Tutorial 3

Name → Tanvi Nautiyal
Section → G
Roll No. → 47

Q.1. Write linear search Pseudo code..

```
for (i=0 to n)
 { if (arr[i] == value)
      // Element found
 }
```

Q.2. Write Pseudo code for insertion sort. Why is it called online sort? Name other sorting algorithms.

Iterative :

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

Recursive :

```
void InsertionSort (int arr[], int n)
  { if (n<=1)
      return;
    InsertionSort (arr, n-1);
    int last = arr[n-1];
```

```
    int j = n-2;
    while (j>=0 && arr[j]>last)
    { arr[j+1] = arr[j];
      j--;
    }
    arr[j+1] = last;
}
```

- Insertion sort is called 'online sort' because it does not need to know anything about what values it will sort and information is requested while algorithm is running.

Other sorting algorithms:
- Bubble Sort
- Quick Sort
- Merge Sort
- Selection Sort
- Heap Sort

Q.3. Complexities:

Sorting Algorith	Best	Worst	Average
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q.4. Divide all sorting algorithms into inplace / stable / online sorting.

Inplace Sorting	Stable Sorting	Online Sorting
Bubble Sort	Bubble Sort	Insertion Sort
Selection Sort	Insertion Sort	
Insertion Sort	Merge Sort	
Quick Sort	Count Sort	
Heap Sort		

Q.5 Write pseudo code of binary search and time complexity of linear and binary search.

Iterative:

```
int B_Search (arr, l, r, key)
{
  while (l <= r)
  {
    m = ((l+r)/2);
    if (arr[m] == key)
      return m;
    else if (key < arr[m])
        r = m-1;
    else
        l = m+1;
  }
  return -1;
}
```

Recursive:

```
int B-search (arr, int l, r, key)
{ while (l <= r) {
    m = ((l+r)/2);
```

```
if (key == arr[m])
    return B_search(arr, l, mid-1, key);
    else
      return B-search(arr, mid+1, r, key);
    }
    return -1;
}
```

⇒ Time Complexity :

Linear search → O(n)

Binary search → O(log n)

Q.6. Write reaurrance relation for Binary recurrsion search.

$T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/4) + 1$ — (2)

$T(n/4) = T(n/8) + 1$ — (3)

$T(n) = T(n/2) + 1$

$= T(n/4) + 1 + 1$

$= T(n/8) + 1 + 1 + 1$

$\vdots$

$= T(n/2^k) + (k \text{ Times})$

Let $g^k = n$

$k = \log n$

$T(n) = T(m/n) + \log n$

$T(n) = T(1) + \log n$

$T(n) = O(\log n)$ Ans.

Q.7. Find two indexes such that A[i] + A[j] = K in minimum time complexity.

→
```
for (i=0; i<n; i++)
{
    for (j=0; j<n; j++)
    {
        if (a[i] + a[j] == K)
            printf("%d %d", i, j);
    }
}
```

Q.8. Which sorting is best for practical uses? Explain.

→ Quick sort is fastest general-purpose sort. In most practical situations quicksort is the method of choice or stability is important and space is available, mergesort might be best.

Q.9. What do you mean by inversions in an array? Count the no. of inversions in Array A[] = {7, 21, 31, 8, 10, 1, 20, 0, 4, 5} using merge sort.

A Pair (A[i], A[j]) is said to be inversion if
- A[i] > A[j]
- i < j
- Total no. of inversions in given array are 31 using merge sort.

Q.10. In which case Quick sort will give best or worst case time complexity.

→ Worst Case $O(n^2)$ → The worst case occurs when the pivot element is an extreme (smallest / largest) element. This happens when input array is sorted or reverse sorted and either first or last element is selected as pivot.

Best case $O(n \log n)$ → The best case occurs when we will select pivot element as a mean element.

Q.11. Write Recurrance Relation of Merge / Quick sort in best & Worst case. State the differences.

Merge Sort:

Best case → $T(n) = 2T(n/2) + O(n)$

Worst case → $T(n) = 2T(n/2) + O(n)$ → $\{O(n \log n)\}$

Quick Sort:

Best case → $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$

Worst case → $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

- In quick sort, array of element is divided into 2 parts repeated until it is not possible to divide it further.
- In merge sort the elements are split into 2 subarray $(n/2)$ again & again until only one element is left.

Q.12. Selection sort is not stable by default but can you write a version of stable selection sort?

```
for (i=0 ; i<n-1 ; i++)
    {
      min = i;
    for (j = i+1 ; j<n ; j++)
     { if (a[min] < a[j])
          min = j ;
      }
       key = a[min];
       while (min >i)
       {a[min] = a[min-j];
          min--;
    }  a[i] = key;
```

Q. 13. Bubble sort scans array even when array is sorted. Can you modify the bubble sort so that it does not scan the whole array once it is sorted.

A better version of bubble sort, known as a bubble sort, includes a flag that is set of an exchange is made after an entire pass over. If no exchange is maden then it should be called the array is already order because no two elements need to be switched.

```
Void bubble (int a[], int n)
{
  for (i=0; i<n; i++)
  { int flag=0;
  for (j=0; j< n-1-j; j++)
   { if (A[j] > A[j+1])
     {
       t = a[j];
      a[j] = a[j+1];
       a[j+1] = t;
       flag++;
     }
   }
  if (flag==0)
    break;
}
```