Tutorial - 03

Ques 1 Linear Search Pseudo Code :-

```
int LinearSearch (arr, n, key) {
    if (abs(arr[0] - key) > abs(arr[n-1] - key))
        for (i=n-1 to 0; i--)
            if (arr[i] == key)
                return i;
    else
        for (i=0 to n-1 ; i++)
            if (arr[i] == key)
                return i;
}
```

Ques 2 Pseudo Code of Iterative Insertion Sort :-

```
insertionSort (int a[], int n) {
    for (i=1 to n; i++) {
        x = a[i];
        j = i-1;
        while (j > -1 && a[j] > x) {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```

Pseudo code of Recursive insertion sort :-

```
InsertionSort (int a[], int n) {
    if (n <= 1) return;
    InsertionSort (a, n-1);
    int x = a[n-1];
    j = n-1;
    while (j >= 0 && a[j] > x) {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = x;
}
```

Insertion sort is called online sorting because it contains only one input per iteration & produces a partial solution without considering future elements whereas other sorting algorithm process the whole problem data together from all the beginning & is required to output an answer which solve the problem at hand

Ques 3 Complexity of All sorting algorithms:-

Sorting	Best	Worst	Average
1) Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
2) Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
3) Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
4) Quick Sort	$O(n\log n)$	$O(n^2)$	$O(n\log n)$
5) merge sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
6) count sort	$O(n+m)$	$O(n+m)$	$O(n+m)$
7) Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

Ques 4

Sorting technique	Inplace	Stable	online
1) Bubble Sort	✓	✓	X
2) Selection Sort	✓	X	X
3) Insertion Sort	✓	✓	✓
4) Quick Sort	✓	X	X
5) Merge sort	X	✓	X
6) Count Sort	X	✓	X
7) Heap Sort	✓	X	X

Ques 5 Recursive Binary Search Pseudo Code

```
int binarySearch(a, l, r, n){
    while(l<=r){
        mid = (l+r)/2;
        if( n > a[mid])
            return binarySearch(a, mid+1, r, n);
        else if( n < a[mid])
            return binarySearch(a, l, mid-1, n);
        else
            return mid;
    }
}
```

Iterative Binary Search Pseudo code :-

```
int binarySearch (a, n, x) {
    l=0, r=n-1;
    while (l<=r) {
        mid = (l+r)/2;
    if (x< a[mid])
        r=mid-1;
    else if (x > a(mid))
        l= mid+1;
    else
        return mid;
}
```

	Time complexity	Space complexity
i) Linear Search	O(n)	O(1)
ii) Binary Search	O(log n)	O(1)

Ques 6 $T(n) = T(n/2) + 1$

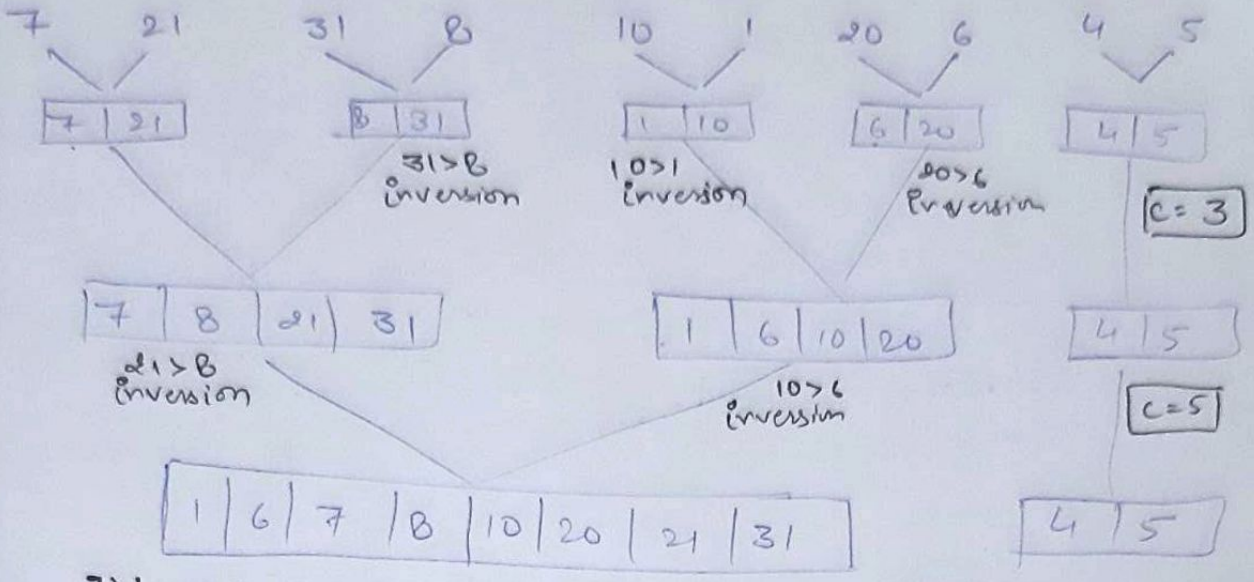
Ques 7.

