DOMAIN WINTER WINNING CAMP

DAY: 5

Submitted by: Agrima Sharma

UID: 22BCS15314

Section: 620/A

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.

Code:

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

// Function to find the first occurrence of k in the array
int findFirstOccurrence(int k, const vector<int>& arr) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == k) {
            return i + 1; // Convert to 1-based index
        }
    }
    return -1; // Return -1 if k is not found
}

int main() {
    // Input
    int k;
    cout << "Enter the value of k: ";
    cin >> k;

    int n;
    cout << "Enter the size of 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];
    }

    // Find the position of the first occurrence of k
    int result = findFirstOccurrence(k, arr);

    // Output the result
    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;
}
```

Output:

```
Enter the value of k: 5
Enter the size of the array: 8
Enter the elements of the array: 1
2
3
4
5
6
7
8
The first occurrence of 5 is at position 5.
```

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.**

   Code:

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

// Function to perform binary search
```

```
bool sortedArraySearch(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; // k is 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; // k is not found
}

int main() {
    // Input
    int n, k;
    cout << "Enter the size of the array: ";
    cin >> n;

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

    cout << "Enter the value of k: ";
    cin >> k;

    // Check if k is present in the array
    if (sortedArraySearch(arr, k)) {
        cout << "Output: true" << endl;
    } else {
        cout << "Output: false" << endl;
```

```
    }

    return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the sorted array: 10
20
30
40
50
Enter the value of k: 50
Output: true
```

3. **Search Insert Position.**

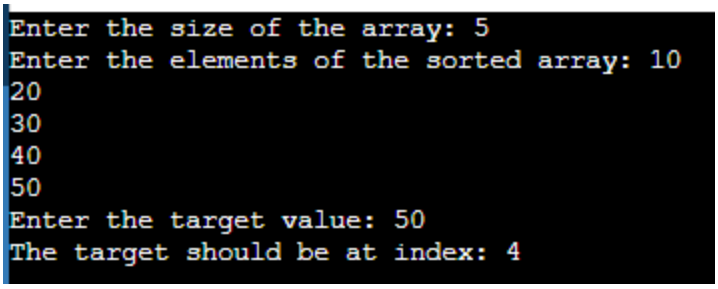
   Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You must write an algorithm with O(log n) runtime complexity.

   Code:

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

// Function to find the search insert position
int searchInsertPosition(const vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1;

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

        if (arr[mid] == target) {
            return mid; // Target found
        } else if (arr[mid] < target) {
            left = mid + 1; // Target is in the right half
        } else {
            right = mid - 1; // Target is in the left half
        }
    }

    return left; // The position where the target should be inserted
```

```
}

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

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

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

    // Find the insert position
    int result = searchInsertPosition(arr, target);

    // Output the result
    cout << "The target should be at index: " << result << endl;

    return 0;
}
```

Output:



Easy

4. **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

Code:

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

// Function to compute the squares of a sorted array and return them in sorted order
vector<int> sortedSquares(const vector<int>& nums) {
    int n = nums.size();
    vector<int> result(n);
    int left = 0, right = n - 1;
    int index = n - 1;

    while (left <= right) {
        int leftSquare = nums[left] * nums[left];
        int rightSquare = nums[right] * nums[right];

        if (leftSquare > rightSquare) {
            result[index] = leftSquare;
            left++;
        } else {
            result[index] = rightSquare;
            right--;
        }
        index--;
    }

    return result;
}

int main() {
    // Input
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

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

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

    // Get the sorted squares
    vector<int> result = sortedSquares(nums);

    // Output the result
    cout << "The sorted squares of the array are: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the sorted array: 10
20
30
40
50
The sorted squares of the array are: 100 400 900 1600 2500
```

5. **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.

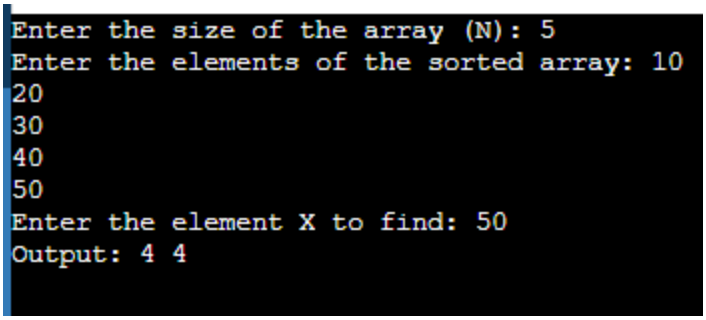
   Code:

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

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

    while (left <= right) {
```

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

        if (v[mid] == X) {
            result = mid; // Found, update result and continue searching left
            right = mid - 1;
        } else if (v[mid] < X) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return result;
}

// Function to find the last occurrence of X
int findLastOccurrence(const vector<int>& v, int X) {
    int left = 0, right = v.size() - 1;
    int result = -1;

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

        if (v[mid] == X) {
            result = mid; // Found, update result and continue searching right
            left = mid + 1;
        } else if (v[mid] < X) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return result;
}

int main() {
    // Input
```

```
    int N, X;
    cout << "Enter the size of the array (N): ";
    cin >> N;

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

    cout << "Enter the element X to find: ";
    cin >> X;

    // Find first and last occurrences
    int first = findFirstOccurrence(v, X);
    int last = findLastOccurrence(v, X);

    // Output the result
    if (first != -1 && last != -1) {
        cout << "Output: " << first << " " << last << endl;
    } else {
        cout << "Output: -1 -1" << endl;
    }

    return 0;
}
```

Output:



6. **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.

