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.

```cpp
#include <iostream>
#include <vector>
using namespace std;
int findFirstOccurrence(const vector<int>& arr, int k) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == k) {
            return i + 1; // Convert 0-based index to 1-based index
        }
    }
    return -1; // Return -1 if k is not found
}
int main() {
    int n, k;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the number to search for: ";
    cin >> k;
    int result = findFirstOccurrence(arr, k);
    if (result != -1) {
        cout << "The first occurrence of " << k << " is at position " << result << "." << endl;
    } else {
        cout << k << " is not present in the array." << endl;
    }
    return 0;
}
```

```
Enter the number of elements in the array: 5
Enter the elements of the array: 5
1
2
3
4
Enter the number to search for: 3
The first occurrence of 3 is at position 4.
```

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.

```cpp
#include <iostream>
#include <vector>
using namespace std;
bool binarySearch(const vector<int>& arr, int k) {
    int left = 0, right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2; // Prevent overflow for large indices
        if (arr[mid] == k) {
            return true; // Element found
        } else if (arr[mid] < k) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }
    return false; // Element not found
}
int main() {
    int n, k;
    cout << "Enter the number of elements in the sorted array: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements of the array in ascending order: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the number to search for: ";
    cin >> k;
    if (binarySearch(arr, k)) {
        cout << k << " is present in the array." << endl;
    } else {
        cout << k << " is not present in the array." << endl;
    }
    return 0;
}
```

```
Enter the number of elements in the sorted array: 5
Enter the elements of the array in ascending order: 1 2 3 4 5
Enter the number to search for: 4
4 is present in the array.
```

3. Find Target Indices After Sorting Array in c++ 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.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> targetIndices(vector<int>& nums, int target) {
// Sort the array in non-decreasing order
sort(nums.begin(), nums.end());
// Vector to store the target indices
vector<int> result;
// Traverse the sorted array to find indices of target
for (int i = 0; i < nums.size(); i++) {
if (nums[i] == target) {
result.push_back(i);
}}
return result;
}
int main() {
int n, target;
cout << "Enter the number of elements in the array: ";
cin >> n;
vector<int> nums(n);
cout << "Enter the elements of the array: ";
for (int i = 0; i < n; i++) {
cin >> nums[i];}
cout << "Enter the target element: ";
cin >> target;
vector<int> indices = targetIndices(nums, target);
if (indices.empty()) {
cout << "No target indices found." << endl;
} else {
cout << "Target indices after sorting the array: ";
for (int index : indices) {
cout << index << " ";}
cout << endl;}
return 0;
}
```

```
Enter the number of elements in the array: 6
Enter the elements of the array: 3 4 5 6 7 8
Enter the target element: 9
No target indices found.
```

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.

```cpp
#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; // Target found
        } else if (nums[mid] < target) {
            left = mid + 1; // Move right
        } else {
            right = mid - 1; // Move left
        }}
    return left;
}
int main() {
    int n, target;
    cout << "Enter the number of elements in the sorted array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array in ascending order: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    cout << "Enter the target value: ";
    cin >> target;
    int result = searchInsert(nums, target);
    cout << "The target value would be at index: " << result << endl;
    return 0;
}
```

```
Enter the number of elements in the sorted array: 5
Enter the elements of the array in ascending order: 2 3 4 5 6
Enter the target value: 6
The target value would be at index: 4
```

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.

```cpp
#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> countMap;
    for (int num : arr1) {
        countMap[num]++;
    }
    vector<int> result;
    // Add elements of arr1 based on the order in arr2
    for (int num : arr2) {
        while (countMap[num] > 0) {
            result.push_back(num);
            countMap[num]--;
        }
    }
    vector<int> leftovers;
    for (auto& pair : countMap) {
        while (pair.second > 0) {
            leftovers.push_back(pair.first);
            pair.second--;
        }
    }
    sort(leftovers.begin(), leftovers.end());

    // Add the sorted leftover elements to the result
    result.insert(result.end(), leftovers.begin(), leftovers.end());

    return result;
}

int main() {
    int n1, n2;
    cout << "Enter the number of elements in arr1: ";
    cin >> n1;

    vector<int> arr1(n1);
    cout << "Enter the elements of arr1: ";
```

```cpp
    for (int i = 0; i < n1; i++) {
        cin >> arr1[i];
    }

    cout << "Enter the number of elements in arr2: ";
    cin >> n2;

    vector<int> arr2(n2);
    cout << "Enter the elements of arr2: ";
    for (int i = 0; i < n2; i++) {
        cin >> arr2[i];
    }

    vector<int> result = relativeSortArray(arr1, arr2);

    cout << "The relative sorted array is: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;

