Tutorial 3

Ans1 ⇒ pseudofunction for linear search ⇒

```
⇒ int linearS ( int * arr[], int n, int key){
        for (int i=0; i<n; i++) {
            if (arr[i] == key)
                return i;
        }
        return -1;
    }
```

Ans2 ⇒ Pseudo code of Insertion Sort ⇒

```
⇒ void insertion( int arr[], int n){
        for (int i=1; i<n; i++){
            int key = arr[i];
            int j = i-1;
            while ( j>=0 && arr[j] > key){
                arr[j+1] = arr[j];
                j--;
            }
            arr[j+1] = key;
        }
}
```

* Insertion sort is called online sorting as, as soon as an element comes in an array ⇒ it is automatically inserted at its correct position.

* Other algos. discussed in lectures are not online.

Ans3 ⇒ Avg. case complexities of Sorting Algos ⇒

* Bubble = $O(n^2)$
* Insertion ⇒ $O(n^2)$
* Selection ⇒ $O(n^2)$
* Merge ⇒ $O(n \log n)$
* Quick ⇒ $O(n \log n)$
* Heap ⇒ $O(n \log n)$

Ans4 ⇒

	Stable	Inplace
Bubble ⇒	✓	✓
Selection ⇒	✗	✓
Insertion ⇒	✓	✓
Merge ⇒	✓	✗
Quick ⇒	✗	✗
Heap ⇒	✗	✓

Ans5 ⇒ pseudocode for Binary Search ⇒

```
⇒ int start = 0
  int end = size - 1
  while (start <= end) {
        int mid = start + (end - start) / 2;
        if (key == arr[mid])
              return mid;
        else if (key < arr[mid])
              end = mid - 1;
        else
              start = mid + 1;
  }
  return -1;
```

⊙ Time ⇒ O(logn)

⊙ space ⇒ O(1)

⊙ Time (of linear search) ⇒ O(n)

Ans6 ⇒ Recurrence relation of Binary Search ⇒ $T(n) = T(n/2) + 1$

⊙ space ⇒ O(1)

Ans8 ⇒ Quick Sort is best sorting algo. in practical uses as it follows the locality of reference and also best case time comp. is O(nlogn).

Ans9 ⇒ Number of inversions ⇒ It tells us how far is array from being sorted.

↳ if $a[i] > a[j]$ and $i < j$.

⇒ 7 21 31 8 10 1 20 6 4 5

↳ no. of inversions ⇒ 4 + 7 + 7 + 4 + 4 + 3 + 2

⇒ 31

Ans10 ⇒ Quick Sort will give ⇒

* Best case complexity ⇒ when array is totally unsorted
* worst comp. ⇒ when arr. is sorted or reverse sorted.

Ans11 ⇒ Recurrence relations of ⇒

⇒ merge Sort

Best ⇒ $2T(n/2) + \theta(n)$

Worst ⇒ $2T(n/2) + \theta(n)$

⇒ quickSort

$T(n) = T(k) + T(n-k-1) + \theta(n)$

$T(n) = T(n-1) + \theta(n)$

* **Similarity** ⇒ Both are types of Divide and Conquer Algorithms.

* **Differences** ⇒ Worst case complexity of Merge sort is $O(n \log n)$ whereas of Quick sort is $O(n^2)$.

Ans13 ⇒ optimised Bubble Sort ⇒

```
⇒ for (int i=0; i<n; i++){
        swap = false;
        for (j=0; j< n-i-1; j++){
            if (arr[j] < arr[j+1]){
                swap (arr[j], arr[j+1]);
                swapped = true;
            }
        }
    }
```

Ans14 ⇒ In such case, Merge sort would be efficient as it is an **External sorting** algo ⇒ data is divided into chunks and then sorted using Merge Sort.

⇒ Sorted data is dumped into files.

⊙ **Internal Sorting** ⇒ It is type of sort in which whole sorting takes place in main memory of computer.