1.Given an array of strings words, return the first palindromic string in the array. If there is no such string, return an empty string "". A string is palindromic if it reads the same forward and backward. aim algorithm.

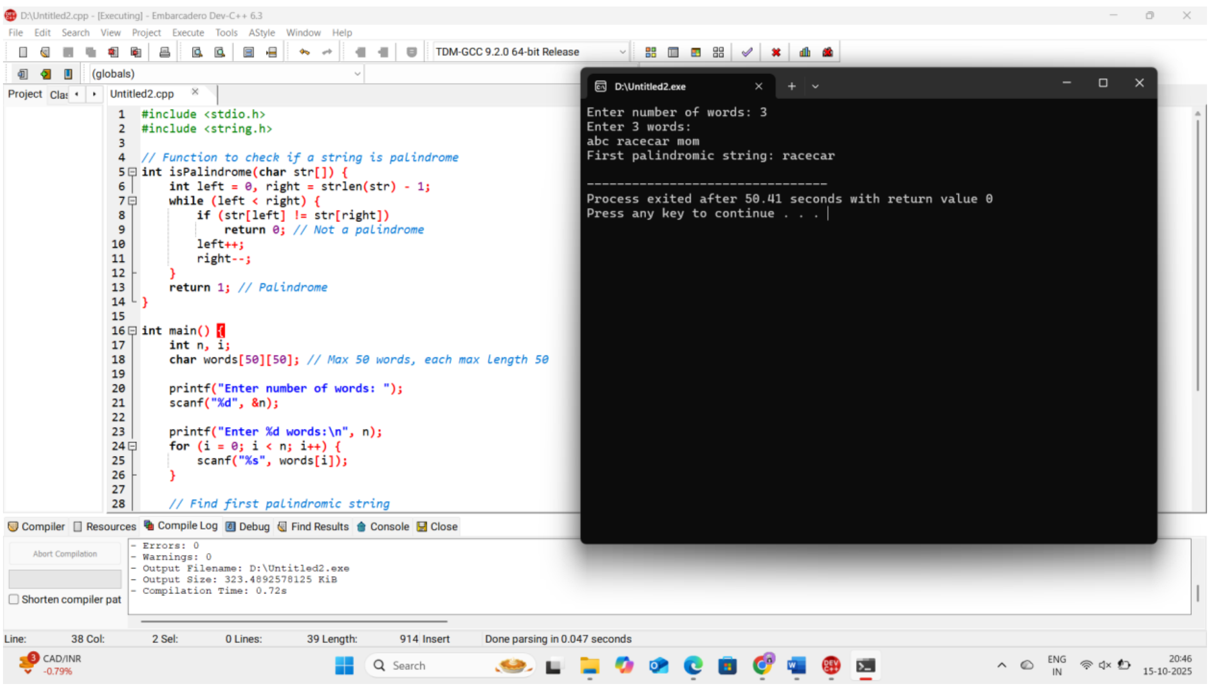
Aim

To find and return the first palindromic string from a given array of strings. If no palindrome exists, return an empty string**.**

Algorithm

1. Start
2. Input an array of strings words[].
3. For each string word in words[]:
   a. Check if word is a palindrome:

   b. If they are equal, return this string (it is the first palindrome).
4. If no palindrome is found after checking all strings, return "".
5. End



2.You are given two integer arrays nums1 and nums2 of sizes n and m, respectively. Calculate the following values: answer1 : the number of indices i such that nums1[i] exists in nums2. answer2 : the number of indices i such that nums2[i] exists in nums1 Return [answer1,answer2].

Aim

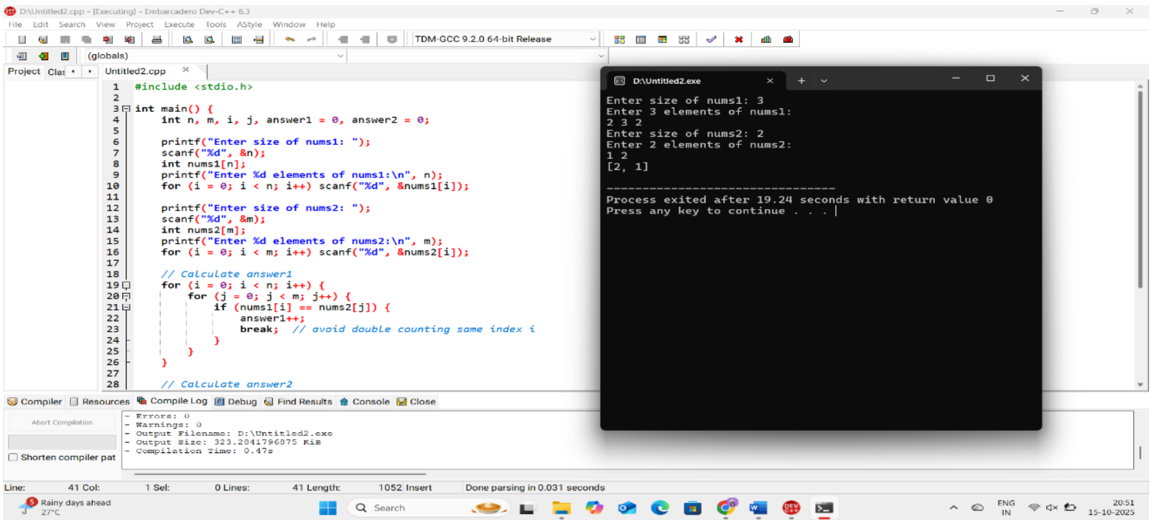
To find:

- answer1: the count of indices i where nums1[i] exists in nums2.
- answer2: the count of indices i where nums2[i] exists in nums1.

Return [answer1, answer2].

Algorithm

1. Start
2. Input two arrays nums1 (size n) and nums2 (size m).
3. Initialize answer1 = 0, answer2 = 0.
4. For each element in nums1:
    - Check if it exists in nums2.
    - If yes, increment answer1.
5. For each element in nums2:
    - Check if it exists in nums1.
    - If yes, increment answer2.
6. Print [answer1, answer2].
7. End



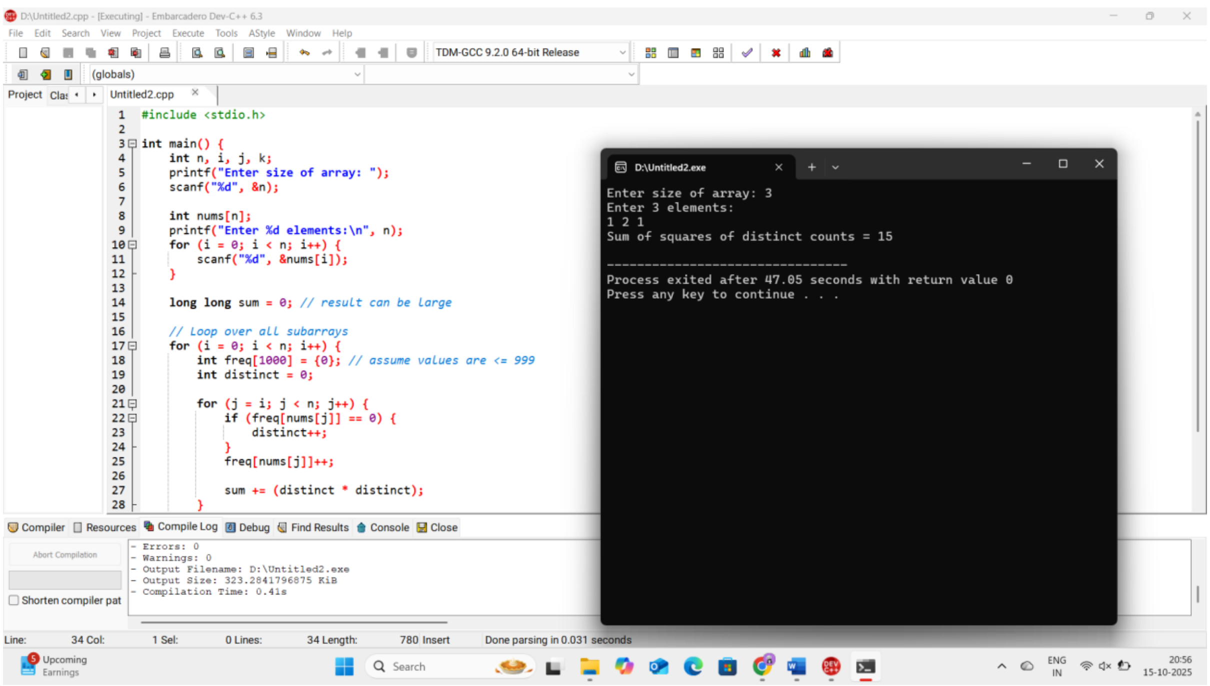
3.You are given a 0-indexed integer array nums. The distinct count of a subarray of nums is defined as: Let nums[i..j] be a subarray of nums consisting of all the indices from i to j such that 0 <= i <= j < nums.length. Then the number of distinct values in nums[i..j] is called the distinct count of nums[i..j]. Return the sum of the squares of distinct counts of all subarrays of nums. A subarray is a contiguous non-empty sequence of elements within an array.

Aim

To calculate the sum of squares of the number of distinct elements in all subarrays of a given array.

Algorithm

1. Start.
2. Input array nums of size n.
3. Initialize sum = 0.
4. For each starting index i from 0 to n-1:
5. Print sum.
6. End.



4. Given a 0-indexed integer array nums of length n and an integer k, return *the number of pairs* (i, j) *where* 0 <= i < j < n, *such that* nums[i] == nums[j] *and* (i * j) *is divisible by* k.

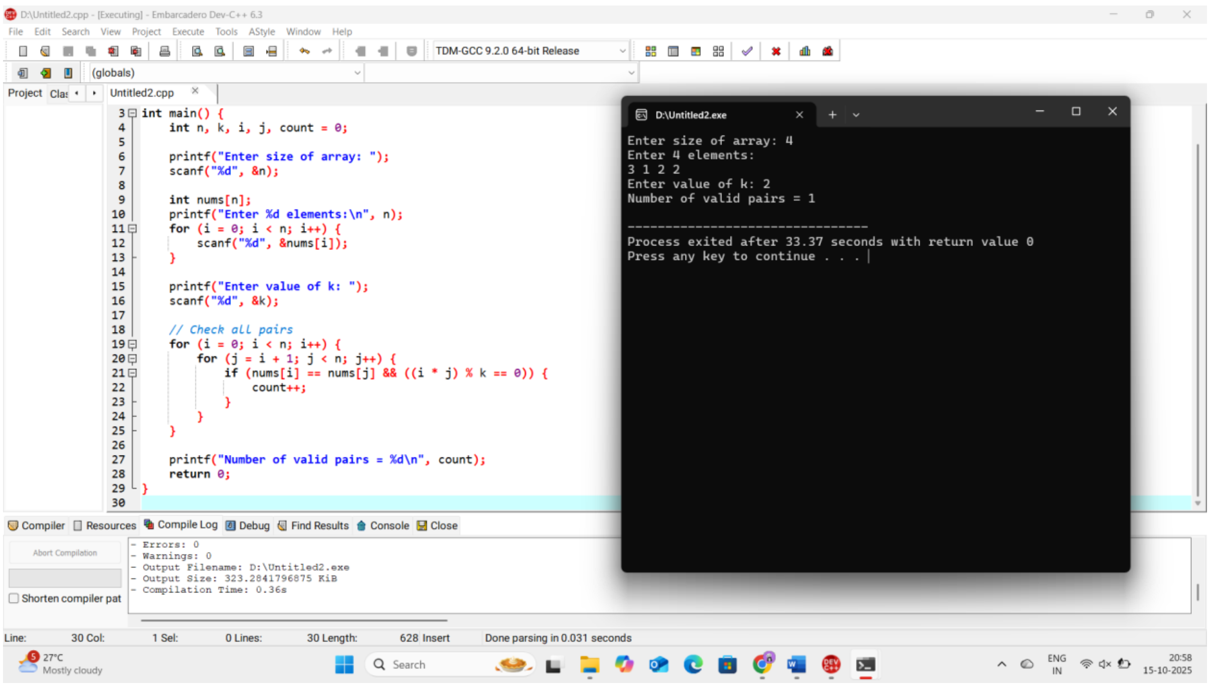
Aim

To count the number of pairs (i, j) in an array nums such that:

- 0 <= i < j < n
- nums[i] == nums[j]
- (i * j) is divisible by k.

