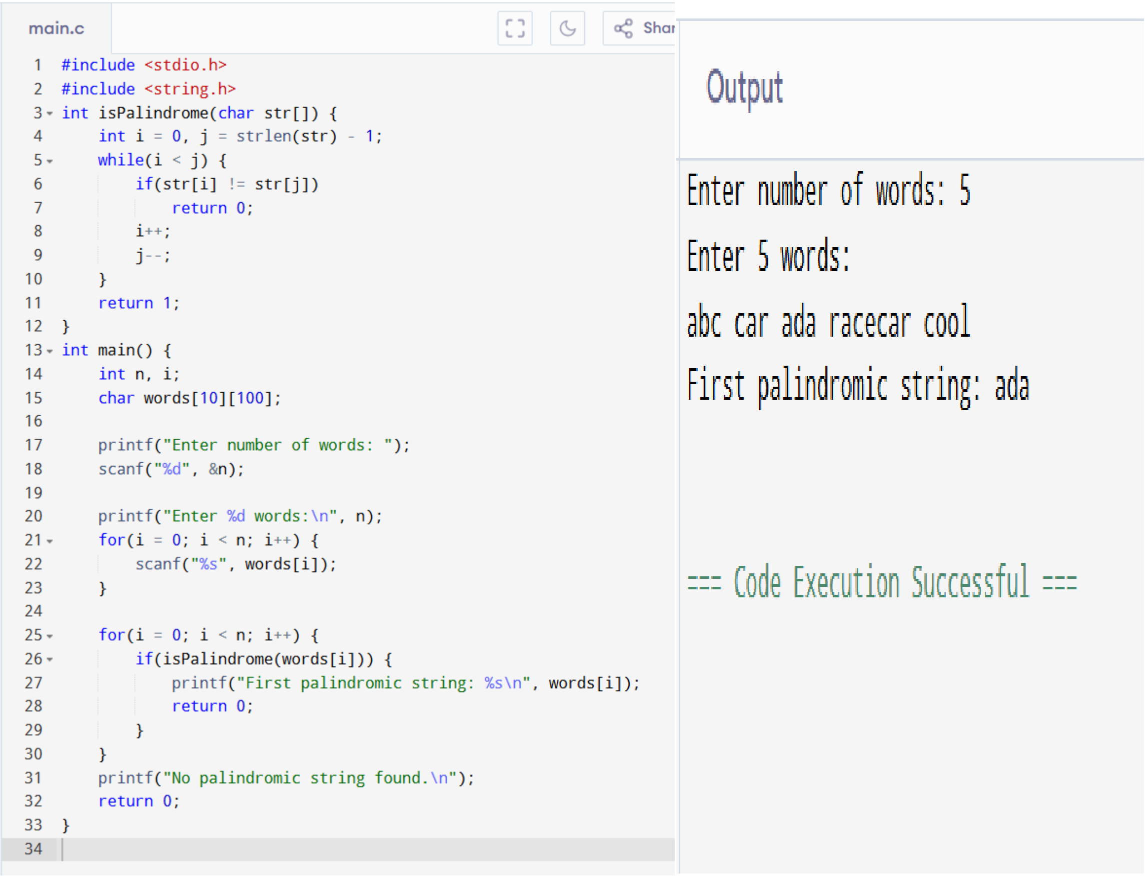
TOPIC 1: INTRODUCTION

1.First Palindromic String in an Array

Aim: To find and return the first palindromic string in a given list of words.

Algorithm:

1. Start with a list of strings.
2. Traverse each word in the list.
3. For each word, check if it reads the same forward and backward.
4. If found, return that word immediately.
5. If no palindromic string exists, return an empty string.



```c
#include <stdio.h>
#include <string.h>
int isPalindrome(char str[]) {
    int i = 0, j = strlen(str) - 1;
    while(i < j) {
        if(str[i] != str[j])
            return 0;
        i++;
        j--;
    }
    return 1;
}
int main() {
    int n, i;
    char words[10][100];

    printf("Enter number of words: ");
    scanf("%d", &n);

    printf("Enter %d words:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%s", words[i]);
    }

    for(i = 0; i < n; i++) {
        if(isPalindrome(words[i])) {
            printf("First palindromic string: %s\n", words[i]);
            return 0;
        }
    }
    printf("No palindromic string found.\n");
    return 0;
}
```

```
Enter number of words: 5
Enter 5 words:
abc car ada racecar cool
First palindromic string: ada


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

Input:

Enter no.of words : 5

Enter 5 words : abc car ada racecar cool

Output:

First palindromic string : ada

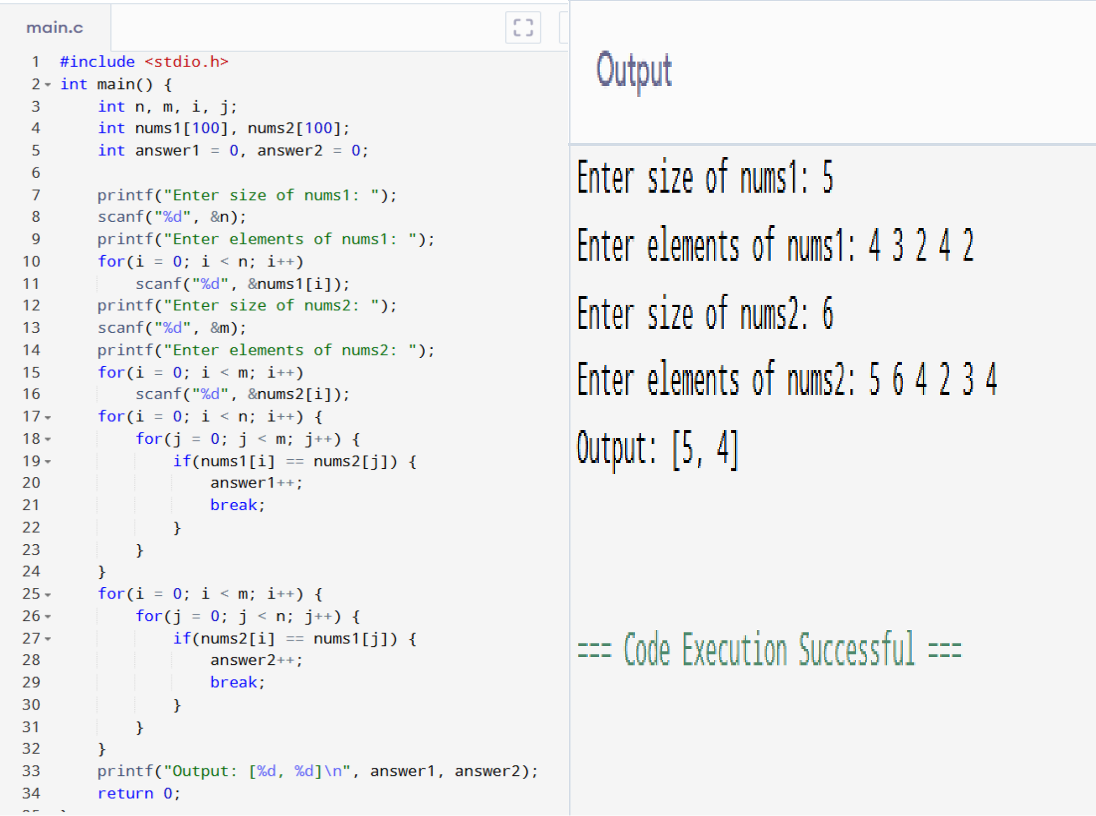
Result: Successfully identified and returned the first palindromic string from the given array.

2. Common Elements Count Between Two Arrays

Aim: To count the number of indices where elements of one array appear in another.

Algorithm:

1. Take two integer arrays nums1 and nums2.
2. For each element in nums1, check if it exists in nums2 and count them.
3. Similarly, for each element in nums2, check if it exists in nums1.
4. Store both counts in a list [answer1, answer2].
5. Display the result.



```c
#include <stdio.h>
int main() {
    int n, m, i, j;
    int nums1[100], nums2[100];
    int answer1 = 0, answer2 = 0;

    printf("Enter size of nums1: ");
    scanf("%d", &n);
    printf("Enter elements of nums1: ");
    for(i = 0; i < n; i++)
        scanf("%d", &nums1[i]);
    printf("Enter size of nums2: ");
    scanf("%d", &m);
    printf("Enter elements of nums2: ");
    for(i = 0; i < m; i++)
        scanf("%d", &nums2[i]);
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            if(nums1[i] == nums2[j]) {
                answer1++;
                break;
            }
        }
    }
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++) {
            if(nums2[i] == nums1[j]) {
                answer2++;
                break;
            }
        }
    }
    printf("Output: [%d, %d]\n", answer1, answer2);
    return 0;
```

```
Enter size of nums1: 5
Enter elements of nums1: 4 3 2 4 2
Enter size of nums2: 6
Enter elements of nums2: 5 6 4 2 3 4
Output: [5, 4]


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

Input:

Enter size of nums1: 5
Enter elements of nums1: 4 3 2 4 2
Enter size of nums 2: 6
Enter elements of nums 2 : 5 6 4 2 3 4

Output:

[5,4]

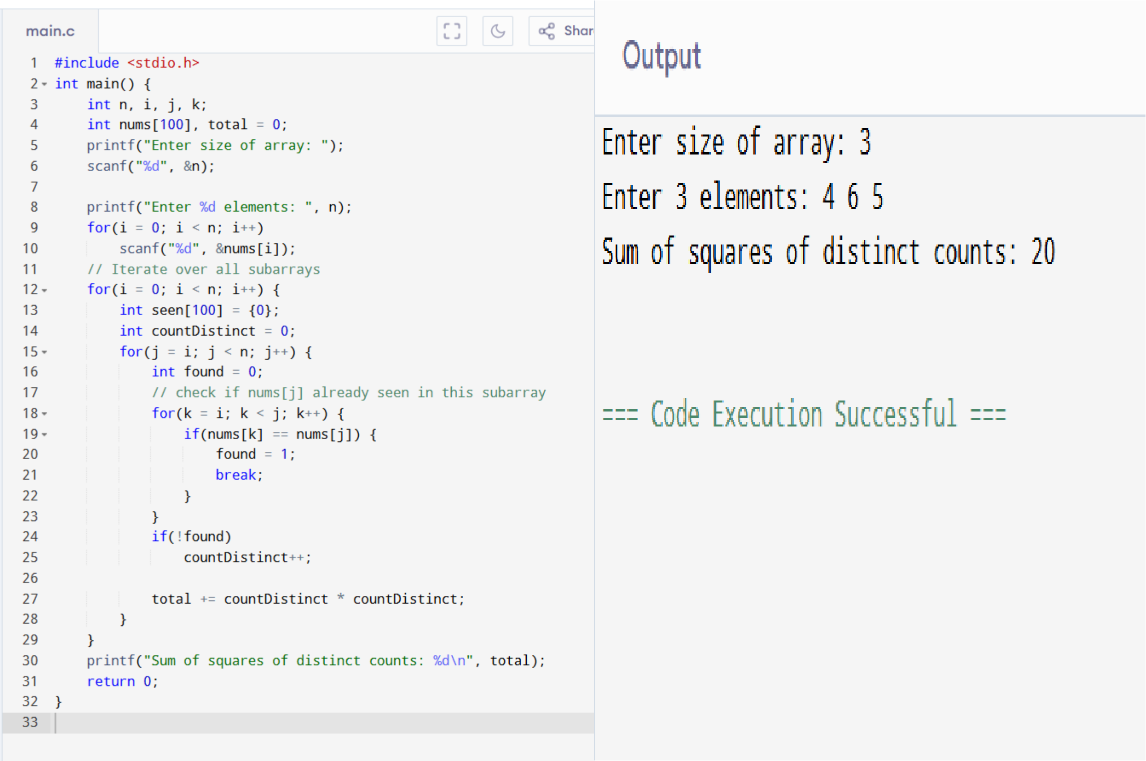
Result: Displayed the count of common elements existing between two arrays in both directions.

3.Sum of Squares of Distinct Counts of Subarrays

Aim: To calculate the sum of squares of distinct element counts for all possible subarrays of an array.

Algorithm:

1. Start with the input array nums.
2. Generate all possible subarrays.
3. For each subarray, find the number of distinct elements.
4. Square this count and add it to a total sum.
5. After all subarrays are processed, display the total sum.



```c
#include <stdio.h>
int main() {
    int n, i, j, k;
    int nums[100], total = 0;
    printf("Enter size of array: ");
    scanf("%d", &n);

    printf("Enter %d elements: ", n);
    for(i = 0; i < n; i++)
        scanf("%d", &nums[i]);
    // Iterate over all subarrays
    for(i = 0; i < n; i++) {
        int seen[100] = {0};
        int countDistinct = 0;
        for(j = i; j < n; j++) {
            int found = 0;
            // check if nums[j] already seen in this subarray
            for(k = i; k < j; k++) {
                if(nums[k] == nums[j]) {
                    found = 1;
                    break;
                }
            }
            if(!found)
                countDistinct++;

            total += countDistinct * countDistinct;
        }
    }
    printf("Sum of squares of distinct counts: %d\n", total);
    return 0;
}
```

```
Enter size of array: 3
Enter 3 elements: 4 6 5
Sum of squares of distinct counts: 20


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

Input:
Enter size of array : 3
Enter 3 elements: 4 6 5
Output:
Sum of squares of distinct counts : 20

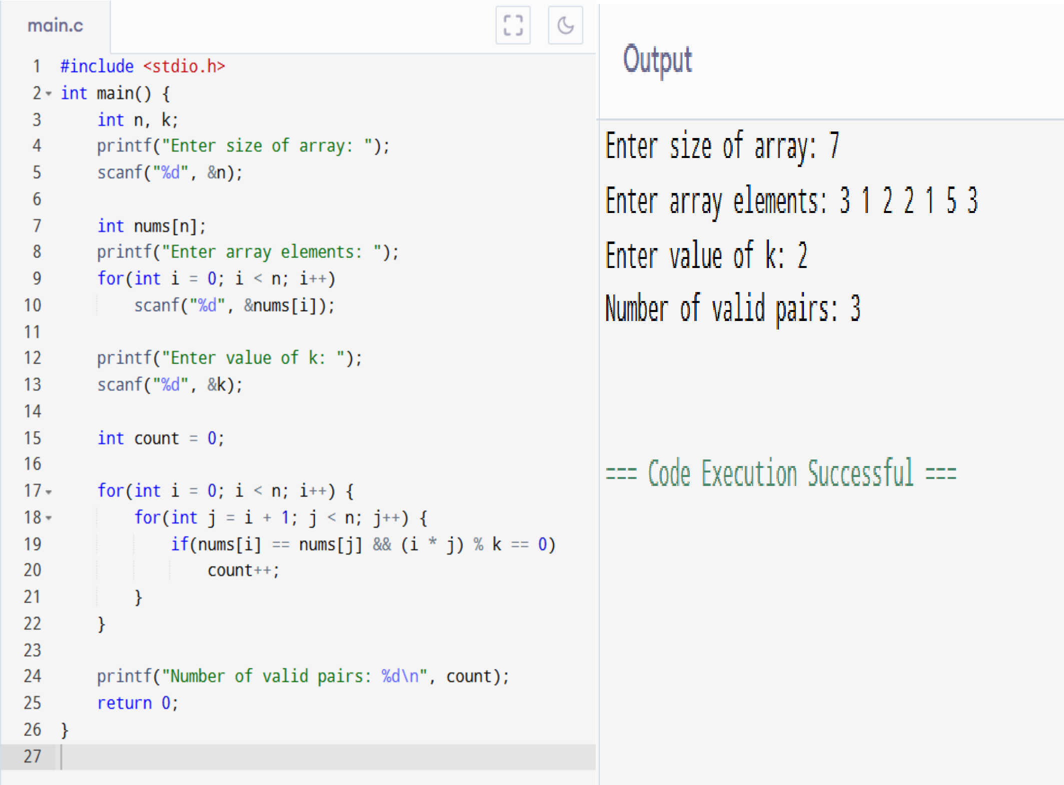
Result: Successfully computed the sum of squared distinct counts for all subarrays.

4.Count of Pairs with Equal Elements and Divisible Product

Aim: To count pairs (i, j) where nums[i] == nums[j] and (i * j) is divisible by k.

Algorithm:

1. Take an integer array nums and integer k.
2. Initialize a counter to 0.
3. For each pair (i, j) where i < j:
4. Check if nums[i] == nums[j] and (i * j) is divisible by k.
5. If both conditions are true, increase the count.
6. Display the count.



```c
#include <stdio.h>
int main() {
    int n, k;
    printf("Enter size of array: ");
    scanf("%d", &n);

    int nums[n];
    printf("Enter array elements: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &nums[i]);

    printf("Enter value of k: ");
    scanf("%d", &k);

    int count = 0;

    for(int i = 0; i < n; i++) {
        for(int j = i + 1; j < n; j++) {
            if(nums[i] == nums[j] && (i * j) % k == 0)
                count++;
        }
    }

    printf("Number of valid pairs: %d\n", count);
    return 0;
}
```

```
Enter size of array: 7
Enter array elements: 3 1 2 2 1 5 3
Enter value of k: 2
Number of valid pairs: 3


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

Input:

Enter size of array : 7

Enter array elements : 3 1 2 2 1 5 3

Enter value of K : 2

Output:

Number of valid pairs : 3

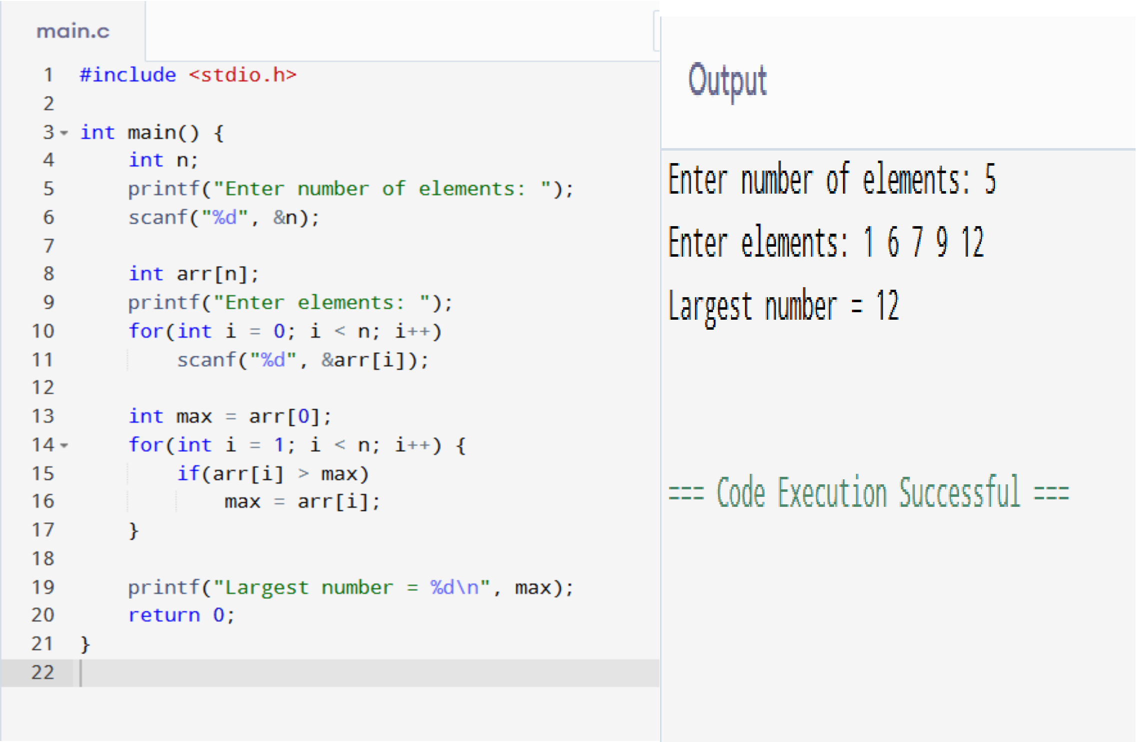
Result: Displayed the total number of valid pairs satisfying both equality and divisibility conditions.

5.Find the Maximum Element in a List

Aim: To find the largest number in a given list using minimal time complexity.

Algorithm:

1. Start with an array of integers.
2. Initialize a variable max with the first element.
3. Traverse through the array comparing each element with max.
4. Update max if a larger number is found.
5. Display max as the largest element.



```c
#include <stdio.h>

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter elements: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

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

    printf("Largest number = %d\n", max);
    return 0;
}
```

```
Enter number of elements: 5
Enter elements: 1 6 7 9 12
Largest number = 12


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

Input:

Enter no.of elements : 5

Enter elements : 1 6 7 9 12

Output:

Largest number : 12

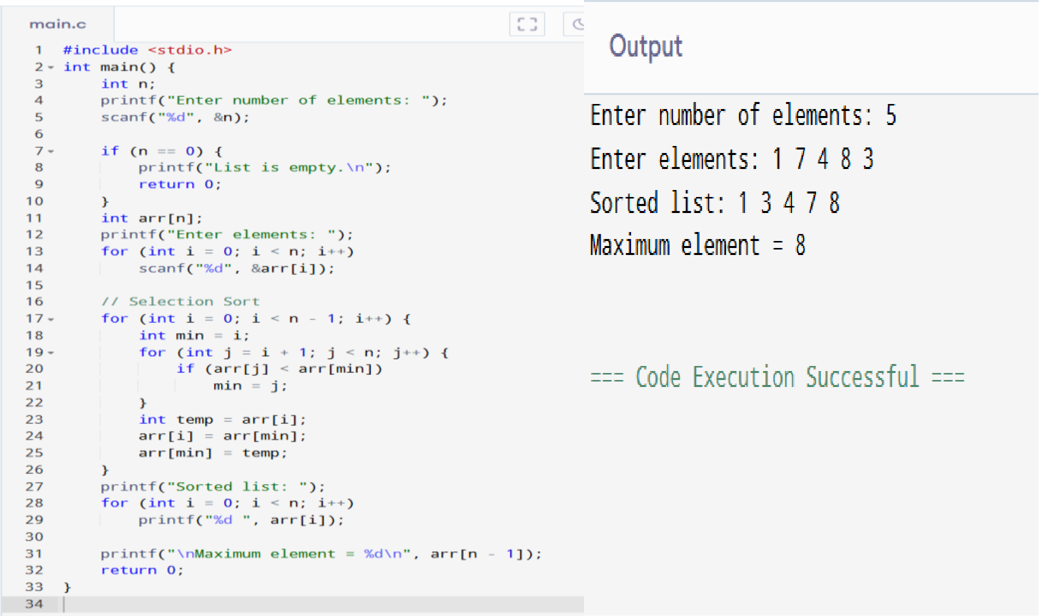
Result: Displayed the maximum element from the given list.

6. Sorting and Finding the Maximum Element

Aim: To sort a list of numbers efficiently and find the maximum element after sorting.

Algorithm:

1. Start with an input list of numbers.
2. If the list is empty, display a message.
3. Use an efficient sorting algorithm (like Merge Sort or Quick Sort) to sort the list.
4. Select the last element from the sorted list as the maximum.
5. Display the sorted list and the maximum element.



```c
#include <stdio.h>
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    if (n == 0) {
        printf("List is empty.\n");
        return 0;
    }
    int arr[n];
    printf("Enter elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    // Selection Sort
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min])
                min = j;
        }
        int temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
    printf("Sorted list: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\nMaximum element = %d\n", arr[n - 1]);
    return 0;
}
```

```
Enter number of elements: 5
Enter elements: 1 7 4 8 3
Sorted list: 1 3 4 7 8
Maximum element = 8


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

Input:

Enter no.of elements : 5

Enter elements : 1 7 4 8 3

Output:

Sorted list : 1 3 4 7 8

Maximum element : 8

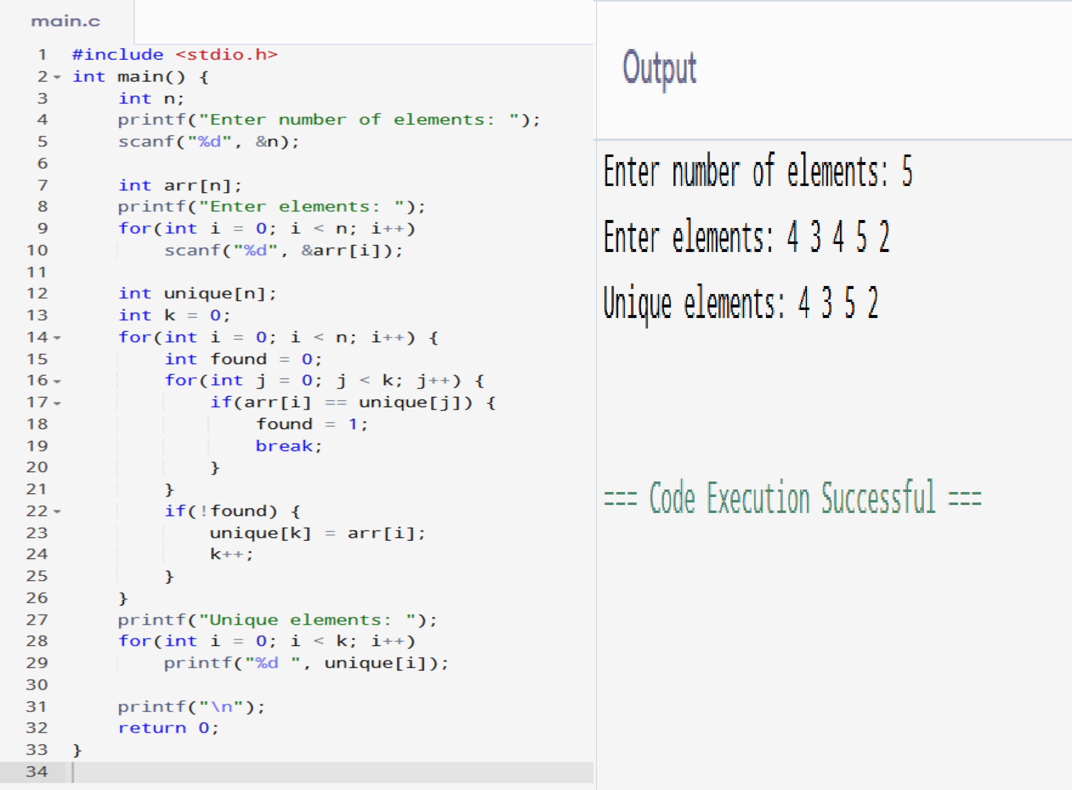
Result: Successfully sorted the list and found the maximum value.

7.Extract Unique Elements from a List

Aim: To generate a new list containing only unique elements from an input list.

Algorithm:

1. Start with an input list of integers.
2. Initialize an empty list unique_list.
3. For each element in the input list, if it is not in unique_list, append it.
4. Display the unique_list.
5. Analyze space complexity as O(n) due to storage of unique elements.



```c
#include <stdio.h>
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter elements: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    int unique[n];
    int k = 0;
    for(int i = 0; i < n; i++) {
        int found = 0;
        for(int j = 0; j < k; j++) {
            if(arr[i] == unique[j]) {
                found = 1;
                break;
            }
        }
        if(!found) {
            unique[k] = arr[i];
            k++;
        }
    }
    printf("Unique elements: ");
    for(int i = 0; i < k; i++)
        printf("%d ", unique[i]);

    printf("\n");
    return 0;
}
```

```
Enter number of elements: 5
Enter elements: 4 3 4 5 2
Unique elements: 4 3 5 2


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

Input:
Enter no.of elements : 5
Enter elements : 4 3 4 5 2
Output:
Unique elements : 4 3 5 2

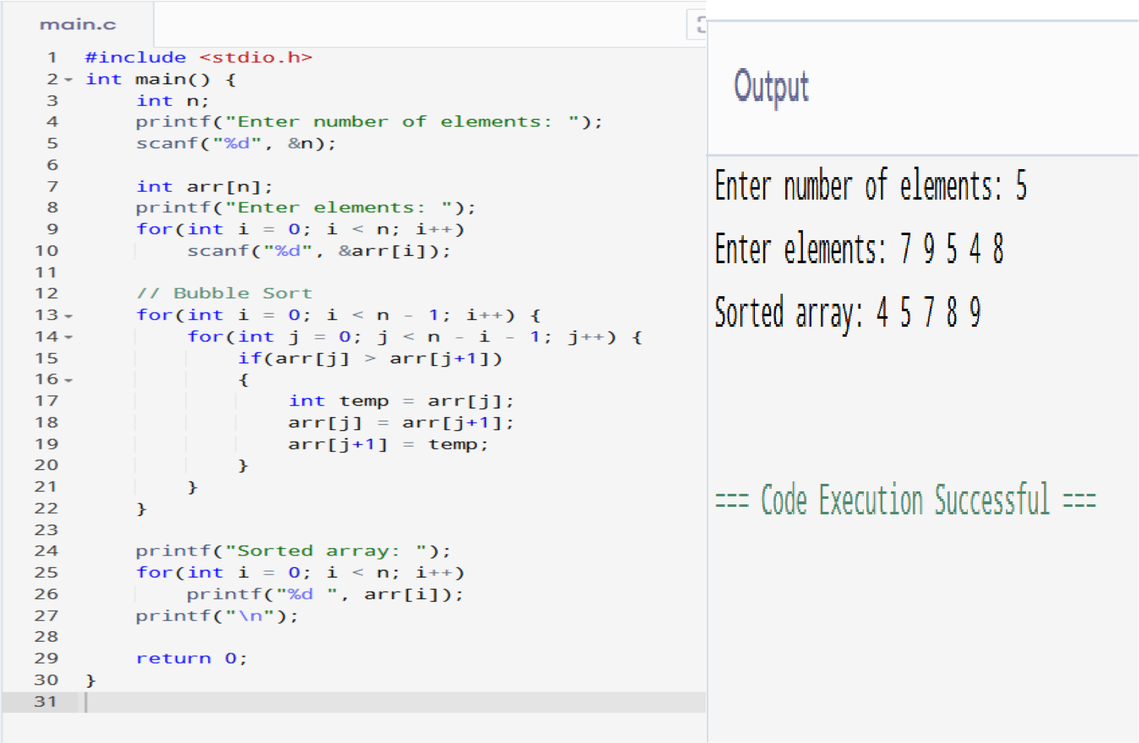
Result: Displayed the list containing only unique elements.

8.Bubble Sort Implementation

Aim: To sort an array of integers using the bubble sort algorithm and analyze its time complexity.

Algorithm:

1. Take an array of integers.
2. Repeat for each element in the array: compare adjacent elements and swap if needed.
3. Continue until the array is fully sorted.
4. Time Complexity: O(n^2) in the worst case.



```c
#include <stdio.h>
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter elements: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    // Bubble Sort
    for(int i = 0; i < n - 1; i++) {
        for(int j = 0; j < n - i - 1; j++) {
            if(arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }

    printf("Sorted array: ");
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}

```

```
Enter number of elements: 5
Enter elements: 7 9 5 4 8
Sorted array: 4 5 7 8 9


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

Input:

Enter no.of elements : 5

Enter elements : 7 9 5 4 8

Output:

Sorted array : 4 5 7 8 9

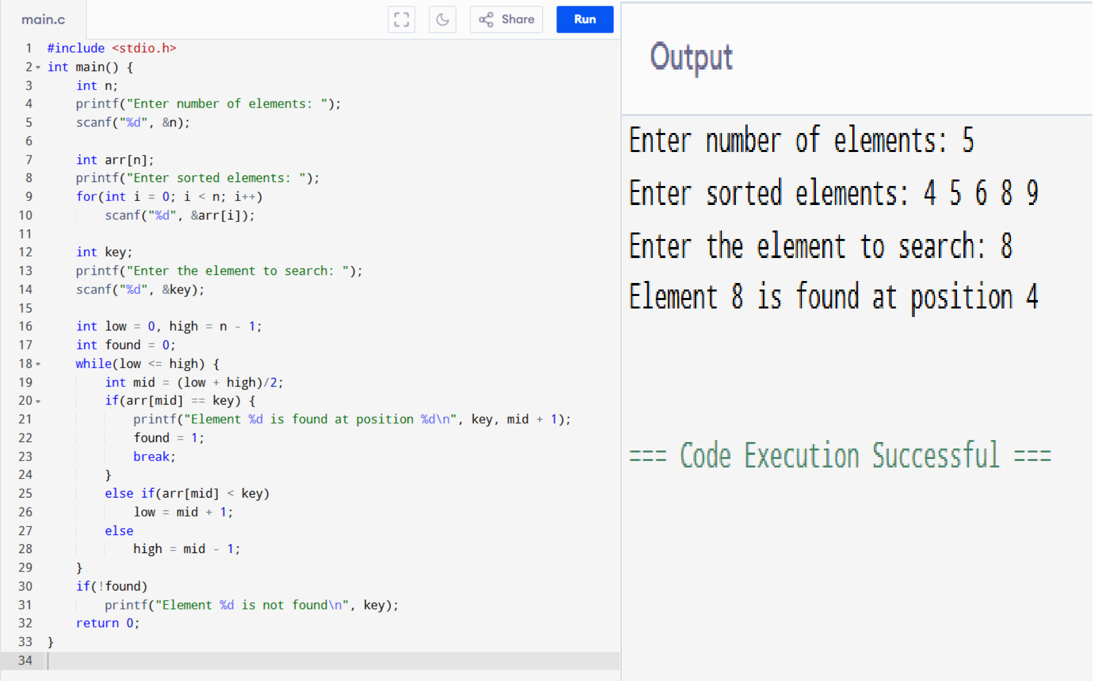
Result: Successfully sorted the array using bubble sort technique.

9.Binary Search on a Sorted Array

Aim: To check whether a given key element exists in a sorted array using binary search.

Algorithm:

1. Take a sorted array and a key to search.
2. Initialize low = 0, high = n-1.
3. Repeat until low <= high: find mid = (low + high) // 2.
4. If arr[mid] == key, return position.
5. If arr[mid] > key, search the left half else search the right half.
6. If not found, display message.
7. Time Complexity: O(log n).



```c
#include <stdio.h>
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter sorted elements: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    int key;
    printf("Enter the element to search: ");
    scanf("%d", &key);

    int low = 0, high = n - 1;
    int found = 0;
    while(low <= high) {
        int mid = (low + high)/2;
        if(arr[mid] == key) {
            printf("Element %d is found at position %d\n", key, mid + 1);
            found = 1;
            break;
        }
        else if(arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    if(!found)
        printf("Element %d is not found\n", key);
    return 0;
}
```

```
Enter number of elements: 5
Enter sorted elements: 4 5 6 8 9
Enter the element to search: 8
Element 8 is found at position 4


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

Input:

Enter no.of elements : 5

Enter sorted elements : 4 5 6 8 9

Enter the element to search : 8

Output:

Element 8 is found at position 4

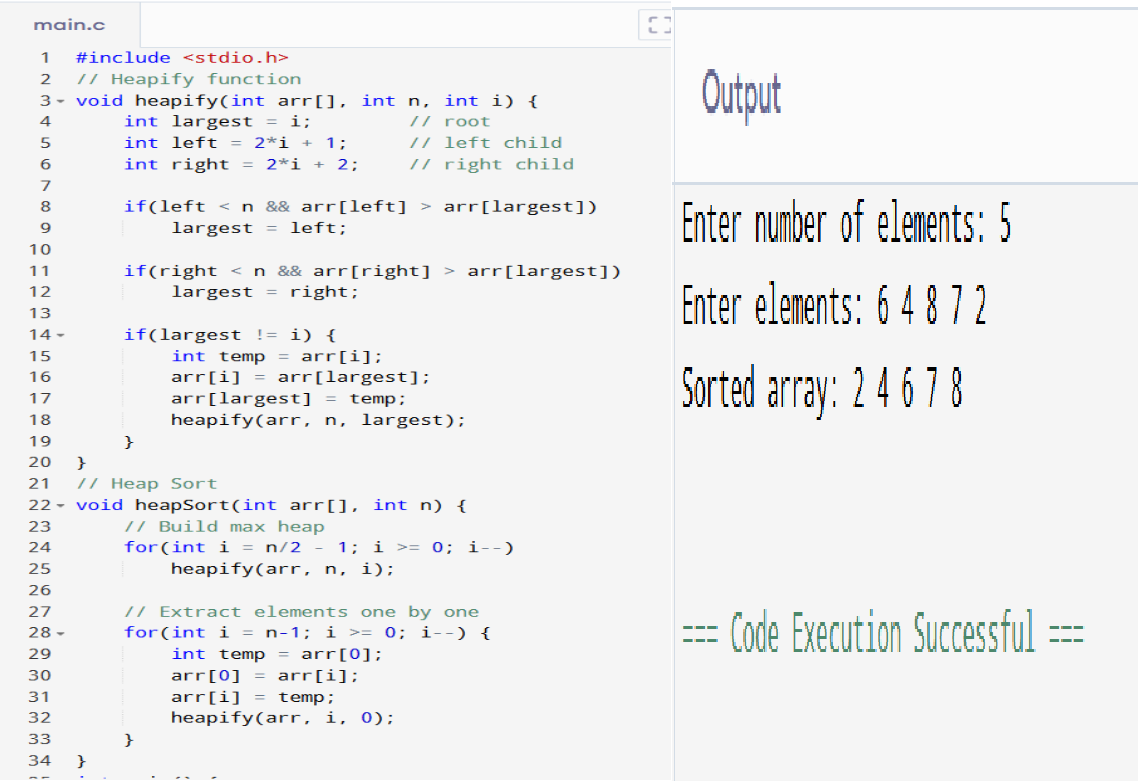
Result: Displayed whether the given element exists in the array or not.

10. Heap Sort Implementation

Aim: To sort an array of integers in ascending order using heap sort in O(n log n) time.

Algorithm:

1. Build a max heap from the input array.
2. Extract the largest element (root) and place it at the end of the array.
3. Reduce the heap size and heapify again.
4. Repeat until all elements are sorted.
5. Display the sorted array.



```c
#include <stdio.h>
// Heapify function
void heapify(int arr[], int n, int i) {
    int largest = i;        // root
    int left = 2*i + 1;     // left child
    int right = 2*i + 2;    // right child

    if(left < n && arr[left] > arr[largest])
        largest = left;

    if(right < n && arr[right] > arr[largest])
        largest = right;

    if(largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
// Heap Sort
void heapSort(int arr[], int n) {
    // Build max heap
    for(int i = n/2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Extract elements one by one
    for(int i = n-1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}
```

```
Enter number of elements: 5
Enter elements: 6 4 8 7 2
Sorted array: 2 4 6 7 8


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

Input:

Enter no.of elements : 5

Enter sorted elements : 6 4 8 7 2

Output:

Sorted array : 2 4 6 7 8

Result: Successfully sorted the array using heap sort algorithm.