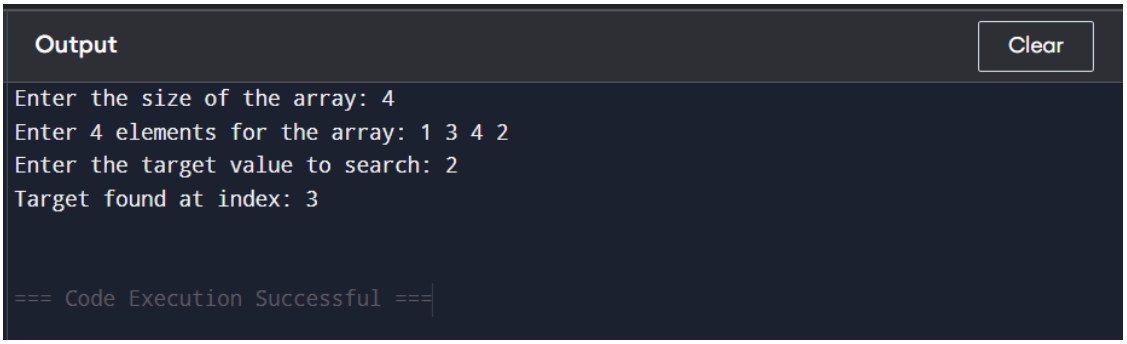
DOMAIN WINTER WINNING CAMP

Student Name: Abhay Bansal
UID: 22BCS15306
Branch: BE-CSE
Section/Group: 620-A
Semester: 5th
Date of Performance: 26/12/24

1. Implement linear search to find the target value.

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

int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}

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

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

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

    int index = linearSearch(arr, size, target);

    if (index != -1) {
        cout << "Target found at index: " << index << endl;
    } else {
        cout << "Target not found." << endl;
    }
```

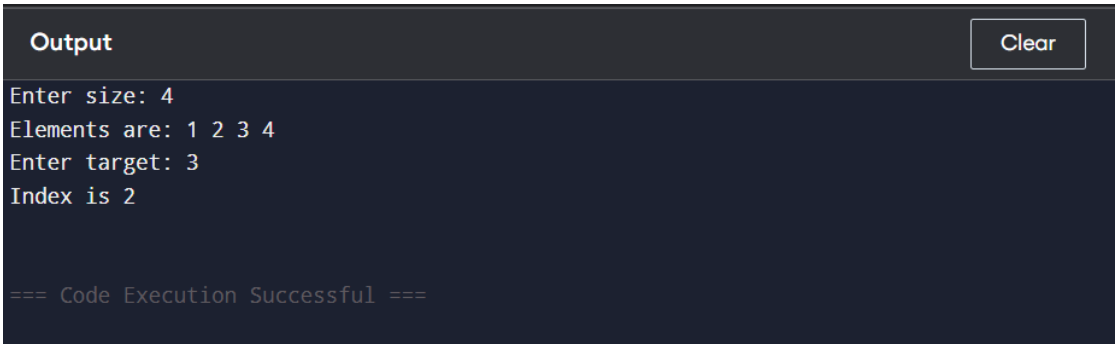
```
    return 0;
}
```



2. Implement binary search to find the target value and return its index

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

int binarySearch(int arr[], int low, int high, int target) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid;
        }
        if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

int main() {
    int size;
    cout<<"Enter size: ";
    cin >> size;
    int arr[size];
    cout<<"Elements are: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    int target;
    cout<<"Enter target: ";
    cin >> target;
```

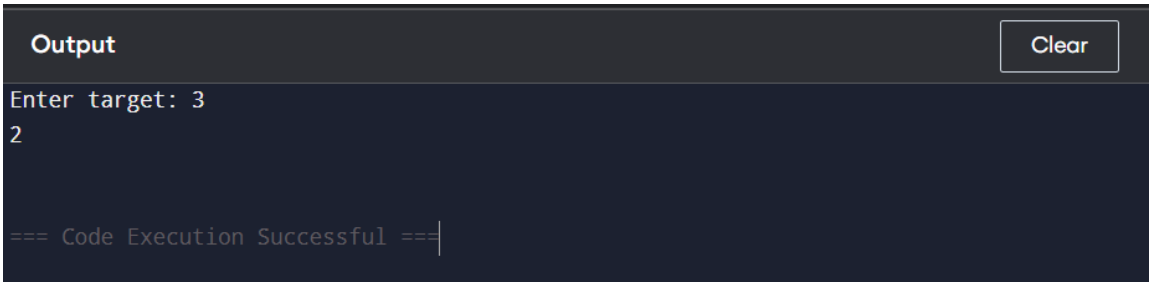
```
    int index = binarySearch(arr, 0, size - 1, target);
    cout << "Index is "<< index << endl;

    return 0;
}
```