Algorithm

1. Start.
2. Input array nums of size n and integer k.
3. Initialize count = 0.
4. For each pair (i, j) where 0 <= i < j < n:
    - If nums[i] == nums[j] and (i * j) % k == 0, then increment count.
5. Print count.
6. End.



5. Write a program FOR THE BELOW TEST CASES with least time complexity
Test Cases: -

Input: {1, 2, 3, 4, 5} Expected Output: 5

Input: {7, 7, 7, 7, 7} Expected Output: 7

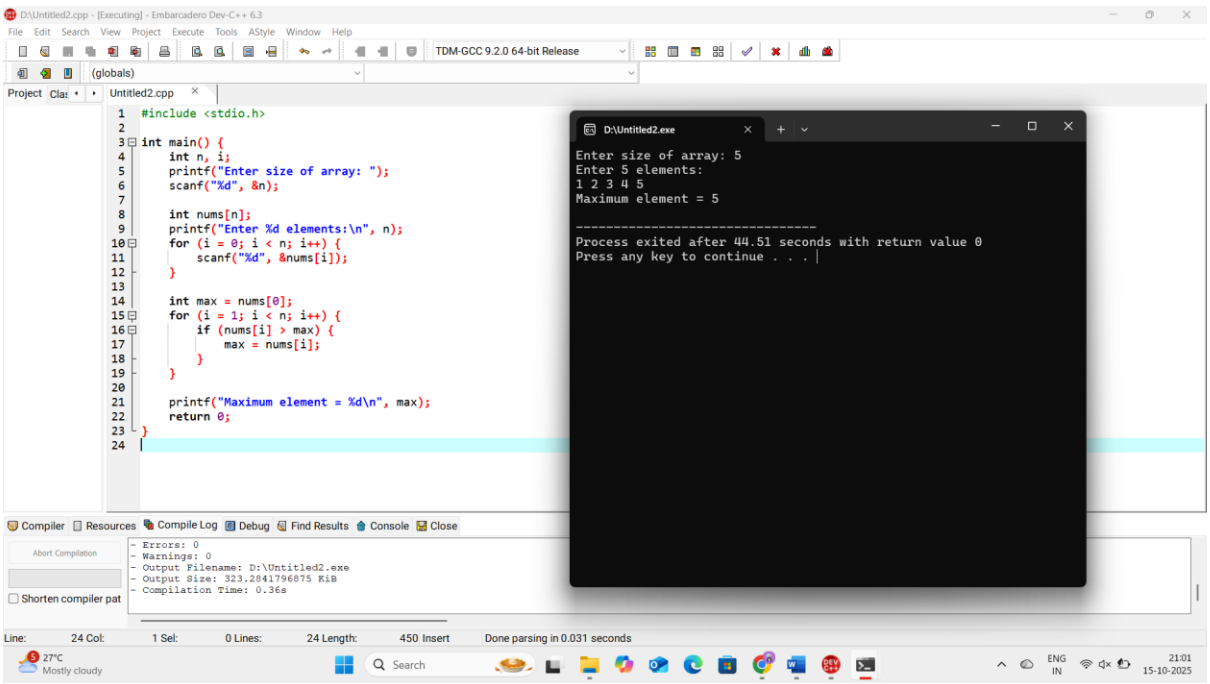
Input: {-10, 2, 3, -4, 5} Expected Output: 5

Aim

Write a program to find the maximum element of an array with least time complexity.

Algorithm

1. Start.
2. Input array of size n.
3. Initialize max = nums[0].
4. Traverse the array from index 1 to n-1:
    - If nums[i] > max, update max = nums[i].
5. Print max.
6. End.