}
```

```
Enter the number of elements in arr1: 5
Enter the elements of arr1: 9 8 7 6 5
Enter the number of elements in arr2: 6
Enter the elements of arr2: 1 2 3 4 5 6
The relative sorted array is: 5 6 7 8 9
```

1.   **Common in 3 Sorted Arrays.**
You are given three arrays sorted in increasing order. Find the elements that are common in all three arrays.
If there are no such elements return an empty array. In this case, the output will be -1.
Code:
```cpp
#include <iostream>
        #include <vector>
        using namespace std;
```

```cpp
// Function to find common elements in three sorted arrays
vector<int> findCommonElements(const vector<int>& arr1, const vector<int>& arr2,
const vector<int>& arr3) {
    vector<int> result;
    int i = 0, j = 0, k = 0;

    while (i < arr1.size() && j < arr2.size() && k < arr3.size()) {
        // If the elements at all three pointers are equal, add to the result
        if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {
            // Avoid duplicates in the result
            if (result.empty() || result.back() != arr1[i]) {
                result.push_back(arr1[i]);
            }
            i++;
            j++;
            k++;
        }
        // Move the pointer with the smallest value
        else if (arr1[i] < arr2[j]) {
            i++;
        } else if (arr2[j] < arr3[k]) {
            j++;
        } else {
            k++;
        }
    }

    return result;
}

int main() {
    // Input
    int n1, n2, n3;
    cout << "Enter the size of the first array: ";
    cin >> n1;
    vector<int> arr1(n1);
    cout << "Enter the elements of the first sorted array: ";
    for (int i = 0; i < n1; i++) {
        cin >> arr1[i];
    }
```

```cpp
    cout << "Enter the size of the second array: ";
    cin >> n2;
    vector<int> arr2(n2);
    cout << "Enter the elements of the second sorted array: ";
    for (int i = 0; i < n2; i++) {
       cin >> arr2[i];
    }

    cout << "Enter the size of the third array: ";
    cin >> n3;
    vector<int> arr3(n3);
    cout << "Enter the elements of the third sorted array: ";
    for (int i = 0; i < n3; i++) {
       cin >> arr3[i];
    }

    // Find common elements
    vector<int> common = findCommonElements(arr1, arr2, arr3);

    // Output the result
    if (common.empty()) {
       cout << "Output: -1" << endl;
    } else {
       cout << "Output: ";
       for (int num : common) {
          cout << num << " ";
       }
       cout << endl;
    }

    return 0;
}
```
Output:

```
Enter the size of the first array: 5
Enter the elements of the first sorted array: 10
20
30
40
50
Enter the size of the second array: 5
Enter the elements of the second sorted array: 60
70
80
90
100
Enter the size of the third array: 5
Enter the elements of the third sorted array: 110
120
130
140
150
Output: -1
```

Medium

2. **You Search in 2D Matrix. You are given an m x 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.**
   Code:

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

// Function to search for the target in the matrix
bool searchMatrix(const vector<vector<int>>& matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) return false;

    int rows = matrix.size();
    int cols = matrix[0].size();
    int left = 0, right = rows * cols - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        int midValue = matrix[mid / cols][mid % cols]; // Map index to matrix row and column

        if (midValue == target) {
```

```cpp
            return true;
        } else if (midValue < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return false; // Target not found
}

int main() {
    // Input
    int m, n, target;
    cout << "Enter the number of rows (m): ";
    cin >> m;
    cout << "Enter the number of columns (n): ";
    cin >> n;

    vector<vector<int>> matrix(m, vector<int>(n));
    cout << "Enter the elements of the matrix row by row:\n";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> matrix[i][j];
        }
    }

    cout << "Enter the target to search for: ";
    cin >> target;

    // Search and output result
    if (searchMatrix(matrix, target)) {
        cout << "Output: true\n";
    } else {
        cout << "Output: false\n";
    }

    return 0;
}
```
Output:

```
Enter the number of rows (m): 3
Enter the number of columns (n): 3
Enter the elements of the matrix row by row:
1
2
3
4
5
6
7
8
9
Enter the target to search for: 6
Output: true
```

3. **Design 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.**
   Code:

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

// Function to find the first occurrence of the target
int findFirstPosition(const vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    int result = -1;

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

        if (nums[mid] == target) {
            result = mid;
            right = mid - 1; // Continue searching in the left half
        } else if (nums[mid] < target) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return result;
}
```