3. Implement binary search to find the first occurrence in sorted array [1, 2, 3, 4, 5, 6]

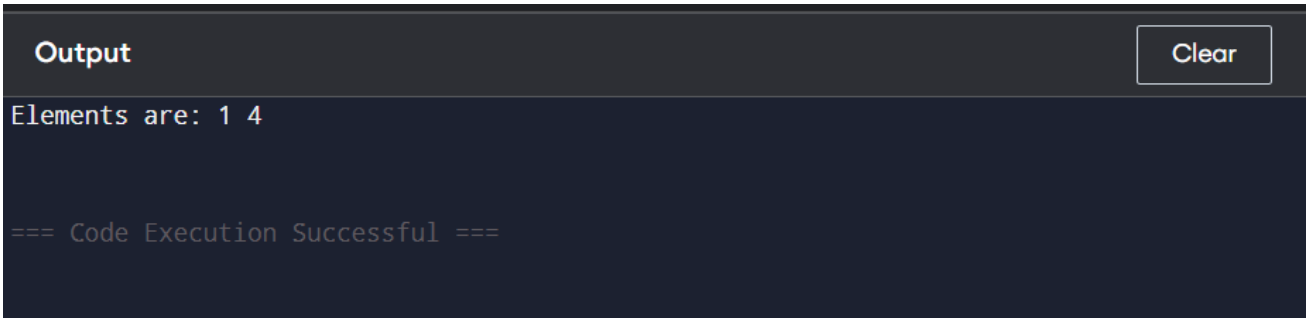
```
#include <iostream>
using namespace std;

int binarySearchFirstOccurrence(int arr[], int size, int target) {
    int low = 0, high = size - 1, result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            result = mid;
            high = mid - 1;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return result;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target;
    cout<<"Enter target: ";
    cin >> target;

    int index = binarySearchFirstOccurrence(arr, size, target);
    cout << index << endl;
```

```
    return 0;
}
```



4. Implement binary search to find the element that occurs only once

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

void findUniqueElements(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        if ((i == 0 || arr[i] != arr[i - 1]) && (i == size - 1 || arr[i] != arr[i + 1])) {
            cout << arr[i] << " ";
        }
    }
    cout << endl;
}

int main() {
    int arr[] = {1, 2, 2, 2, 3, 3, 4};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout<<"Elements are: ";

    findUniqueElements(arr, size);

    return 0;
}
```



5. Implement bubble sort

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

void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void displayArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    displayArray(arr, size);

    bubbleSort(arr, size);

    cout << "Sorted array: ";
    displayArray(arr, size);

    return 0;
}
```

```
Output                                                    Clear
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90


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

6. Given an integer array numbers sorted in non decreasing order return the array square of each number in non decreasing order [-4, -1 ,0 ,3, 10]

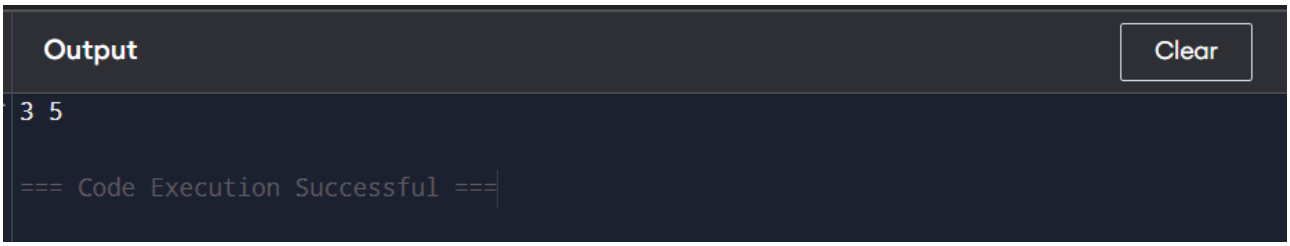
```
#include <iostream>
#include <cmath>
using namespace std;

void sortedSquares(int numbers[], int size, int result[]) {
    int left = 0, right = size - 1;
    int index = size - 1;

    while (left <= right) {
        int leftSquare = numbers[left] * numbers[left];
        int rightSquare = numbers[right] * numbers[right];
        if (leftSquare > rightSquare) {
            result[index--] = leftSquare;
            left++;
        } else {
            result[index--] = rightSquare;
            right--;
        }
    }
}
void displayArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int numbers[] = {-4, -1, 0, 3, 10};
    int size = sizeof(numbers) / sizeof(numbers[0]);
    int result[size];
    sortedSquares(numbers, size, result);
    cout << "Sorted squares: ";
    displayArray(result, size);

    return 0;
}
```

```
Output                                                    Clear
Sorted squares: 0 1 9 16 100


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

7. 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.

