Name- Anshul Kumar
Sec- CST SPL-2
Roll no- 11

Design & Analysis of Algorithms

TUTORIAL-3

Ans 1.

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

Ans 2.

```
void insertion(int arr[], int n)
{                   // RECURSIVE
    if (n<=1)
        return;
    insertion(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;
}
```

ANSHUL

```
for (i=1 to n)          //ITERATIVE
{
    key = A[i];
    j = i-1;
    while(j>=0 && A[j]>key)
    {
        A[j+1] = A[j];
        j--;
    }
    A[j+1]=key;
}
```

More input can be inserted with the insertion sorting, it doesn't know the whole Input, Online Sorting.

Ans 3.

Name	Best	Worst	Average
1) Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
2) Bubble	$O(n)$	$O(n^2)$	$O(n^2)$
3) Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
4) Count	$O(n+k)$	$O(n+k)$	$O(n+k)$
5) Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
6) Quick	$O(n\log n)$	$O(n^2)$	$O(n\log n)$
7) Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

ANSHUL

Ans 4.

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

Ans 5.

```
int binary(int arr[], int l, int r, int x)
{                          //Recursive
    if(r>=l){
        int mid = l+(r-l)/2;
        if(arr[mid]==k)
            return mid;
        else if(arr[mid]>x)
            return binary(arr, l, m-1, x);
        else
            return binary(arr, m+1, r, x);
    }
    return -1;
}
```

ANSHUL

// ITERATIVE

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

Time complexity of:

Binary Search ⇒ $O(\log n)$

Linear Search ⇒ $O(n)$

Ans 6. Recurrence relation for Binary Search

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

where, $T(n)$ is the time required for binary search in an array of size 'n'.

Ans 7.

```
int find (A[], n, k){
    Sort (A, n);
    for (i = 0 to n-1)
    {
        x = binarySearch(A, 0, n-1, k-A[i]);
        if (x)
            return 1;
    }
    return -1;
}
```

$$T.C. = O(n \log n) + n \cdot O(\log n)$$
$$= O(n \log(n)).$$

Ans 8.

- QuickSort is the fastest general purpose sort!
- In most practical situations, quickSort is is the method of choice, If stability is important and space is available, merge sort might be best.

Ans 9.

A pair (a[i], a[j]) is said to be inversion if a[i] > a[j].

In arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

Total no. of Inversions are 31, using merge Sort.

Ans 10.

The worst case complexity of quick Sort is $O(n^2)$

This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted. Best case is when we select pivot as a mean element.

Ans 11.

Recurrence relation of :-

① Merge Sort : $T(n) = 2T(n/2) + n$.

② Quick Sort : $T(n) = 2T(n/2) + n$.

- Merge Sort is more efficient and works faster than quick sort in case of larger array size or datasets.

- Worst-case complexity for quick-sort is $O(n^2)$ whereas $O(n \log n)$ for merge sort.

Ans 12. Stable Selection Sort.

```
void stableselection(int arr[], int n){
    for (int i=0; i<n-1; i++){
        int min = i;
        for(int j=i+1; j<n; j++){
            if (arr[min] > arr[j])
                min = j;
        }
```

```
        int key = arr[min];
        while (min > i) {
            arr[min] = arr[min-1];
            min--;
        }
        arr[i] = key;
    }
}
```

Ans 13. Modified Bubble Sorting.

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