Unit II: Analysis of Sorting and Searching Algorithms

1. Introduction to Sorting Algorithms

What is Sorting?

Sorting is the process of arranging data elements in a specific order. The most common orders are:

- **Ascending order** (smallest to largest)
- **Descending order** (largest to smallest)

Sorting is a fundamental operation in computer science because it improves the efficiency of searching, simplifies data analysis, and makes information easier to understand and manage.

Why Do We Need Sorting?

- To make **searching faster** (binary search requires sorted data)
- To **organize data** for better presentation
- To help in **efficient algorithm design**
- Used in databases, operating systems, and real-world applications like ranking systems

---

2. Classification of Sorting Algorithms

Sorting algorithms can be broadly classified as:

- **Elementary (Simple) Sorting Techniques**
- Advanced Sorting Techniques

In this unit, we begin with **Elementary Sorting Techniques**, which are easy to understand and suitable for small datasets.

---

3. Elementary Sorting Techniques

Definition

Elementary sorting techniques are basic sorting algorithms that use simple logic and comparison-based methods to arrange elements.

Characteristics

- Easy to understand and implement
- Use simple loops and comparisons
- Not efficient for large datasets

Examples

- Selection Sort
- Bubble Sort
- Insertion Sort

---

4. Selection Sort

4.1 Concept of Selection Sort

Selection Sort works by **repeatedly selecting the smallest (or largest) element from the unsorted part of the array and placing it at the correct position**.

At every step:

- The array is divided into two parts:
  - Sorted part (left side)
  - Unsorted part (right side)
- The smallest element from the unsorted part is selected and swapped with the first element of the unsorted part

---

4.2 Step-by-Step Explanation

Let us consider an array:

A = [64, 25, 12, 22, 11]

Pass 1:

- Minimum element in the array = 11
- Swap 11 with 64
- Array becomes: `[11, 25, 12, 22, 64]`

Pass 2:

- Minimum element in the unsorted part = 12
- Swap 12 with 25
- Array becomes: `[11, 12, 25, 22, 64]`

Pass 3:

- Minimum element = 22
- Swap 22 with 25
- Array becomes: `[11, 12, 22, 25, 64]`

Pass 4:

- Minimum element = 25
- No swap needed

Now the array is sorted.

4.4 Pseudocode

```
for i = 0 to n-1
   min = i
   for j = i+1 to n-1
      if A[j] < A[min]
         min = j
   swap A[i] and A[min]
```

4.5 Flow of Selection Sort

- Start
- Select the minimum element from the unsorted array
- Swap it with the first unsorted position
- Move the boundary of the sorted array by one
- Repeat until the entire array is sorted

4.6 Time Complexity Analysis

Case	Time Complexity
Best Case	(O(n^2))
Average Case	(O(n^2))
Worst Case	(O(n^2))

- Selection sort always performs the same number of comparisons, regardless of input order

---

4.7 Space Complexity

- Space Complexity: **O(1)**
- Reason: Sorting is done in-place and no extra memory is required

---

4.8 Stability of Selection Sort

- **Selection Sort is NOT a stable sorting algorithm**
- Equal elements may change their relative order after sorting

---

Advantages of Selection Sort

- Very easy to understand
- Simple to write and use
- Uses less memory
- Swaps elements only a few times

Disadvantages of Selection Sort

- Very slow for large data
- Takes more time (O(n²))
- Not good for big or real-time programs

4.11 Applications of Selection Sort

- Suitable for small lists
- Educational purposes
- Used when swap operations are expensive

---

5. Summary of Selection Sort

- Selection Sort repeatedly selects the minimum element
- Divides array into sorted and unsorted parts
- In-place and comparison-based algorithm
- Time Complexity: (O(n^2))
- Space Complexity: (O(1))
- Not stable

Note for Students:
Selection Sort is important for understanding the basic idea of sorting and comparison-based algorithms. Although not used in practice for large data, it forms the foundation for learning more efficient sorting techniques.