nums2, target2);
    for (int idx : result2) {
    cout << idx << " ";  }
    cout << endl;
    vector<int> nums3 = {1, 2, 5, 2, 3};
    int target3 = 5;
    vector<int> result3 = targetIndices(nums3, target3);
    for (int idx : result3) {
    cout << idx << " ";  }
    cout << endl;
    return 0;
}
```



Easy

QUESTION 1:- Squares of a Sorted Array

Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

Example 1:
Input: nums = [-4,-1,0,3,10]
Output: [0,1,9,16,100]
Explanation: After squaring, the array becomes [16,1,0,9,100].
After sorting, it becomes [0,1,9,16,100].

Example 2:
Input: nums = [-7,-3,2,3,11]
Output: [4,9,9,49,121]

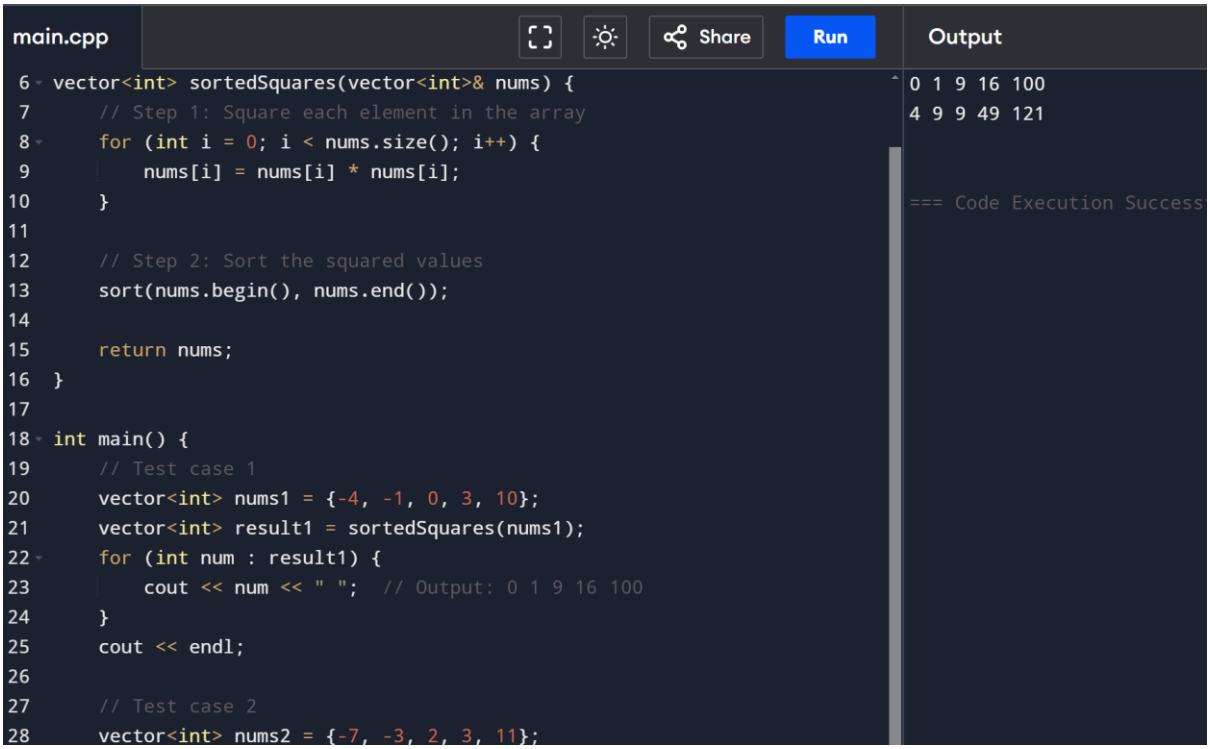
Constraints:

- $1 <= nums.length <= 10^4$
- $-10^4 <= nums[i] <= 10^4$
- nums is sorted in non-decreasing order.

CODE:-

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> sortedSquares(vector<int>& nums) {

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

    nums[i] = nums[i] * nums[i];  }

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

    return nums;

}

int main() {

    vector<int> nums1 = {-4, -1, 0, 3, 10};

    vector<int> result1 = sortedSquares(nums1);

    for (int num : result1) {

     cout << num << " ";  // Output: 0 1 9 16 100    }

    cout << endl;
```

```
    vector<int> nums2 = {-7, -3, 2, 3, 11};
    vector<int> result2 = sortedSquares(nums2);
    for (int num : result2) {
     cout << num << " ";  // Output: 4 9 9 49 121    }
    cout << endl;
    return 0;
}
```



QUESTION 2:- . Sort Even and Odd Indices Independently.

