

CS -08

DATA STRUCTURE USING  
C LANGUAGE

By Rachel

# UNIT 2

# SORTING AND SEARCHING

# SORTING

## What is Sorting?

- Sorting in data structures in C involves arranging elements of a collection (like an array or a linked list) in a specific order, such as ascending or descending.
- Various algorithms exist to achieve this, each with its own advantages and disadvantages in terms of time and space complexity.

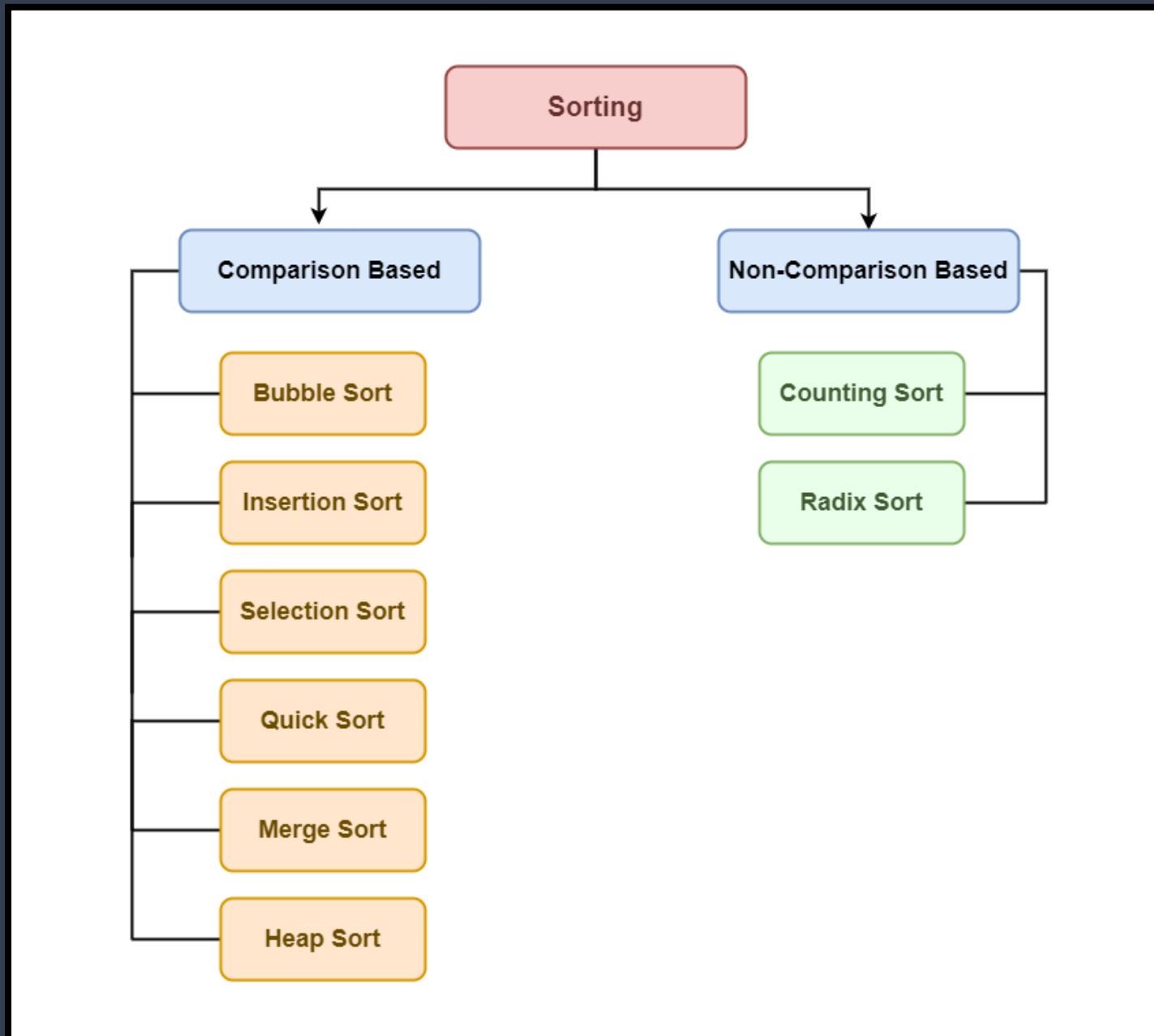
## Why Sorting Algorithms are Important?

