Day 5

1.Searching a Number

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

int main() {
    int arr[] = {10, 20, 30, 40, 50}; // Sample array
    int size = sizeof(arr) / sizeof(arr[0]); // Calculate size of the array
    int target, found = -1; // target is the number to search

    cout << "Enter number to search: ";
    cin >> target;

    // Linear search
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            found = i;  // Store index if number is found
            break;  // Exit loop if number is found
        }
    }

    if (found != -1)
        cout << "Number found at index " << found << endl;
    else
```

```
        cout << "Number not found in the array." << endl;

    return 0;
}
```

```
Output

Enter number to search: 30
Number found at index 2


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

2.Sorted array Search.

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

// Function for Binary Search
int binarySearch(int arr[], int size, int target) {
    int left = 0, right = size - 1;

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

        // If target is found at mid
        if (arr[mid] == target) {
            return mid; // Return the index of the target
```

```
        }

        // If target is greater, ignore the left half
        if (arr[mid] < target) {
            left = mid + 1;
        }
        // If target is smaller, ignore the right half
        else {
            right = mid - 1;
        }
    }

    return -1; // Target not found
}

int main() {
    // Sorted array
    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
    int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array
    int target;

    // Ask user for the number to search
    cout << "Enter number to search: ";
    cin >> target;
```

```
    // Perform binary search
    int result = binarySearch(arr, size, target);


    // Check the result
    if (result != -1) {
        cout << "Number " << target << " found at index " << result << endl;
    } else {
        cout << "Number " << target << " not found in the array." << endl;
    }


    return 0;
}
```

Output

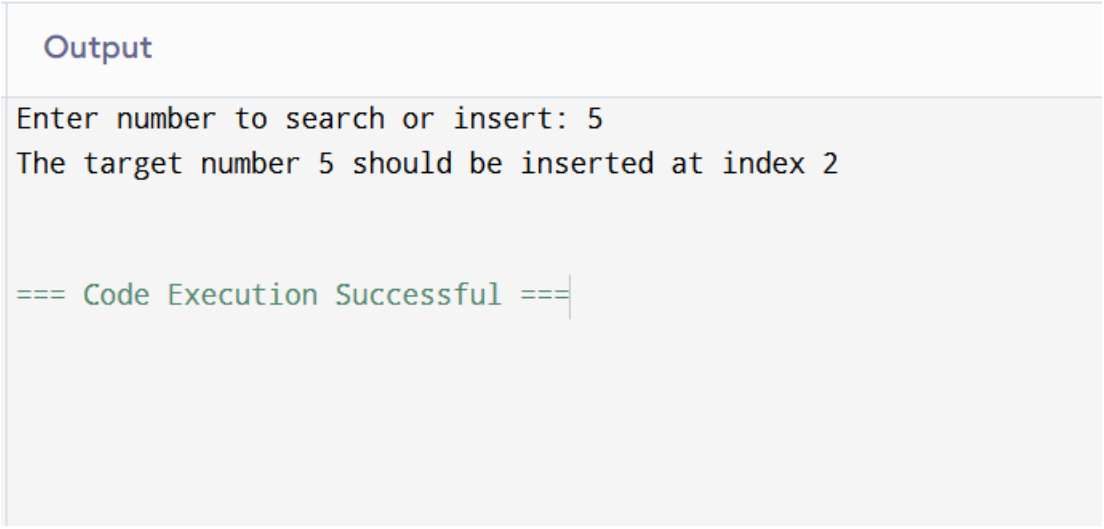
```
Enter number to search: 11
Number 11 found at index 5


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

3. **Search Insert Position.**

```
#include <iostream>

using namespace std;
```

```
// Function to find the insert position
int searchInsertPosition(int arr[], int size, int target) {
    int left = 0, right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2; // Calculate mid to avoid overflow

        // If target is found, return its index
        if (arr[mid] == target) {
            return mid;
        }

        // If target is greater, ignore the left half
        if (arr[mid] < target) {
            left = mid + 1;
        }
        // If target is smaller, ignore the right half
        else {
            right = mid - 1;
        }
    }

    // If not found, left will be the position where target should be inserted
    return left;
}
```

```
int main() {
    // Sorted array
    int arr[] = {1, 3, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]); // Calculate size of array
    int target;

    // Ask user for the target number to search or insert
    cout << "Enter number to search or insert: ";
    cin >> target;

    // Get the insert position
    int position = searchInsertPosition(arr, size, target);

    cout << "The target number " << target << " should be inserted at index " << position << endl;

    return 0;
}
```