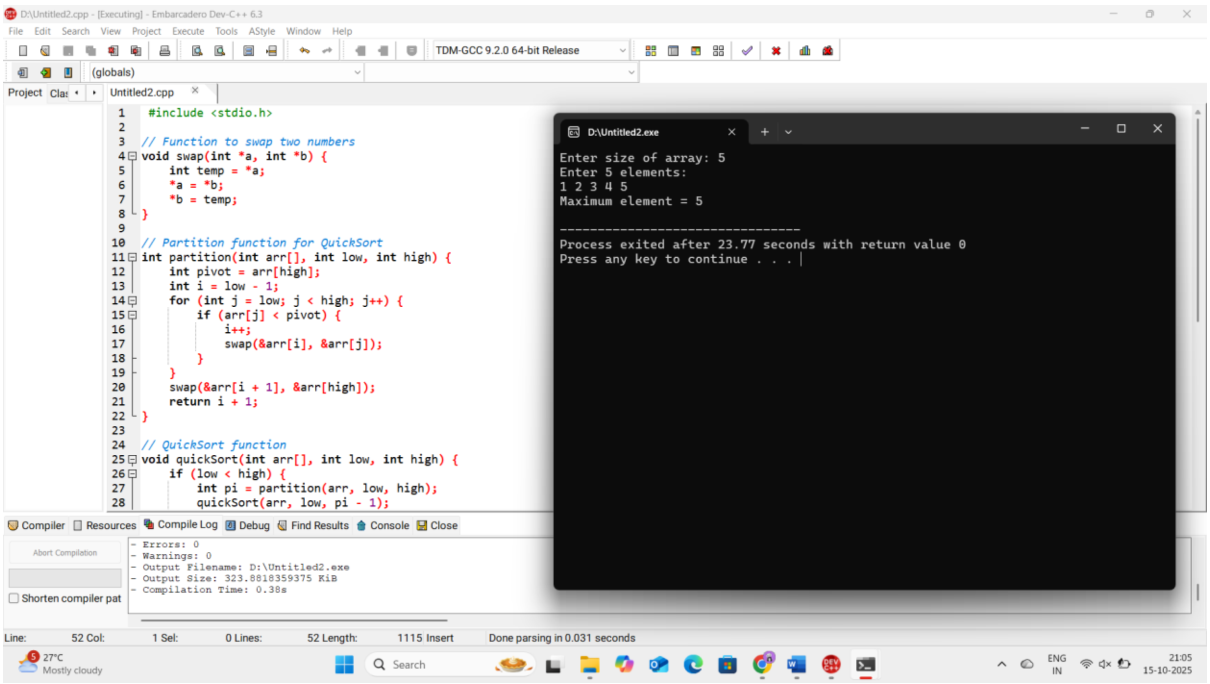
6. **You have an algorithm that process a list of numbers. It firsts sorts the list using an efficient sorting algorithm and then finds the maximum element in sorted list. Write the code for the same.**

Aim

To design a program that first sorts a list of numbers using an efficient sorting algorithm (like QuickSort or MergeSort) and then finds the maximum element in the sorted list.

Algorithm

1. Start
2. Input the size n of the array and the n elements.
3. Sort the array using an efficient sorting algorithm (QuickSort or MergeSort).
4. After sorting, the array is in ascending order.
5. The maximum element will be at the last index (arr[n-1]).
6. Print arr[n-1].
7. End



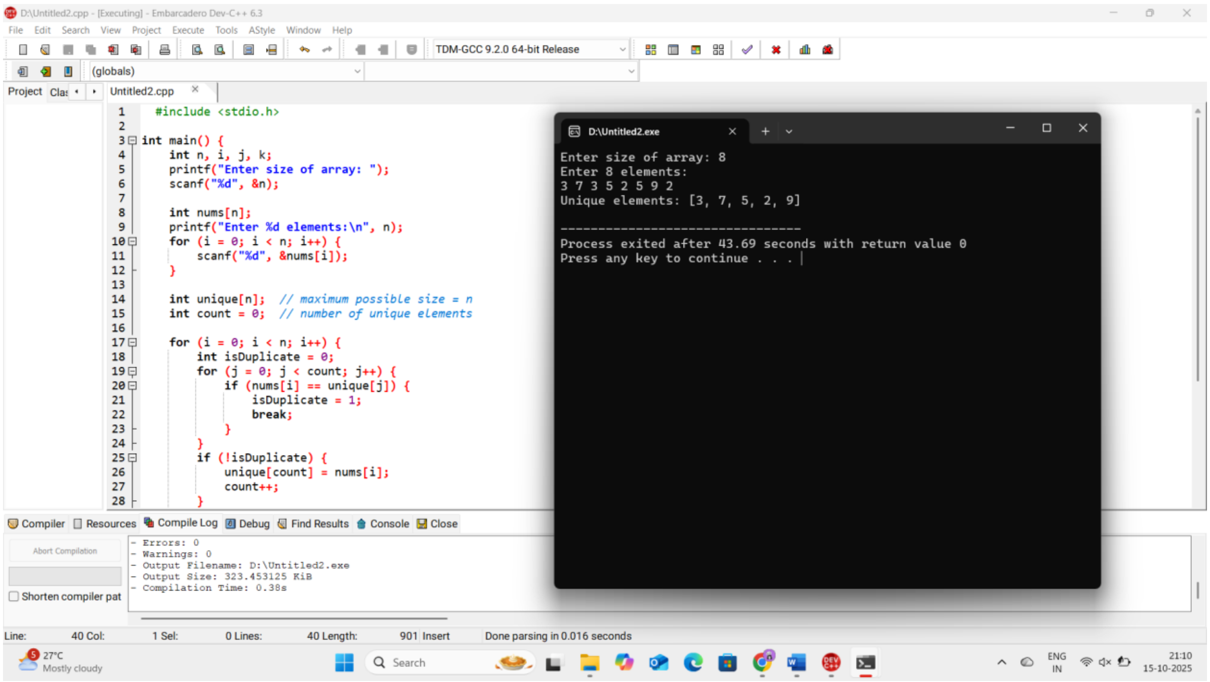
7. Write a program that takes an input list of n numbers and creates a new list containing only the unique elements from the original list. What is the space complexity of the algorithm?