- Sorting algorithms are essential in Computer Science as they simplify complex problems and improve efficiency. They are widely used in searching, databases, divide and conquer strategies, and data structures.

# Types of Sorting Techniques

There are various sorting algorithms are used in data structures. The following two types of sorting algorithms can be broadly classified:

- Comparison-based: We compare the elements in a comparison-based sorting algorithm
- Non-comparison-based: We do not compare the elements in a non-comparison-based sorting algorithm



# TOPIC 1

# BUBBLE SORTING

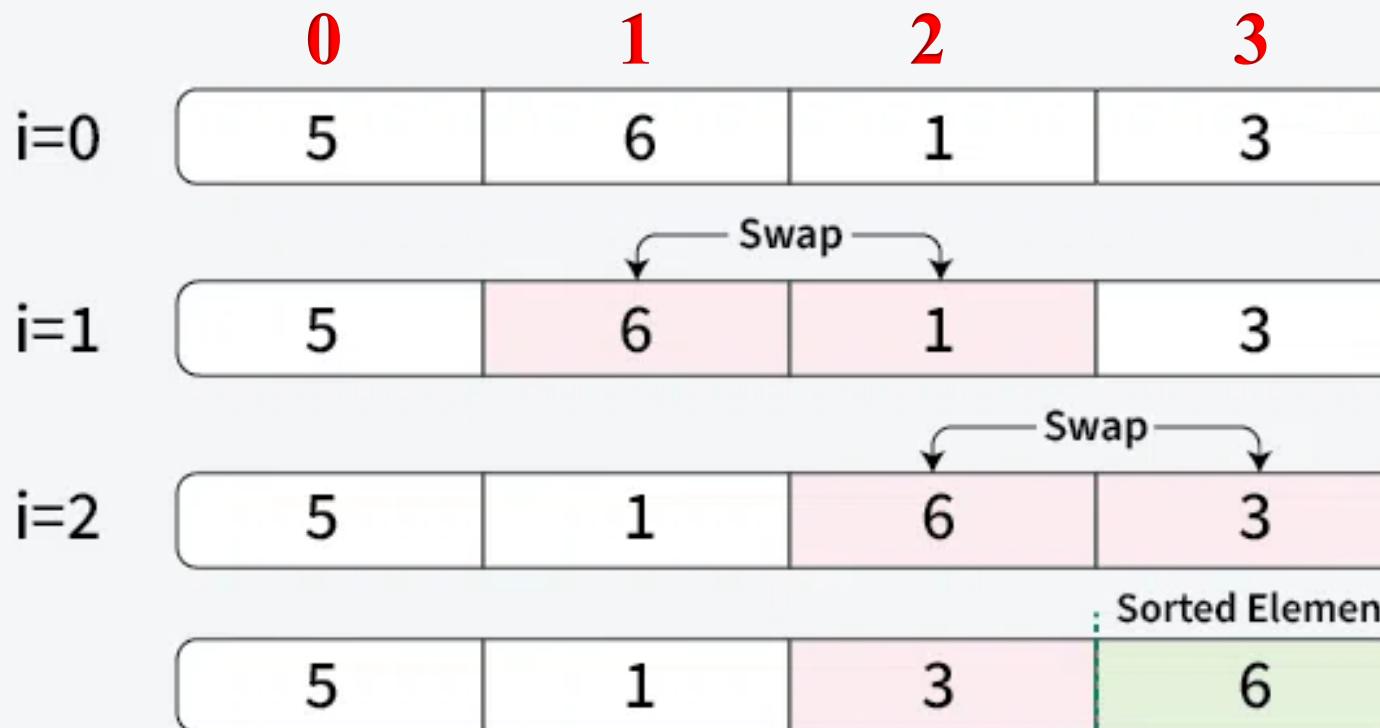
## What is Bubble Sorting?