```cpp
// Function to find the last occurrence of the target
int findLastPosition(const vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    int result = -1;

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

        if (nums[mid] == target) {
            result = mid;
            left = mid + 1; // Continue searching in the right half
        } else if (nums[mid] < target) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return result;
}

// Main function to find both first and last positions
vector<int> searchRange(const vector<int>& nums, int target) {
    vector<int> result(2, -1);
    result[0] = findFirstPosition(nums, target); // First position
    if (result[0] != -1) {
        result[1] = findLastPosition(nums, target); // Last position
    }
    return result;
}

int main() {
    // Input
    int n, target;
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    vector<int> nums(n);
    cout << "Enter the elements of the sorted array: ";
    for (int i = 0; i < n; i++) {
```

```cpp
        cin >> nums[i];
    }

    cout << "Enter the target value: ";
    cin >> target;

    // Find first and last positions
    vector<int> result = searchRange(nums, target);

    // Output
    cout << "Output: [" << result[0] << ", " << result[1] << "]" << endl;

    return 0;
}
```
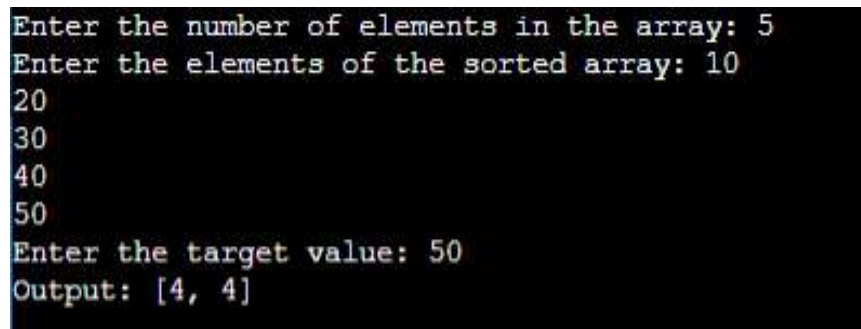
Output:

```
Enter the number of elements in the array: 5
Enter the elements of the sorted array: 10
20
30
40
50
Enter the target value: 50
Output: [4, 4]
```

Hard

4. **Maximum 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.**
   Code:

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

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

    // Step 1: Place each number in its correct index, i.e., arr[i] = i + 1
    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]);
        }
    }
}
```

```cpp
    // Step 2: Find the first index where the number doesn't match index + 1
    for (int i = 0; i < n; i++) {
        if (arr[i] != i + 1) {
            return i + 1;
        }
    }

    // Step 3: If all numbers are in their correct places, the missing number is n + 1
    return n + 1;
}

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

Output:

Smallest missing positive number is 2

5. **Maximum 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.**

Code:

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

// Custom comparator to compare pairs of (sum, indices)
class Compare {
public:
    bool operator()(const pair<int, vector<int>>& a, const pair<int, vector<int>>& b) {
        return a.first > b.first; // Min-heap based on sum
    }
};

int kthSmallest(vector<vector<int>>& mat, int k) {
```

```cpp
    int m = mat.size();    // Number of rows
    int n = mat[0].size();  // Number of columns

    // Min-heap to store the sum and the indices of the elements chosen
    priority_queue<pair<int, vector<int>>, vector<pair<int, vector<int>>>, Compare>
minHeap;

    // Initialize the heap with the first element from each row
    vector<int> initialIndices(m, 0); // Initially, pick the first element from each row
    int initialSum = 0;
    for (int i = 0; i < m; i++) {
        initialSum += mat[i][0];
    }

    minHeap.push({initialSum, initialIndices});

    // We need to pop k elements from the heap
    for (int i = 0; i < k - 1; i++) {
        pair<int, vector<int>> current = minHeap.top();
        minHeap.pop();

        // Explore the next possible sums by incrementing each row index
        for (int j = 0; j < m; j++) {
            if (current.second[j] + 1 < n) {
                vector<int> nextIndices = current.second;
                nextIndices[j]++; // Increment the index of the j-th row

                int    nextSum    =    current.first    -    mat[j][current.second[j]]    +
mat[j][nextIndices[j]];
                minHeap.push({nextSum, nextIndices});
            }
        }
    }

    // The top element of the heap now contains the kth smallest sum
    return minHeap.top().first;
}

int main() {
    vector<vector<int>> mat = {{1, 3, 11},
                               {2, 4, 6}};
```

```
    int k = 5;
    cout << "The " << k << "th smallest sum is: " << kthSmallest(mat, k) << endl;
    return 0;
}
```

Output:

The 5th smallest sum is: 7