Lecture Two
SORTING

- **Sorting** refers to the process of organizing a set of items in a specific order. It can be applied to both numeric and alphabetic data, arranging them in either ascending or descending order. There are various sorting algorithms available, and the selection of an appropriate one depends on factors such as the number of items in relation to available memory, the level of order in the data, the range of keys, and the relative costs of comparing keys versus moving items.

- **The Sorting Problem**
  - **Input:** A sequence of n numbers
  - **Output:** A permutation (reordering) $xl, x2, .., xn$ of the input sequence.
- **Complexity of Sorting**

  - The complexity function of sorting is a notation of different data comparison done while sorting the data items of an array.
  - The sorting complexity is represented using Big O notation.

- It is defined as O (f (n)), where f (n) is a function of no. of data item sorted in an array.
- The complexity enables you to compare the memory space occupied and run time required for each sorting technique.

1. Bubble Sort

The idea of bubble sort is to repeatedly move the largest element to the highest index position of the array. Bubble sort focuses on successive adjacent pairs of elements in the array, compares them, and either swaps them or not. In either case, after such a step, the larger of the two elements will be in the higher index position. The focus then moves to the next higher position, and the process is repeated. When the focus reaches the end of the array, the largest element will have ``bubbled'' from whatever its original position to the highest index position in the array.

```
Initial array: [64, 34, 25, 12, 22, 11, 90]
Array indices: 0   1   2   3   4   5   6
Values:       64  34  25  12  22  11  90

Pass 1 (i=0): j compares pairs (0-1), (1-2), (2-3), (3-4), (4-5), (5-6)
Pass 2 (i=1): j compares pairs (0-1), (1-2), (2-3), (3-4), (4-5)
Pass 3 (i=2): j compares pairs (0-1), (1-2), (2-3), (3-4)
Pass 4 (i=3): j compares pairs (0-1), (1-2), (2-3)
Pass 5 (i=4): j compares pairs (0-1), (1-2)
Pass 6 (i=5): j compares pairs (0-1)
```

--- Pass 1 (i = 0) ---

j = 0: compare arr[0]=64 and arr[1]=34 → swap → [34, 64, 25, 12, 22, 11, 90]

j = 1: compare arr[1]=64 and arr[2]=25 → swap → [34, 25, 64, 12, 22, 11, 90]

j = 2: compare arr[2]=64 and arr[3]=12 → swap → [34, 25, 12, 64, 22, 11, 90]

j = 3: compare arr[3]=64 and arr[4]=22 → swap → [34, 25, 12, 22, 64, 11, 90]

j = 4: compare arr[4]=64 and arr[5]=11 → swap → [34, 25, 12, 22, 11, 64, 90]

j = 5: compare arr[5]=64 and arr[6]=90 → no swap → [34, 25, 12, 22, 11, 64, 90]

--- Pass 2 (i = 1) ---

j = 0: compare arr[0]=34 and arr[1]=25 → swap → [25, 34, 12, 22, 11, 64, 90]

j = 1: compare arr[1]=34 and arr[2]=12 → swap → [25, 12, 34, 22, 11, 64, 90]

j = 2: compare arr[2]=34 and arr[3]=22 → swap → [25, 12, 22, 34, 11, 64, 90]

j = 3: compare arr[3]=34 and arr[4]=11 → swap → [25, 12, 22, 11, 34, 64, 90]

j = 4: compare arr[4]=34 and arr[5]=64 → no swap → [25, 12, 22, 11, 34, 64, 90]

--- Pass 3 (i = 2) ---

j = 0: compare arr[0]=25 and arr[1]=12 → swap → [12, 25, 22, 11, 34, 64, 90]

j = 1: compare arr[1]=25 and arr[2]=22 → swap → [12, 22, 25, 11, 34, 64, 90]

j = 2: compare arr[2]=25 and arr[3]=11 → swap → [12, 22, 11, 25, 34, 64, 90]

j = 3: compare arr[3]=25 and arr[4]=34 → no swap → [12, 22, 11, 25, 34, 64, 90]

--- Pass 4 (i = 3) ---

j = 0: compare arr[0]=12 and arr[1]=22 → no swap → [12, 22, 11, 25, 34, 64, 90]

j = 1: compare arr[1]=22 and arr[2]=11 → swap → [12, 11, 22, 25, 34, 64, 90]

j = 2: compare arr[2]=22 and arr[3]=25 → no swap → [12, 11, 22, 25, 34, 64, 90]

--- Pass 5 (i = 4) ---

j = 0: compare arr[0]=12 and arr[1]=11 → swap → [11, 12, 22, 25, 34, 64, 90]

j = 1: compare arr[1]=12 and arr[2]=22 → no swap → [11, 12, 22, 25, 34, 64, 90]

--- Pass 6 (i = 5) ---

j = 0: compare arr[0]=11 and arr[1]=12 → no swap → [11, 12, 22, 25, 34, 64, 90]

Final sorted array: [11, 12, 22, 25, 34, 64, 90]

➢ Advantages of Bubble Sort:

- 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 Sort:

- Bubble sort has a time complexity of O(n2) 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.