- Bubble Sorting is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
- This algorithm is not efficient for large data sets as its average and worst-case time complexity are quite high.
- It sorts the array using multiple passes. After the first pass, the maximum goes to end (its correct position). Same way, after second pass, the second largest goes to second last position and so on.
- In every pass, process only those that have already not moved to correct position.
- It's Time Complexity is  $O(n^2)$  and Auxiliary Space is  $O(1)$

# 01

Step

Placing the 1st largest element at its correct position

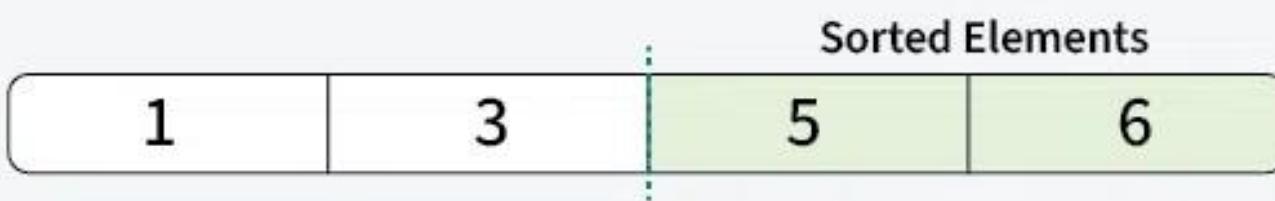
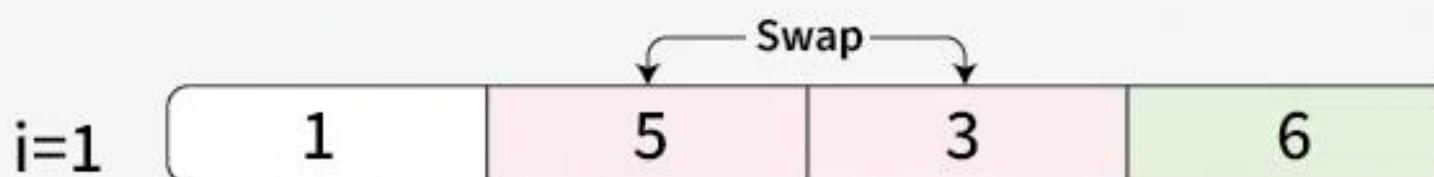
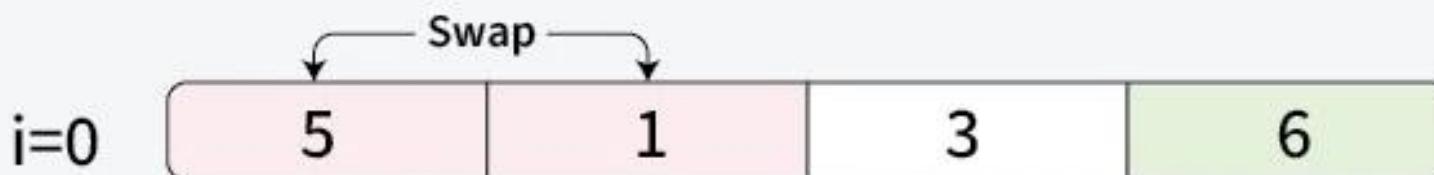