Aim

To take an input list of n numbers and create a new list containing only the unique elements from the original list.

Algorithm

1. Start
2. Input the size n of the array and the array elements nums[].
3. Create an empty array unique[] to store unique elements.
4. For each element in nums[]:
    - Check if it already exists in unique[].
    - If not, add it to unique[].
5. Print the unique[] array.
6. End



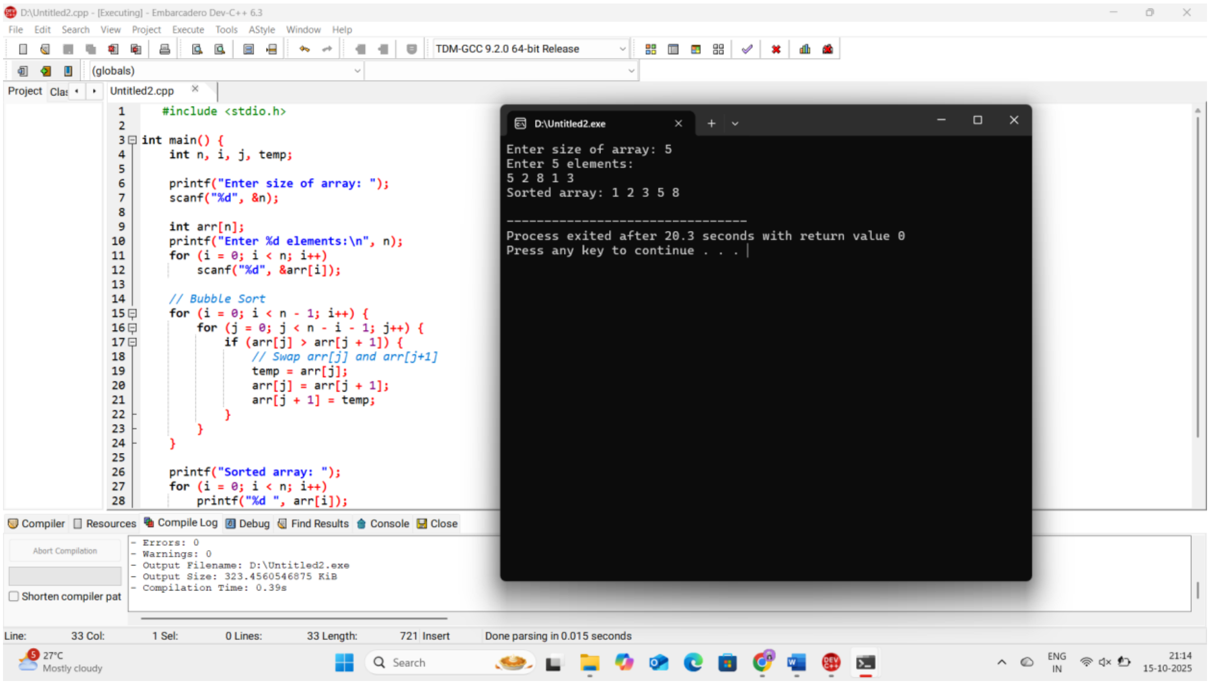
8. Sort an array of integers using the bubble sort technique. Analyze its time complexity using Big-O notation. Write the code