You are given a 0-indexed integer array nums. Rearrange the values of nums according to the following rules:
Sort the values at odd indices of nums in non-increasing order.
For example, if nums = [4,1,2,3] before this step, it becomes [4,3,2,1] after. The values at odd indices 1 and 3 are sorted in non-increasing order.
Sort the values at even indices of nums in non-decreasing order.
For example, if nums = [4,1,2,3] before this step, it becomes [2,1,4,3] after. The values at even indices 0 and 2 are sorted in non-decreasing order.
Return the array formed after rearranging the values of nums.

Example 1:
Input: nums = [4,1,2,3]
Output: [2,3,4,1]

Explanation:
First, we sort the values present at odd indices (1 and 3) in non-increasing order. So, nums changes from [4,1,2,3] to [4,3,2,1].
Next, we sort the values present at even indices (0 and 2) in non-decreasing order.So, nums changes from [4,1,2,3] to [2,3,4,1].
Thus, the array formed after rearranging the values is [2,3,4,1].

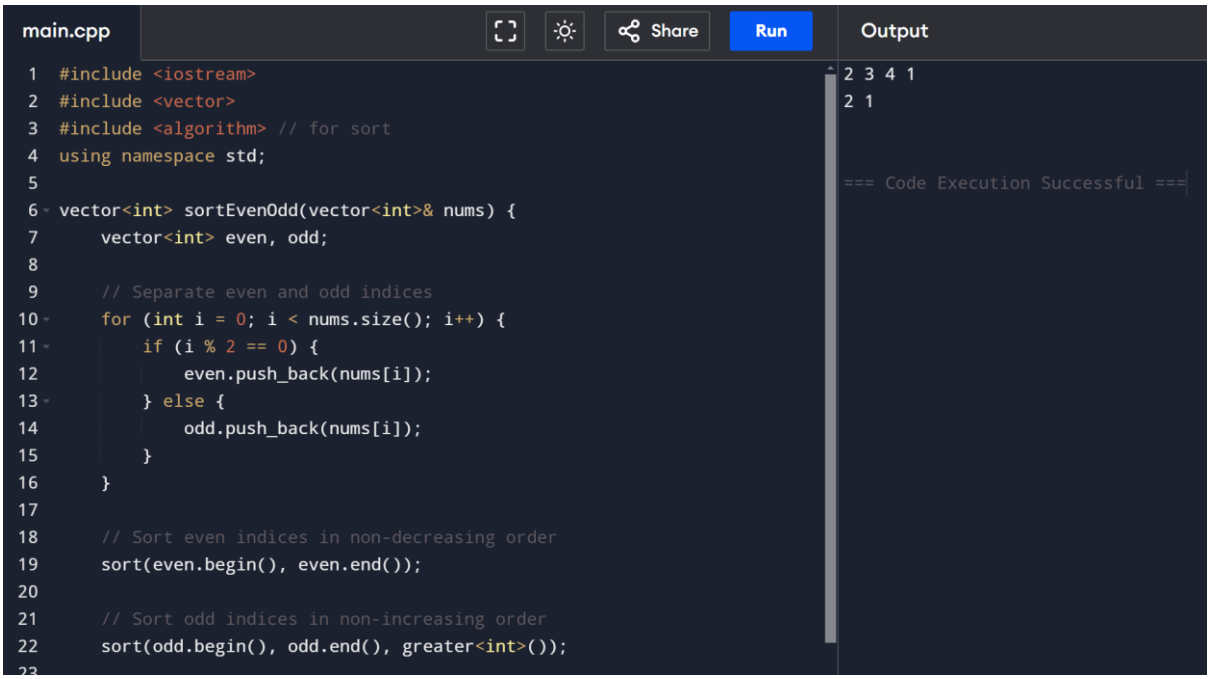
Example 2:
Input: nums = [2,1]
Output: [2,1]
Explanation:
Since there is exactly one odd index and one even index, no rearrangement of values takes place.The resultant array formed is [2,1], which is the same as the initial array.