Bubble sort

## 02

Step

Placing 2nd largest element at its correct position

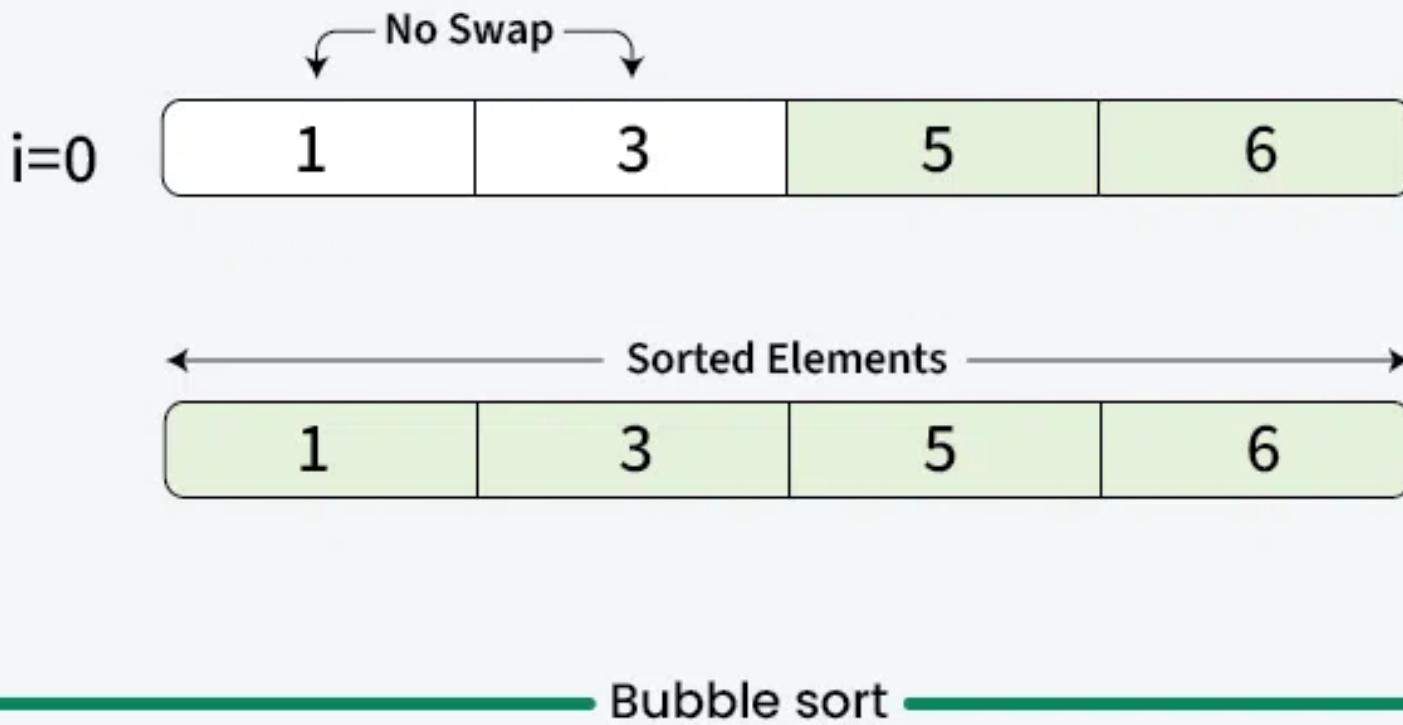


Bubble sort

## 03

Step

Placing 3rd largest element at its correct position



# Program

```
// Function to perform Bubble Sort
void bubblesort(int arr[], int n)
{
    int i, j, temp;
    // Outer Loop for passes
    for (i = 0; i < n - 1; i++)
    {
        // Inner Loop for comparisons and swaps in each pass
        for (j = 0; j < n - i - 1; j++)
        {
            // Compare adjacent elements
            if (arr[j] > arr[j + 1])
            {
                // Swap if they are in the wrong order (for ascending sort)
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

```
// Function to print an array
void printarray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array
    printf("Original array: \n");
    printarray(arr, n);
    bubblesort(arr, n); // Call the bubble sort function
    printf("Sorted array: \n");
    printarray(arr, n);
    return 0;
}
```

## Output