Aim

To sort an array of integers in ascending order using the Bubble Sort technique and analyze its time complexity**.**

Algorithm

1. Start
2. Input array arr[] of size n.
3. Repeat for i from 0 to n-2:
    - For j from 0 to n-i-2:
        - If arr[j] > arr[j+1], swap arr[j] and arr[j+1].
4. After all iterations, the array is sorted in ascending order.
5. Print the sorted array.
6. End



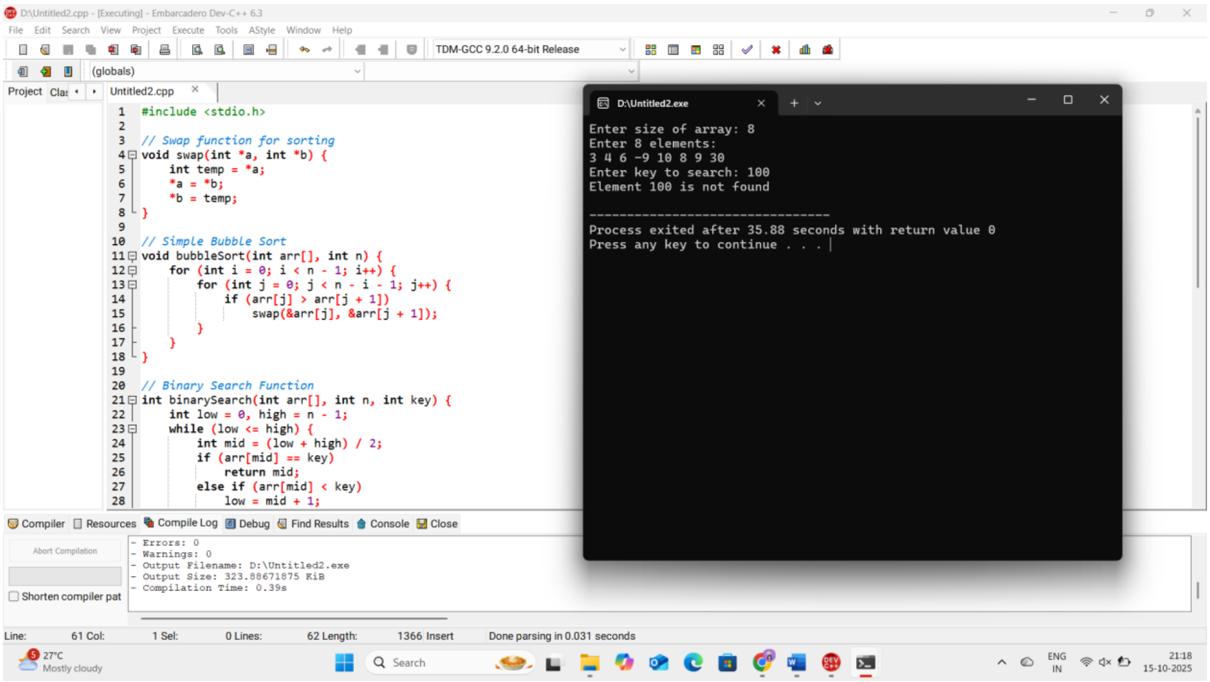
9.Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.

Aim

To check if a given number key exists in a sorted array using Binary Search and analyze its time complexity.

Algorithm

1. Start
2. Input array arr[] of size n and number key.
3. Sort the array in ascending order (use any efficient sorting like QuickSort or Bubble Sort).
4. Initialize low = 0 and high = n - 1.
5. Repeat while low <= high:
6. If loop ends without finding key, element is not found.
7. End



10.Given an array of integers nums, sort the array in ascending order and return it. You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

Aim

To sort an array of integers in ascending order using QuickSort in O(n log n) time without using built-in functions, with minimal space.

Algorithm

1. Start
2. Input array nums[] of size n.
3. Call QuickSort(nums, 0, n-1)
4. After recursion ends, array is sorted in ascending order.
5. Print the sorted array.
6. End