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

int* findCommonElements(int arr1[], int size1, int arr2[], int size2, int arr3[], int size3, int&
resultSize) {
    int* result = new int[std::min(size1, std::min(size2, size3))];
    int i = 0, j = 0, k = 0;
    resultSize = 0;

    while (i < size1 && j < size2 && k < size3) {
        if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {
            result[resultSize++] = arr1[i];
            i++, j++, k++;
        } else if (arr1[i] < arr2[j]) {
            i++;
        } else if (arr2[j] < arr3[k]) {
            j++;
        } else {
            k++;
        }
    }

    if (resultSize == 0) {
        delete[] result;
        return nullptr;
    }

    return result;
}

int main() {
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {2, 3, 5, 7};
    int arr3[] = {3, 5, 6, 8};
    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    int size3 = sizeof(arr3) / sizeof(arr3[0]);
    int resultSize = 0;

    int* result = findCommonElements(arr1, size1, arr2, size2, arr3, size3, resultSize);
    if (result) {
        for (int i = 0; i < resultSize; i++) {
            cout << result[i] << " ";
        }
        delete[] result;
```

```
    } else {
        cout << "-1" << endl;
    }

    return 0;
}
```



8. 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.

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

pair<int, int> findFirstAndLastOccurrences(int arr[], int size, int X) {
    int left = 0, right = size - 1;
    int firstOccurrence = -1, lastOccurrence = -1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == X) {
            firstOccurrence = mid;
            right = mid - 1;
        } else if (arr[mid] < X) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    left = 0, right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == X) {
            lastOccurrence = mid;
            left = mid + 1;
        } else if (arr[mid] < X) {
            left = mid + 1;
        } else {
```

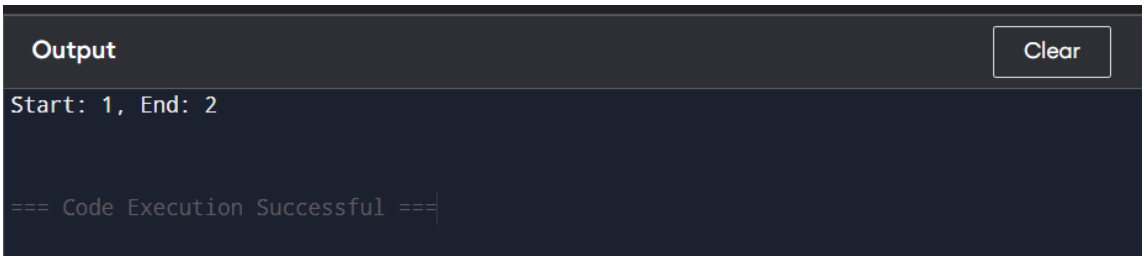
```
            right = mid - 1;
        }
    }

    return {firstOccurrence, lastOccurrence};
}

int main() {
    int arr[] = {1, 2, 2, 3, 4, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);
    int X = 2;

    pair<int, int> result = findFirstAndLastOccurrences(arr, size, X);

    if (result.first != -1) {
        cout << "First occurrence: " << result.first << endl;
        cout << "Last occurrence: " << result.second << endl;
    } else {
        cout << "-1 -1" << endl;
    }

    return 0;
}
```

9 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.

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

pair<int, int> searchRange(int arr[], int size, int target) {
    int left = 0, right = size - 1;
    int firstOccurrence = -1, lastOccurrence = -1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
```

```
            firstOccurrence = mid;
            right = mid - 1;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    left = 0, right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            lastOccurrence = mid;
            left = mid + 1;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return {firstOccurrence, lastOccurrence};
}

int main() {
    int arr[] = {1, 2, 2, 3, 4, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 2;

    pair<int, int> result = searchRange(arr, size, target);

    cout << "Start: " << result.first << ", End: " << result.second << endl;

    return 0;
}
```



10. You are given an integer array arr[]. Your task is to find the smallest positive number missing from the array.

.

```
#include <iostream>
#include <climits>

using namespace std;

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

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

    return size + 1;
}

int main() {
    int arr[] = {3, 4, -1, 1};
    int size = sizeof(arr) / sizeof(arr[0]);

    int result = findSmallestMissingPositive(arr, size);

    cout << "The smallest missing positive integer is: " << result << endl;

    return 0;
}
```