4. **Sort Even and Odd Indices Independently.**

```
#include <iostream>
#include <vector>
#include <algorithm> // For sort

using namespace std;

// Function to sort even and odd indexed elements independently
void sortEvenOddIndices(int arr[], int size) {
    // Vectors to hold the even and odd indexed elements
    vector<int> evenElements, oddElements;

    // Separate even and odd indexed elements
    for (int i = 0; i < size; i++) {
        if (i % 2 == 0) {
            evenElements.push_back(arr[i]); // Even index
        } else {
```

```
            oddElements.push_back(arr[i]);  // Odd index
        }
    }

    // Sort both sub-arrays
    sort(evenElements.begin(), evenElements.end());
    sort(oddElements.begin(), oddElements.end());

    // Place the sorted elements back into their respective positions
    int evenIndex = 0, oddIndex = 0;
    for (int i = 0; i < size; i++) {
        if (i % 2 == 0) {
            arr[i] = evenElements[evenIndex++]; // Place sorted even element
        } else {
            arr[i] = oddElements[oddIndex++];  // Place sorted odd element
        }
    }
}

int main() {
    // Example array
    int arr[] = {10, 2, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Display the original array
```

```
    cout << "Original array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Sort even and odd indices independently
    sortEvenOddIndices(arr, size);

    // Display the modified array
    cout << "Array after sorting even and odd indices independently: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output

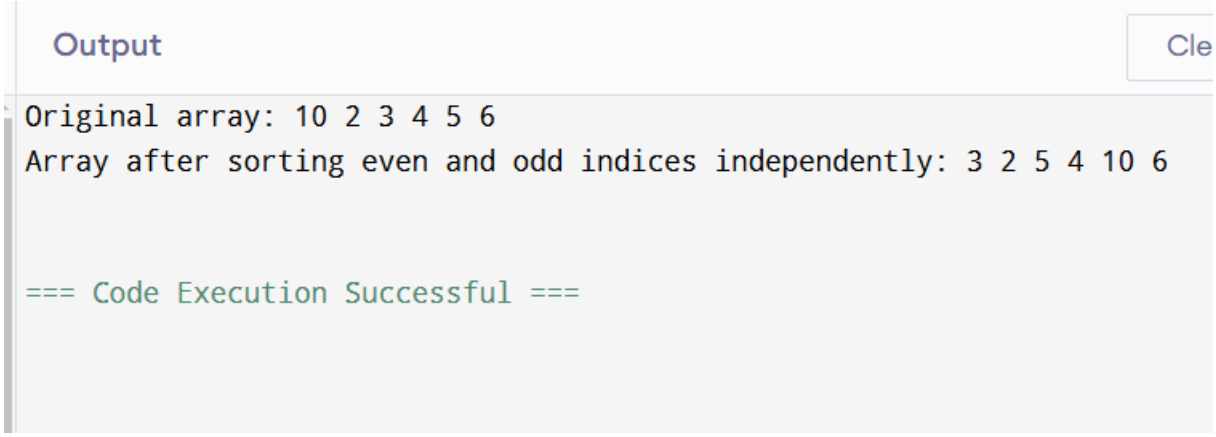
```
Original array: 10 2 3 4 5 6
Array after sorting even and odd indices independently: 3 2 5 4 10 6


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

5. **Search in 2D Matrix.**

```
#include <iostream>
#include <vector>
#include <algorithm> // For sort

using namespace std;

// Function to sort even and odd indexed elements independently
void sortEvenOddIndices(int arr[], int size) {
    // Vectors to hold the even and odd indexed elements
    vector<int> evenElements, oddElements;

    // Separate even and odd indexed elements
    for (int i = 0; i < size; i++) {
        if (i % 2 == 0) {
            evenElements.push_back(arr[i]); // Even index
        } else {
            oddElements.push_back(arr[i]);  // Odd index
        }
    }

    // Sort both sub-arrays
    sort(evenElements.begin(), evenElements.end());
    sort(oddElements.begin(), oddElements.end());
```

```
    // Place the sorted elements back into their respective positions
    int evenIndex = 0, oddIndex = 0;
    for (int i = 0; i < size; i++) {
        if (i % 2 == 0) {
            arr[i] = evenElements[evenIndex++]; // Place sorted even element
        } else {
            arr[i] = oddElements[oddIndex++];  // Place sorted odd element
        }
    }
}

int main() {
    // Example array
    int arr[] = {10, 2, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Display the original array
    cout << "Original array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Sort even and odd indices independently
```

```
    sortEvenOddIndices(arr, size);

    // Display the modified array
    cout << "Array after sorting even and odd indices independently: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



6. Merge k Sorted Lists.

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

// Function to search for a number in an unsorted 2D matrix
bool searchInMatrix(int matrix[][4], int rows, int cols, int target) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
```

```
            if (matrix[i][j] == target) {
                return true; // Target found
            }
        }
    }
    return false; // Target not found
}

int main() {
    // Example unsorted 2D matrix
    int matrix[3][4] = {
        {10, 20, 30, 40},
        {50, 60, 70, 80},
        {90, 100, 110, 120}
    };
    int target;

    cout << "Enter number to search: ";
    cin >> target;

    if (searchInMatrix(matrix, 3, 4, target)) {
        cout << "Number " << target << " found in the matrix." << endl;
    } else {
        cout << "Number " << target << " not found in the matrix." << endl;
    }
```

```
    return 0;
}
```

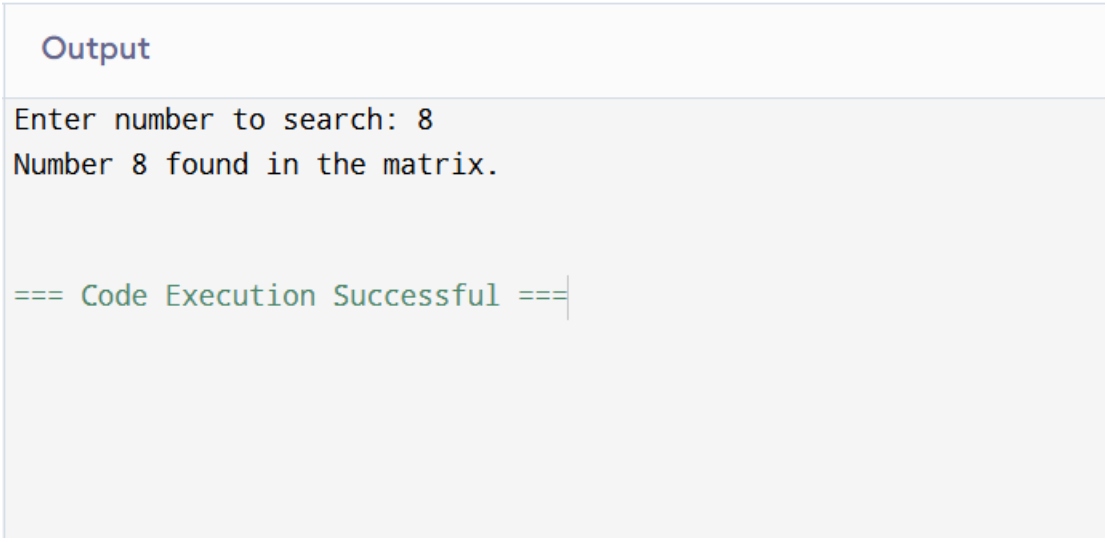
```
Output

Enter number to search: 110
Number 110 found in the matrix.


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

7. Median of Two Sorted Arrays.

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

// Function to search for a number in a sorted 2D matrix
bool searchInSortedMatrix(int matrix[][4], int rows, int cols, int target) {
    int i = 0;  // Start from the first row
    int j = cols - 1;  // Start from the last column

    while (i < rows && j >= 0) {
        if (matrix[i][j] == target) {
            return true; // Target found
        } else if (matrix[i][j] > target) {
            j--;  // Move left
        } else {
            i++;  // Move down
```

```
        }
    }

    return false; // Target not found
}

int main() {
    // Example sorted 2D matrix
    int matrix[3][4] = {
        {1, 4, 7, 11},
        {2, 5, 8, 12},
        {3, 6, 9, 16}
    };
    int target;

    cout << "Enter number to search: ";
    cin >> target;

    if (searchInSortedMatrix(matrix, 3, 4, target)) {
        cout << "Number " << target << " found in the matrix." << endl;
    } else {
        cout << "Number " << target << " not found in the matrix." << endl;
    }

    return 0;
```