```
Original array:
```

```
64 34 25 12 22 11 90
```

```
Sorted array:
```

```
11 12 22 25 34 64 90
```

## Advantages of Bubble Sorting

- Bubble sort is easy to understand and implement.
- It does not require any additional memory space.
- It is a stable sorting algorithm, meaning that elements with the same key value maintain their relative order in the sorted output.

## Disadvantages of Bubble Sorting

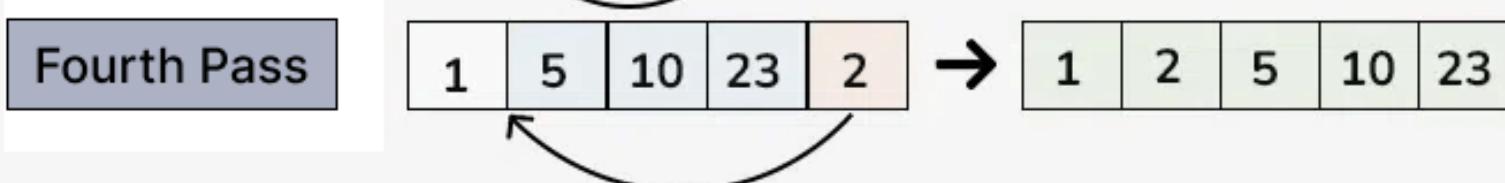
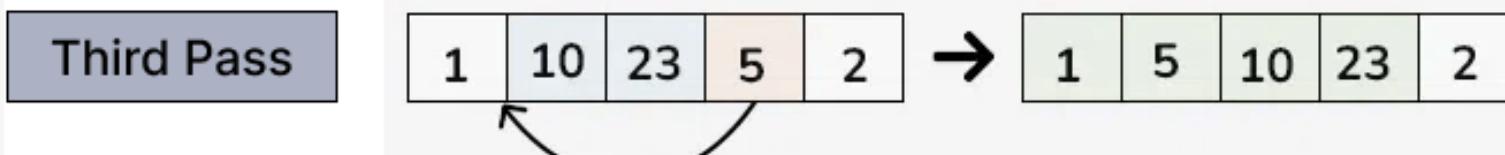
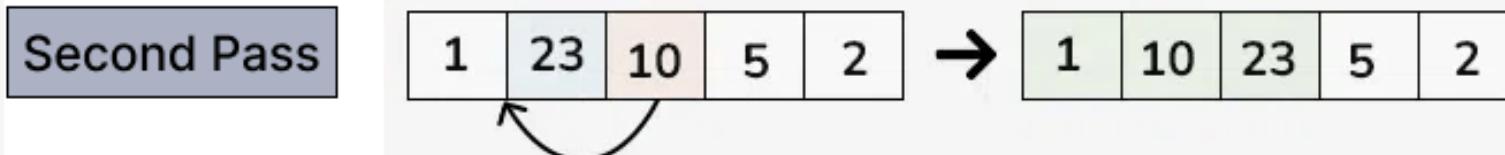
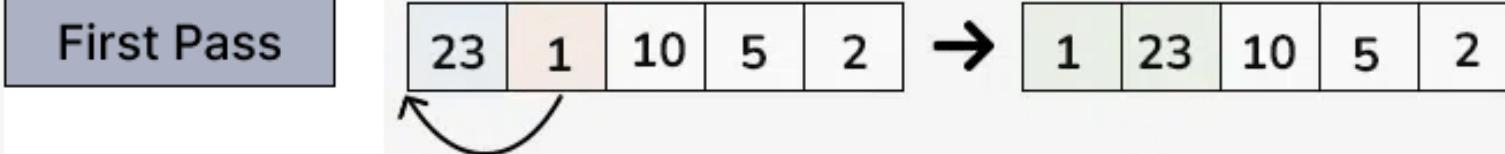
- Bubble sort has a time complexity of  $O(n^2)$  which makes it very slow for large data sets.
- Bubble sort has almost no or limited real world applications. It is mostly used in academics to teach different ways of sorting.

# TOPIC 2

# INSERTION SORTING

## What is Insertion Sorting?

- Insertion sort is one of the simple and comparison-based sorting algorithms.
- The basic idea behind the algorithm is to virtually divide the given list into two parts: a sorted part and an unsorted part, then pick an element from the unsorted part and insert it in its place in the sorted part.
- It does this till all the elements are placed in the sorted part.
- It is less efficient on large lists than more advanced algorithms such as quicksort, heap sort, or merge sort but it is simple to implement and is suitable to sort small data lists.
- It's Time Complexity is  $O(n^2)$  and Auxiliary Space is  $O(1)$ .



# Program

```
//Insertion Sorting Program in C
#include <stdio.h>

void insertionsort(int arr[], int n)
{
    int i, key, j;
    // Loop from the second element (index 1) to the end of the array
    for (i = 1; i < n; i++)
    {
        key = arr[i]; // Store the current element to be inserted
        j = i - 1;    // Start comparing with the element just before it
        while (j>=0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key; // Insert the key at the correct position
    }
}
```

```
void printarray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: \n");
    printarray(arr, n);

    insertionsort(arr, n);

    printf("Sorted array in ascending order: \n");
    printarray(arr, n);

    return 0;
}
```

## Output

```
Original array:
```

```
12 11 13 5 6
```

```
Sorted array in ascending order:
```

```
5 6 11 12 13
```

## Advantages of Insertion Sorting

- Simple and easy to implement.
- Stable sorting algorithm.
- Efficient for small lists and nearly sorted lists.
- Space-efficient as it is an in-place algorithm.
- Adaptive. the number of inversions is directly proportional to number of swaps. For example, no swapping happens for a sorted array and it takes  $O(n)$  time only.

## Disadvantages of Insertion Sorting

- Inefficient for large lists.
- Not as efficient as other sorting algorithms (e.g., merge sort, quick sort) for most cases.