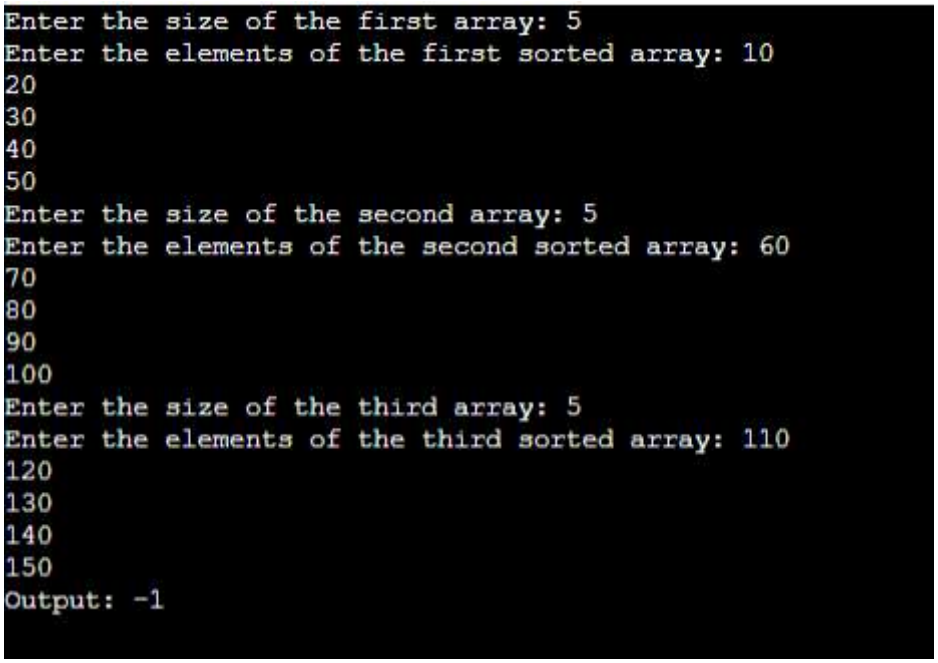
Code:

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

// 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];
}

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:



Medium

7. **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.**

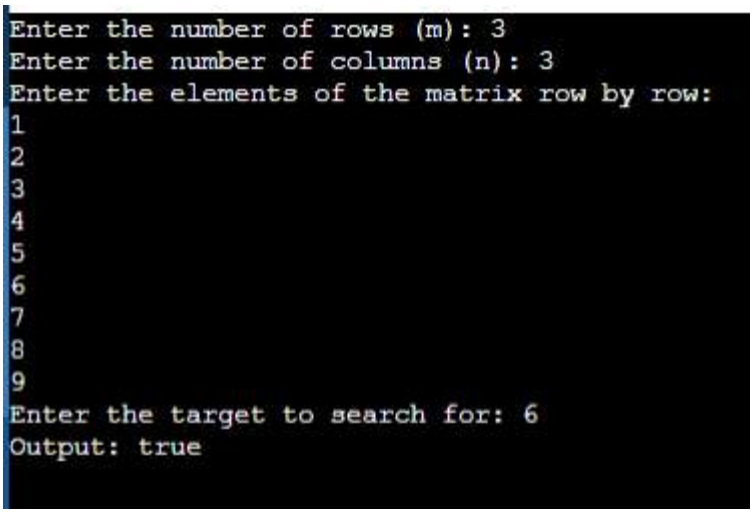
Code:

```
#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) {
            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:



8. **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.**

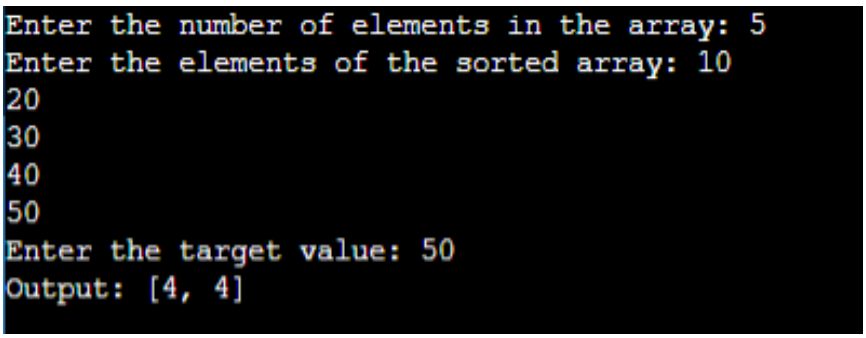
Code:

```
#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;
}

// 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++) {
        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:



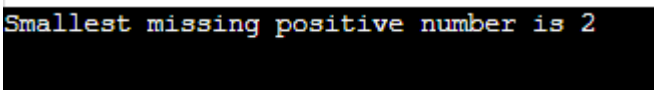
Hard

9. **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:

```
#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]);
        }
    }

    // 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:



10. **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:

```
#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) {
   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: