Lecture One: Sorting and Searching

What is Sorting?

Sorting is the process of arranging a group of items into a specific order based on a certain criterion.
For example, arranging a list of people in **ascending order according to age**.

Example:

Unsorted:

Ages: **32, 18, 25, 40**

Sorted (ascending):

18, 25, 32, 40

How Sorting Algorithms Are Classified

Sorting algorithms can be classified based on several important factors:

1. Stability

A sorting algorithm is **stable** if it keeps the original order of equal elements.

Example:
If two students have the same grade, a stable sort keeps their original order.

2. Time

Refers to how long the algorithm takes to complete the sorting process.

3. Space (Memory Usage)

Refers to the amount of memory the algorithm needs while sorting.

- **In-place sorting:** uses very little extra memory.
- **Out-of-place sorting:** requires additional memory.

4. Adaptability

Measures how well the algorithm performs when the data is already partially sorted.

Some algorithms become faster if the list is nearly sorted.

5. Computational Complexity (Big O Notation)

Sorting algorithms are analyzed using **Big O notation**.

This theory measures:

- Number of comparisons
- Number of swaps
- Number of operations

These are studied in three cases:

1. **Worst Case:**
   The slowest possible scenario.
2. **Average Case:**
   Typical performance.
3. **Best Case:**
   Fastest possible scenario.

All of these are measured in terms of the size of the list, denoted by **n**.

Complexity Examples

For most serial (normal) sorting algorithms:

- Good performance:
  O(n log n)
  (e.g., Quick Sort, Merge Sort)

- Bad performance:
  $O(n^2)$
  (e.g., Bubble Sort)
- Ideal performance:
  O(n)
  (when the list is already sorted in some algorithms)

For parallel sorting algorithms:

- Performance can reach **$O(\log^2 n)$** in some models.

There are many types of sorting algorithms, including:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- Merge Sort

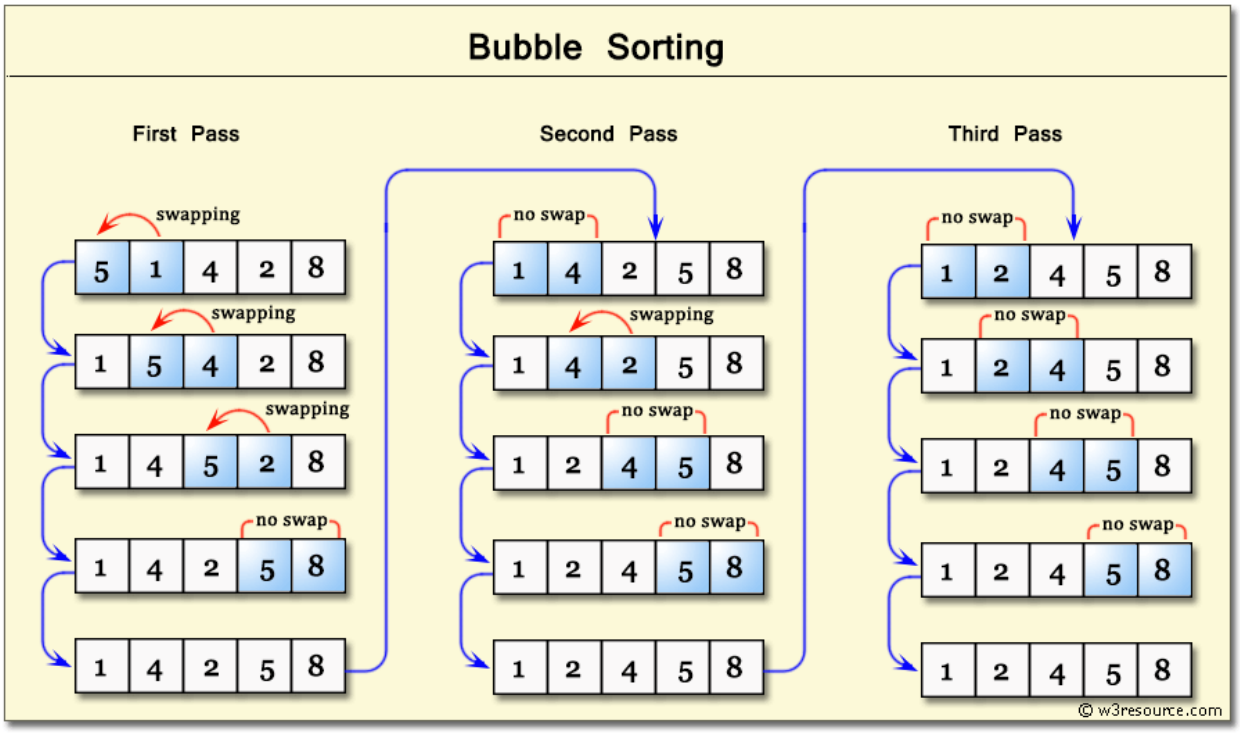
Bubble Sort

• The bubble sort improves performance by making more than one exchange during each pass.

• By making multiple exchanges, more elements move toward their correct positions using the same number of comparisons as selection sort.

• The key idea of bubble sort is to make **pairwise comparisons** and swap the elements if they are in the wrong order.

Lecture One: Sorting and Searching



In bubble sort, each element is compared with its neighbor.

If the left element is larger, the two elements are swapped.

This process continues across the list.

Quick Sort

• Quick sort is a fast and efficient sorting algorithm based on the **divide and conquer** technique.

• It works by selecting a special element called the **pivot**.

• The array is then divided into two parts:

- Elements **smaller than the pivot**.
- Elements **greater than the pivot**.
  • The same process is repeated on each part until the whole array becomes sorted.

Lecture One: Sorting and Searching