```
findIndex (int a[], int n, int k){
    i=0, j=1;
    while (i<n && j<n){
        if (i!=j && a[j]-a[i]==k || (a[i]-a[j] ==k))
            printf ("%d %d ",i, j);
        else if(a[j]-a[i]<k)
            j++;
        else  i++;
    }
}
```

Ques 8 Quick Sort is one of the best efficient sorting algorithms which makes it one of the most used as well, it is faster as compared to other sorting algorithm. Also its time complexity is O(n log n). But in case of a ~~big~~ larger array, Merge sort is preferred.

Ques 9

Inversions is an array basically define how far or close an array is from being sorted. If array is in reverse order, inversion count → maximum.



7>1, 7>6, 8>1, 8>6, 21>10, 21>20, 31>1, 31>6, 31>10, 31>20

c=17

1 | 4 | 5 | 6 | 7 | 8 | 10 | 20 | 21 | 31

6>4, 6>5, 7>4, 7>5, 8>4, 8>5, 10>4, 10>5, 20>4, 20>5, 21>4, 21>5
31>4, 31>5

c=14

total = 14 + 17 = 31

Ques 10

Best Case:- if ~~partially~~ partioning element is in the middle.

Time complexity = $O(n \log n)$

Worst Case:- if pivot is at extreme position & array is already sorted in increasing / decreasing order

Time complexity = $O(n^2)$

Ques 11

Recursive Relations of Merge & Quick Sort

Quick Sort:- Best : $T(n) = 2T(n/2) + n$

Worst : $T(n) = T(n-1) + n$

Merge Sort:- $T(n) = 2T(n/2) + n$

in merge sort, the array is divided into 2 equal halves n times.

$\therefore$ T.C. = $O(n \log n)$

In Quick sort, the array is divided into any ratio depending on the position of pivot element

$\therefore$ Time complexity varies from $O(n^2)$ to $O(n \log n)$.

Ques 12. Selection sort is not a stable sort but you can write a version of stable selection sort.

In selection sort, normally we swap the minimum value with the first value, which makes it unstable. to make it stable, instead of swapping, insert the least value at position – 0 to n.

Ques 13. Bubble sort scans whole array when array is sorted. Can you modify the bubble sort so that it doesn't scan whole array.

```
void bubbleSort (int a[], int n){
    for(i=0 to n){
        swaps=0;
        for(j= 0 to n-1-i)
            if (a[j] > a[j+1]) {
                swap(a[j], a[j+1]);
                swaps++;
            }
        if (swaps==0)
            breaks;
    }
}
```

Ques 14

In such cases, external sorting algorithms such as K-way merge sort is used that can handle large data amounts which can't fit into main memory. A part of array resides in RAM during the execution whereas in Internal sorting, process takes place entirely within the main memory, mainly used data to be sorted in small.

Eg:- Bubble sort, Quick Sort etc.