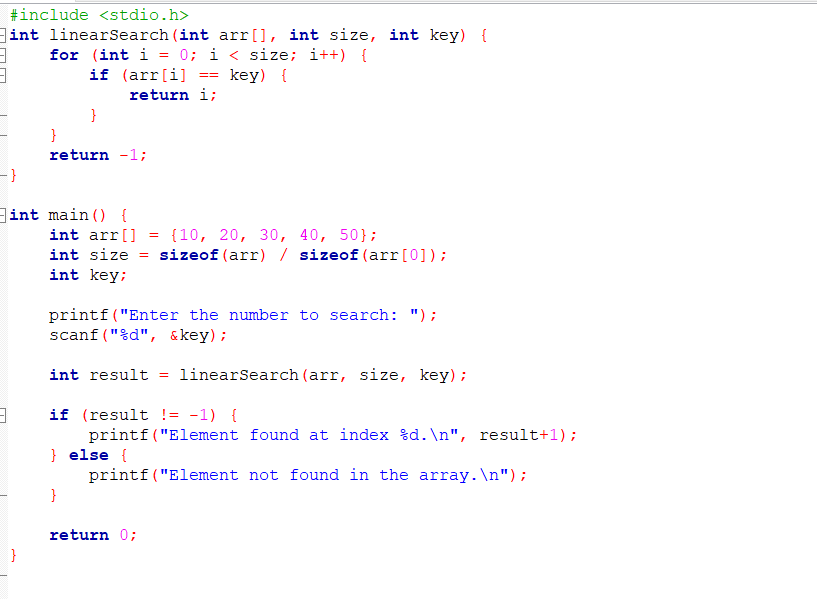
**Linear Search**

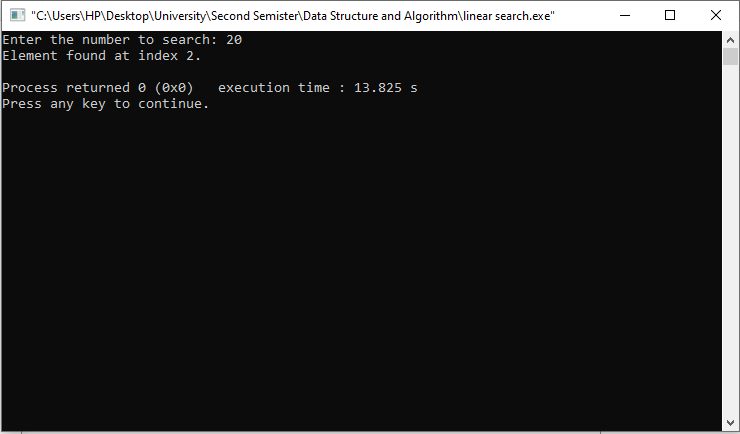
**Introduction**

**Linear Search** is a straightforward technique that examines each item in a list sequentially to locate a specific value. It operates on unsorted data, making it versatile but less optimal for large datasets due to its O(n) time complexity. Simple and intuitive, it’s best suited for smaller collections or scenarios requiring minimal setup.

Code:



Output:

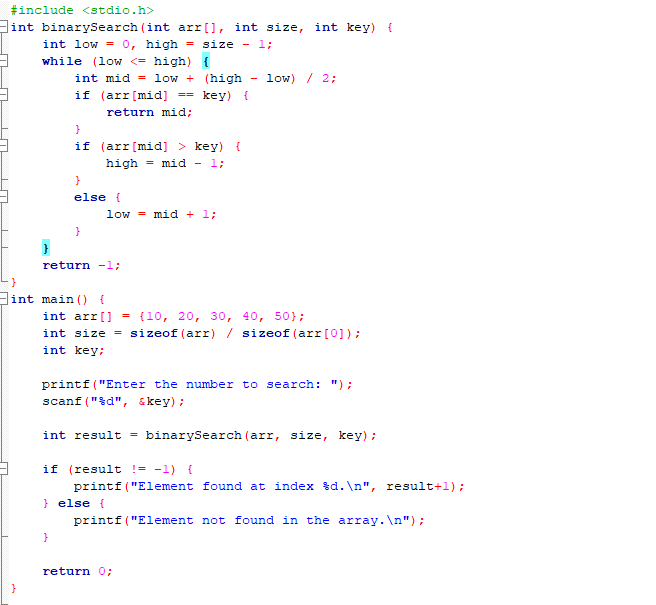


**BINATY SEARCH**

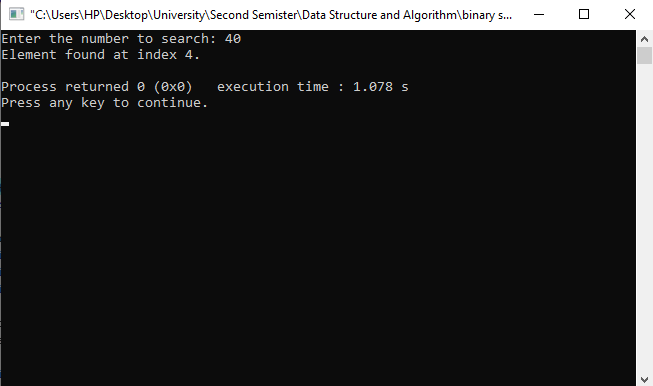
**Introduction**

**Binary Search** is an efficient algorithm to find an element in a sorted list. It works by repeatedly dividing the search interval in half, comparing the target with the middle element, and narrowing the range based on the comparison. With a time complexity of O(log n), it's faster than linear search for large, sorted datasets.