}

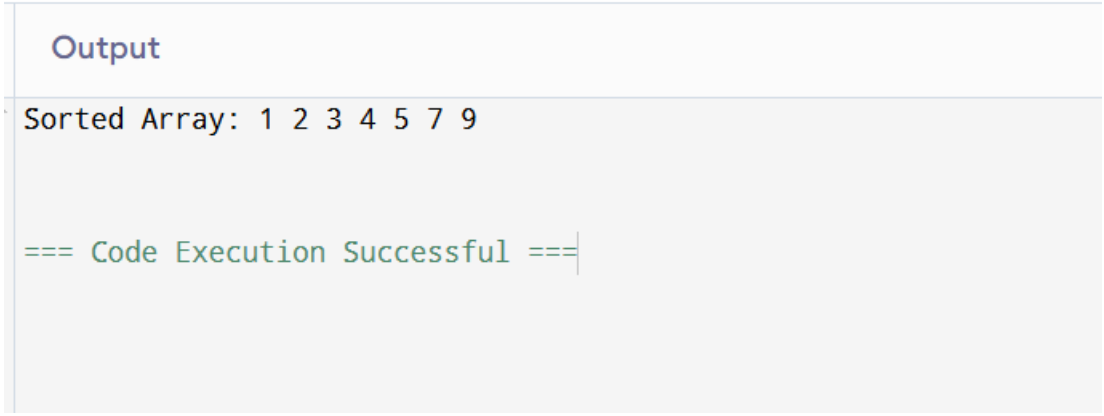


8. Create Sorted Array through Instructions.

```
#include <iostream>
#include <vector>
#include <algorithm>  // For sorting
using namespace std;

// Function to process instructions and create the sorted array
vector<int> createSortedArray(vector<int>& instructions) {
    vector<int> result;

    // Step 1: Process each instruction and add the number to the result array
    for (int instruction : instructions) {
        result.push_back(instruction);
    }

    // Step 2: Sort the array in ascending order
```

```
    sort(result.begin(), result.end());

    return result;
}

// Function to print the array
void printArray(const vector<int>& arr) {
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;
}

int main() {
    // Example of instructions
    vector<int> instructions = {4, 1, 7, 3, 9, 2, 5};

    // Create the sorted array from instructions
    vector<int> sortedArray = createSortedArray(instructions);

    // Print the sorted array
    cout << "Sorted Array: ";
    printArray(sortedArray);

    return 0;
```

}



9. Kth Smallest Product of Two Sorted Arrays.

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

// Function to find the k-th smallest product of two sorted arrays
int kthSmallestProduct(vector<int>& A, vector<int>& B, int k) {
    int n = A.size(), m = B.size();

    // Min-heap to store the current smallest product along with the indices
    priority_queue<tuple<int, int, int>, vector<tuple<int, int, int>>,
greater<tuple<int, int, int>>> pq;

    // Initialize the min-heap with the first row (A[0] * B[i] for all i)
    for (int j = 0; j < m; ++j) {
        pq.push({A[0] * B[j], 0, j});  // (product, index in A, index in B)
```

```
    }

    int count = 0;
    while (!pq.empty()) {
        auto [prod, i, j] = pq.top();
        pq.pop();

        count++;
        if (count == k) {
            return prod;  // Return the k-th smallest product
        }

        // Move to the next element in array A (keeping the current element in array B)
        if (i + 1 < n) {
            pq.push({A[i + 1] * B[j], i + 1, j});
        }
    }

    return -1;  // If no result is found, return -1 (this should not happen for valid k)
}

int main() {
    vector<int> A = {1, 7, 11};
    vector<int> B = {2, 4, 6};
    int k = 5;
```

```
    cout << "The " << k << "-th smallest product is: " << kthSmallestProduct(A, B, k) << endl;


    return 0;

}
```

```
Output

The 5-th smallest product is: 22


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

10. Smallest Positive Missing Number

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


int smallestMissingPositive(vector<int>& nums) {

    int n = nums.size();


    // Step 1: Rearrange the array

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

        // Swap the number to its correct position if it's within the range [1, n]
```

```
        while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i]) {
            swap(nums[i], nums[nums[i] - 1]);
        }
    }

    // Step 2: Find the first index where the value is not i + 1
    for (int i = 0; i < n; i++) {
        if (nums[i] != i + 1) {
            return i + 1;
        }
    }

    // Step 3: If all numbers are in the correct position, return n + 1
    return n + 1;
}

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

Output

```
The smallest missing positive number is: 2


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