Constraints:
- 1 <= nums.length <= 100
- 1 <= nums[i] <= 100

CODE:-

```
#include <iostream>

#include <vector>

#include <algorithm> // for sort

using namespace std;

vector<int> sortEvenOdd(vector<int>& nums) {

   vector<int> even, odd;

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

      if (i % 2 == 0) {

         even.push_back(nums[i]);

      } else {

         odd.push_back(nums[i]); }  }

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

   sort(odd.begin(), odd.end(), greater<int>());

   int evenIdx = 0, oddIdx = 0;

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

      if (i % 2 == 0) {

         nums[i] = even[evenIdx++];

      } else {
```

```
        nums[i] = odd[oddIdx++]; }  }
    return nums; }
int main() {
    vector<int> nums1 = {4, 1, 2, 3};
    vector<int> result1 = sortEvenOdd(nums1);
    for (int num : result1) {
    cout << num << " ";   }
    cout << endl;
    vector<int> nums2 = {2, 1};
    vector<int> result2 = sortEvenOdd(nums2);
    for (int num : result2) {
    cout << num << " ";
 }
    cout << endl;
    return 0;
}
```



QUESTION 3:- . Left most and Right most index.

Given a sorted array with possibly duplicate elements. The task is to find indexes of first and last occurrences of an element X in the given array. Note: If the element is not present in the array return {-1,-1} as pair.

Example1:

Input: N = 9

v[] = {1, 3, 5, 5, 5, 5, 67, 123, 125}

X = 5

Output:2 5

Explanation:

Index of first occurrence of 5 is 2

and index of last occurrence of 5 is 5.

Example2:

Input:

N = 9

v[] = {1, 3, 5, 5, 5, 5, 7, 123, 125}

X = 7

Output:

6 6

Expected Time Complexity: O(Log(N))

Expected Auxiliary Space: O(1)

Constraints:

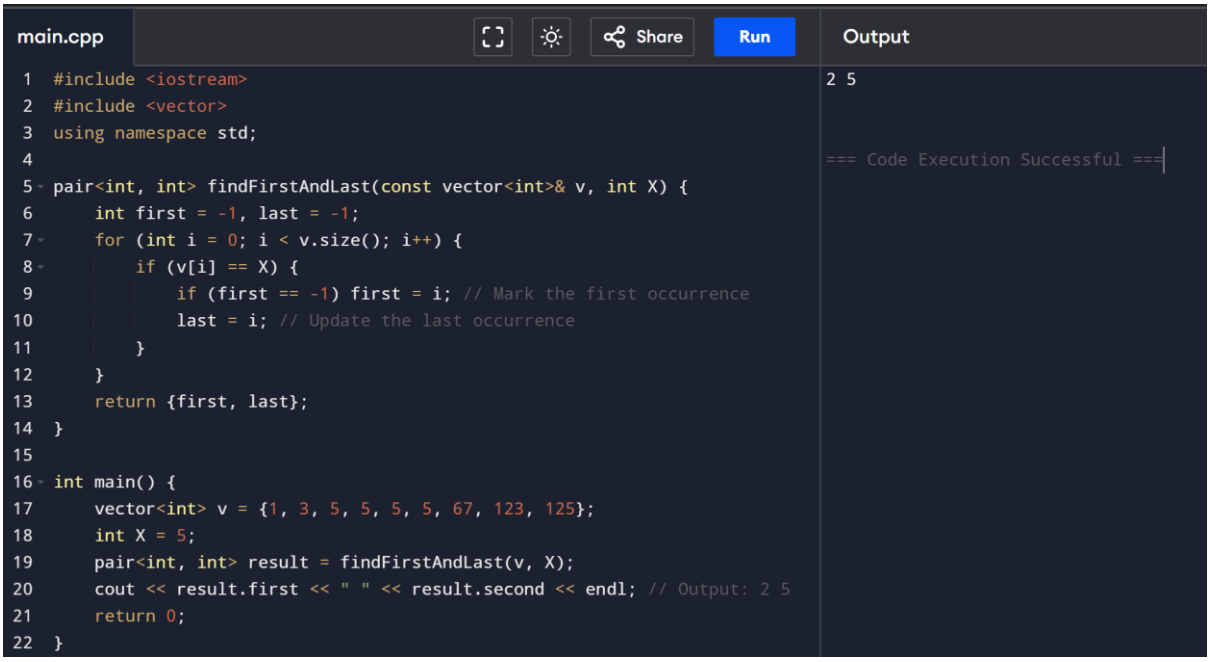
$1 \leq N \leq 10^5$

$1 \leq v[i], X \leq 10^{18}$

CODE:-

```
#include <iostream>
```

```
#include <vector>
using namespace std;
pair<int, int> findFirstAndLast(const vector<int>& v, int X) {
    int first = -1, last = -1;
    for (int i = 0; i < v.size(); i++) {
        if (v[i] == X) {
            if (first == -1) first = i; // Mark the first occurrence
            last = i; // Update the last occurrence
        }
    }
    return {first, last};
}
int main() {
    vector<int> v = {1, 3, 5, 5, 5, 5, 67, 123, 125};
    int X = 5;
    pair<int, int> result = findFirstAndLast(v, X);
    cout << result.first << " " << result.second << endl; // Output: 2 5
    return 0;
}
```



Medium

QUESTION 1:- Find First and Last Position of Element in Sorted Array.

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

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

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]

Example 3:

Input: nums = [], target = 0

Output: [-1,-1]

Constraints:

- 0 <= nums.length <= 10^5
- -10^9 <= nums[i] <= 10^9
- nums is a non-decreasing array.
- -10^9 <= target <= 10^9

CODE:-

```
#include <iostream>

#include <vector>

using namespace std;

vector<int> findFirstAndLastPosition(const vector<int>& nums, int target) {

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

    int first = -1, last = -1;
```

```
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            first = mid;
            right = mid - 1;  }
 else if (nums[mid] < target) {
            left = mid + 1;}
 else {
            right = mid - 1; } }
    left = 0, right = nums.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            last = mid;
            left = mid + 1; }
else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }   }
    return {first, last};
}
int main() {
    vector<int> nums = {5, 7, 7, 8, 8, 10};
    int target = 8;
    vector<int> result = findFirstAndLastPosition(nums, target);
    cout << "[" << result[0] << ", " << result[1] << "]" << endl;
    return 0;
}
```

main.cpp Share Run Output

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

vector<int> findFirstAndLastPosition(const vector<int>& nums, int target
    ) {
    int left = 0, right = nums.size() - 1;
    int first = -1, last = -1;

    // Find the first occurrence
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            first = mid;
            right = mid - 1; // Continue searching in the left half
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

```

```
[3, 4]


=== Code Execution Success
```

<u>QUESTION 2:-</u> Find Minimum in Rotated Sorted Array.

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] 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 of unique elements, return the minimum element of this array.
You must write an algorithm that runs in O(log n) time.

Example 1:
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
Example 2:
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.
Example 3:
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

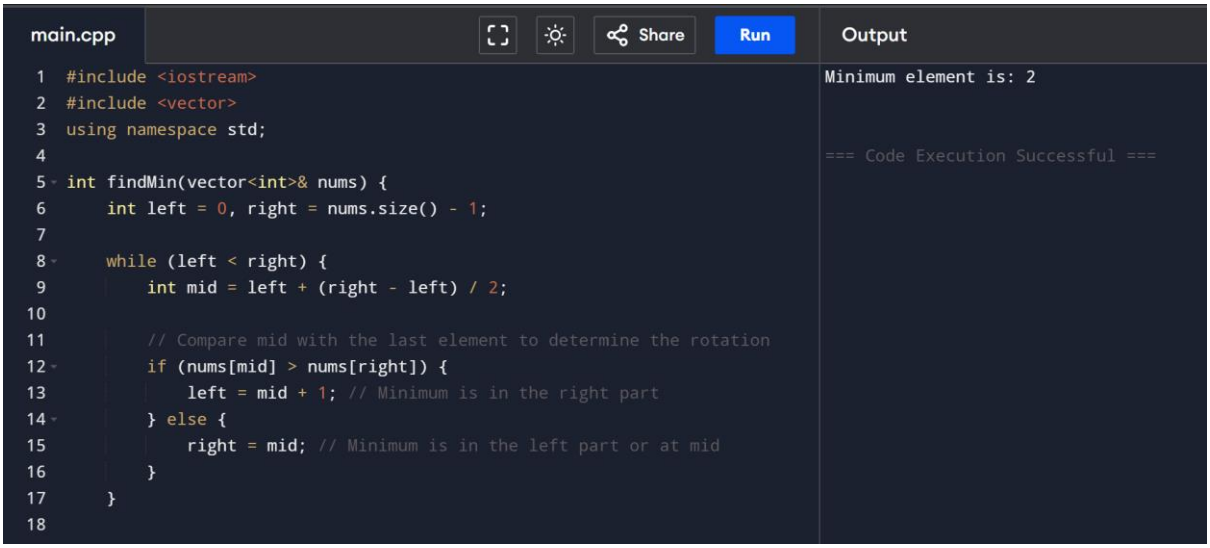
Constraints:

- n == nums.length

- 1 <= n <= 5000
- -5000 <= nums[i] <= 5000
- All the integers of nums are unique.
- nums is sorted and rotated between 1 and n times.

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]) {
            left = mid + 1;
        } else {
            right = mid; }
    }
   return nums[left];
}

int main() {
    vector<int> nums = {4, 5, 6, 7, 0, 1, 2};
    cout << "Minimum element is: " << findMin(nums) << endl;
    return 0;
}
```



QUESTION 3:- Smallest Positive Missing Number.

You are given an integer array arr[]. Your task is to find the smallest positive number missing from the array.
Note: Positive number starts from 1. The array can have negative integers too.
Examples1:
Input: arr[] = [2, -3, 4, 1, 1, 7]
Output: 3
Explanation: Smallest positive missing number is 3.

Examples2:
Input: arr[] = [5, 3, 2, 5, 1]
Output: 4
Explanation: Smallest positive missing number is 4.

Examples3:
Input: arr[] = [-8, 0, -1, -4, -3]
Output: 1
Explanation: Smallest positive missing number is 1.
Constraints:

- $1 <= arr.size() <= 10^5$
- $-10^6 <= arr[i] <= 10^6$

CODE:-

```
#include <iostream>

#include <vector>

using namespace std;

int smallestMissingPositive(vector<int>& arr) {

    int n = arr.size();

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

        while (arr[i] > 0 && arr[i] <= n && arr[arr[i] - 1] != arr[i]) {

            swap(arr[i], arr[arr[i] - 1]);  }

    }

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

        if (arr[i] != i + 1) {

            return i + 1; }  }

    return n + 1; }

int main() {

    vector<int> arr = {2, -3, 4, 1, 1, 7};

    cout << "Smallest positive missing number: " << smallestMissingPositive(arr) << endl;
```

return 0;

}

main.cpp

Share Run Output

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

int smallestMissingPositive(vector<int>& arr) {
    int n = arr.size();

      for (int i = 0; i < n; i++) {
        while (arr[i] > 0 && arr[i] <= n && arr[arr[i] - 1] != arr[i]) {
            swap(arr[i], arr[arr[i] - 1]);
        }
    }


    for (int i = 0; i < n; i++) {
        if (arr[i] != i + 1) {
            return i + 1; }
    }

    return n + 1;
}

int main() {
    vector<int> arr = {2, -3, 4, 1, 1, 7};
    cout << "Smallest positive missing number: " <<
        smallestMissingPositive(arr) << endl;
    return 0;
}
```

```
Smallest positive missing number: 3


=== Code Execution Successful ===
```

Hard

QUESTION 1:- Find the Kth Smallest Sum of a Matrix With Sorted Rows.

You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.
You are allowed to choose exactly one element from each row to form an array.
Return the kth smallest array sum among all possible arrays.

Example 1:
Input: mat = [[1,3,11],[2,4,6]], k = 5
Output: 7
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.
Example 2:
Input: mat = [[1,3,11],[2,4,6]], k = 9
Output: 17
Example 3:
Input: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7
Output: 9
Explanation: Choosing one element from each row, the first k smallest sum are:

[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.

Constraints:

- m == mat.length
- n == mat.length[i]
- 1 <= m, n <= 40
- 1 <= mat[i][j] <= 5000
- 1 <= k <= min(200, n^m)
- mat[i] is a non-decreasing array.

CODE:-

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

int kthSmallestSum(vector<vector<int>>& mat, int k) {
    priority_queue<int> maxHeap;
    maxHeap.push(0);
    for (auto& row : mat)
{
        priority_queue<int> tempHeap;
        while (!maxHeap.empty()) {
            int currSum = maxHeap.top();
            maxHeap.pop();
            for (int num : row) {
                tempHeap.push(currSum + num);
                if (tempHeap.size() > k) {
                    tempHeap.pop();
                } }
 }
        swap(maxHeap, tempHeap);  }
    return maxHeap.top();
 }
int main() {
    vector<vector<int>> mat = {{1, 3, 11}, {2, 4, 6}};
    int k = 5;
    cout << "The " << k << "th smallest sum is: " << kthSmallestSum(mat, k) << endl;
    return 0; }
```

main.cpp

Share Run

Output

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

int kthSmallestSum(vector<vector<int>>& mat, int k) {
    priority_queue<int> maxHeap;
    maxHeap.push(0);

    for (auto& row : mat) {
        priority_queue<int> tempHeap;

        while (!maxHeap.empty()) {
            int currSum = maxHeap.top();
            maxHeap.pop();

            for (int num : row) {
                tempHeap.push(currSum + num);
                if (tempHeap.size() > k) {
                    tempHeap.pop();
                }
            }
        }
```

```
The 5th smallest sum is: 7


=== Code Execution Successful ===
```

<u>QUESTION 2:-</u>Merge k Sorted Lists.

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

Example 1:
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6

Example 2:
Input: lists = []
Output: []

Example 3:
Input: lists = [[]]
Output: []
Constraints:

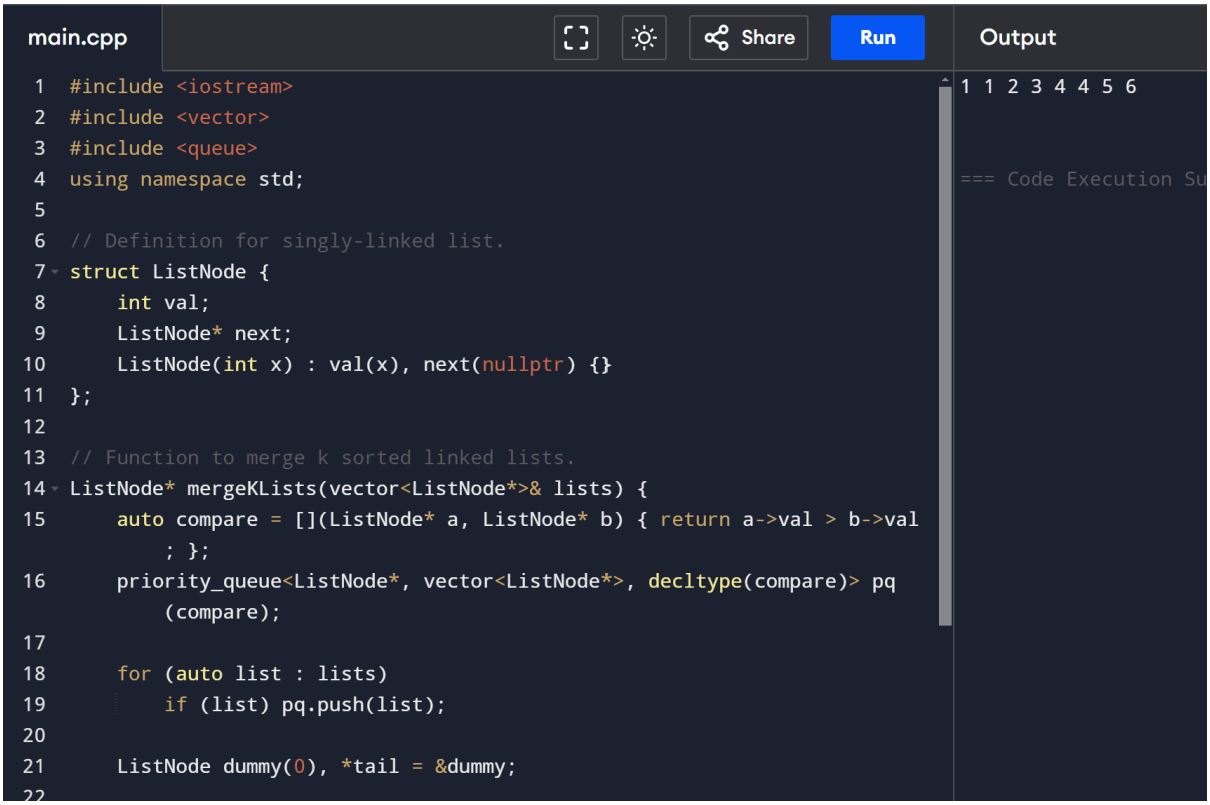
- k == lists.length
- $0 <= k <= 10^4$
- 0 <= lists[i].length <= 500
- $-10^4 <= lists[i][j] <= 10^4$

- lists[i] is sorted in ascending order.
- The sum of lists[i].length will not exceed 10^4.

CODE:-

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* mergeKLists(vector<ListNode*>& lists) {
    auto compare = [](ListNode* a, ListNode* b) { return a->val > b->val; };
    priority_queue<ListNode*, vector<ListNode*>, decltype(compare)> pq(compare);
    for (auto list : lists)
        if (list) pq.push(list);
    ListNode dummy(0), *tail = &dummy;
    while (!pq.empty()) {
        ListNode* node = pq.top(); pq.pop();
        tail->next = node;
        tail = tail->next;
        if (node->next) pq.push(node->next);  }
    return dummy.next;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next; }
    cout << endl;  }
```

```
ListNode* createList(const vector<int>& nums) {
    ListNode dummy(0), *tail = &dummy;
    for (int num : nums) {
        tail->next = new ListNode(num);
        tail = tail->next;     }
    return dummy.next;
}
int main() {
    vector<ListNode*> lists = {
        createList({1, 4, 5}),
        createList({1, 3, 4}),
        createList({2, 6})
    };
    ListNode* result = mergeKLists(lists);
    printList(result);
    return 0;
}
```



QUESTION 3:- Max Chunks To Make Sorted II

You are given an integer array arr.We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.Return the largest number of chunks we can make to sort the array.

Example 1:

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

Output: 1

Explanation: Splitting into two or more chunks will not return the required result.For example, splitting into [5, 4], [3, 2, 1] will result in [4, 5, 1, 2, 3], which isn't sorted.

Example 2:

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

Output: 4

Explanation: We can split into two chunks, such as [2, 1], [3, 4, 4].However, splitting into [2, 1], [3], [4], [4] is the highest number of chunks possible.

Constraints:

- 1 <= arr.length <= 2000
- 0 <= arr[i] <= 108

CODE:-

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;
int maxChunksToSorted(vector<int>& arr) {
    int n = arr.size();
    vector<int> rightMin(n + 1, INT_MAX);
    for (int i = n - 1; i >= 0; --i) {
        rightMin[i] = min(rightMin[i + 1], arr[i]);
    }
```

```
    int leftMax = INT_MIN, chunks = 0;
    for (int i = 0; i < n; ++i) {
        leftMax = max(leftMax, arr[i]);
        if (leftMax <= rightMin[i + 1]) {
            ++chunks;   }
    }
    return chunks;
}
int main() {
    vector<int> arr = {2, 1, 3, 4, 4};
    cout << maxChunksToSorted(arr) << endl;
    return 0;
}
```