Code



Output

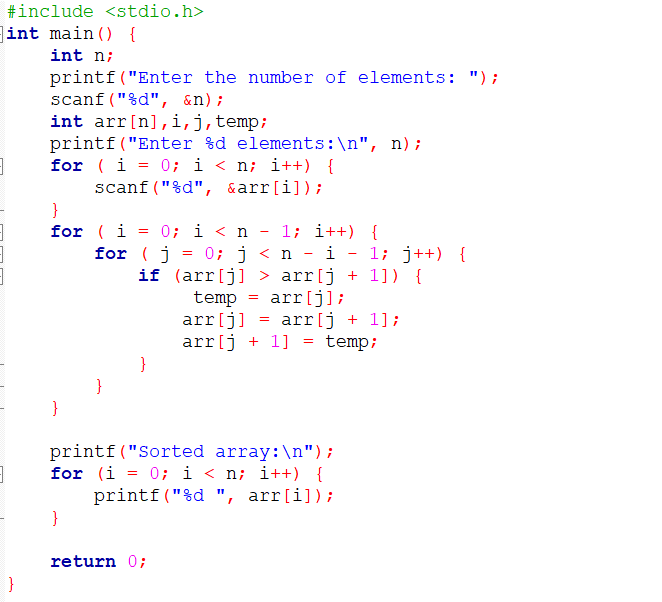
t

**Bubble Sort**

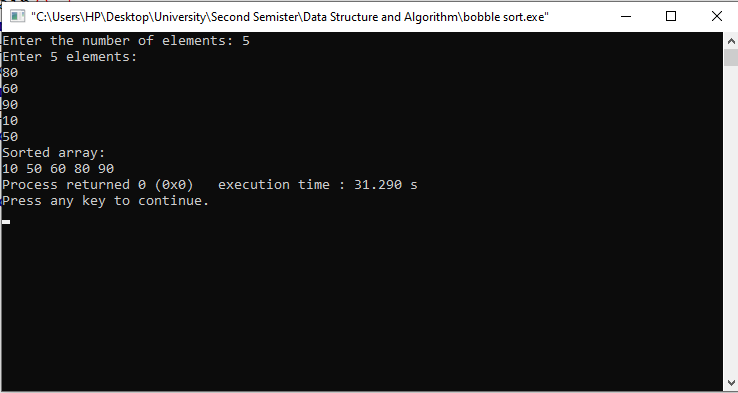
**Introduction**

**Bubble Sort** is a simple sorting algorithm that repeatedly steps through a list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until the list is fully sorted. It has a time complexity of O(n²) and is best suited for small datasets or when simplicity is more important than efficiency.

Code:



Code:

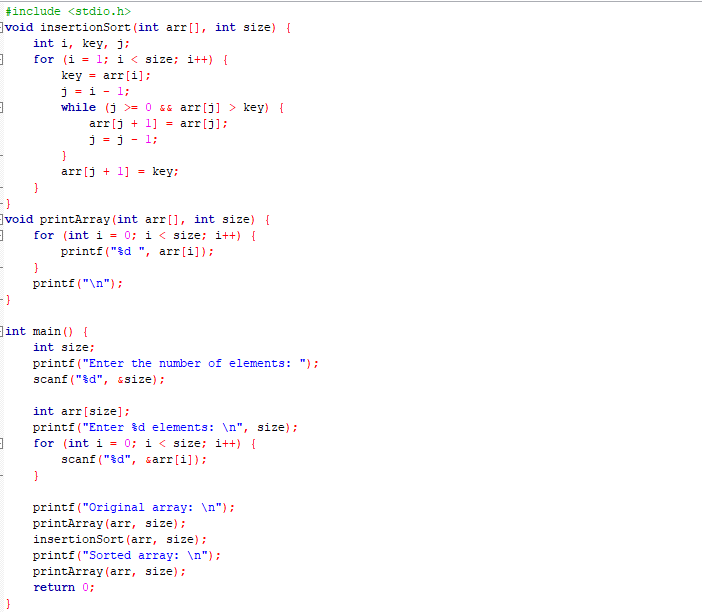


**Insertion Sort**

**Introduction**

**Insertion Sort** is a simple algorithm that sorts a list by dividing it into sorted and unsorted parts. Elements from the unsorted section are picked and inserted into the correct position in the sorted section. It's efficient for small or nearly sorted datasets, with a worst-case time complexity of O(n²).

Code:



Output:

