Tutorial→03

Name → Mohd Nasir
Sect → 'F'
Roll → 14
Uni Roll → 2016855

<u>Ans:-01</u>

```
while (low ≤ high)
{
  mid = low+(high-low)/2;
  if (arr[mid] == key)
  {
    Print("Found");
    Print (Count);
  }
  else if (arr[mid] > key)
        high = mid-1;
  else
        low = mid+1;
  Count++;
}
print ("Not found");
```

Ans:-02 Iterative Insertion Sort:-

```
for (int i=1; i<n; i++)
{   j=i-1;
    x=arr[i];
    while (j>-1 and Arr[j]>x)
    {
        A[j+1] = A[j];
        j--;
    }
    Arr [j+1]= x;
}
```

Recursive:-

```
Void Insertion Sort (int arr[], int n)
{
    if (n≤1)
        return;
    insertionSort(arr, n-1);
    int last= arr[n-1];
    j=n-2;
    while(j>=0 and arr[j]>last)
    {
        arr [j+1] = arr[j];
        j--;
    }
    arr[j+1] = last.
}.
```

Insertion sort is online sorting because whenever a new element come, insertion sort define its right place.

Ans :-03 Bubble Sort → $O(n^2)$
Insertion sort → $O(n^2)$
selection sort → $O(n^2)$
Merge Sort → $O(n\log n)$
Quick sort → $O(n\log n)$
count sort → $O(n)$
bucket sort → $O(n)$.

Ans :-04 Online sorting :→ Insertion sort.
Stable Sorting :→ Merge sort, Bubble sort.
Inplace sorting → bubble sort, selection sort.

Ans :-05 Iterative Binary search.

```
while(low<=high)
{
   mid = low + (high + low)/2;
   if(arr[mid] == key)
       return true;
   else if (arr[mid]>key)
       high = mid-1;
   else
       low=mid+1;
}
return false;
```

$O(n\log n)$

Recursive Binary Search.

```
if(low <= high)
{
    int mid = low + (high+low)/2;
    if(arr[mid] == key)
        return true;
    else if (arr[mid] > key)
        Binarysearch(arr, low, mid-1);
    else
        Binarysearch(arr, mid+1, high);
}
return false;
```

Ans:>6 $T(n) = T(n/2) \cancel{+} T(n/2) + C$

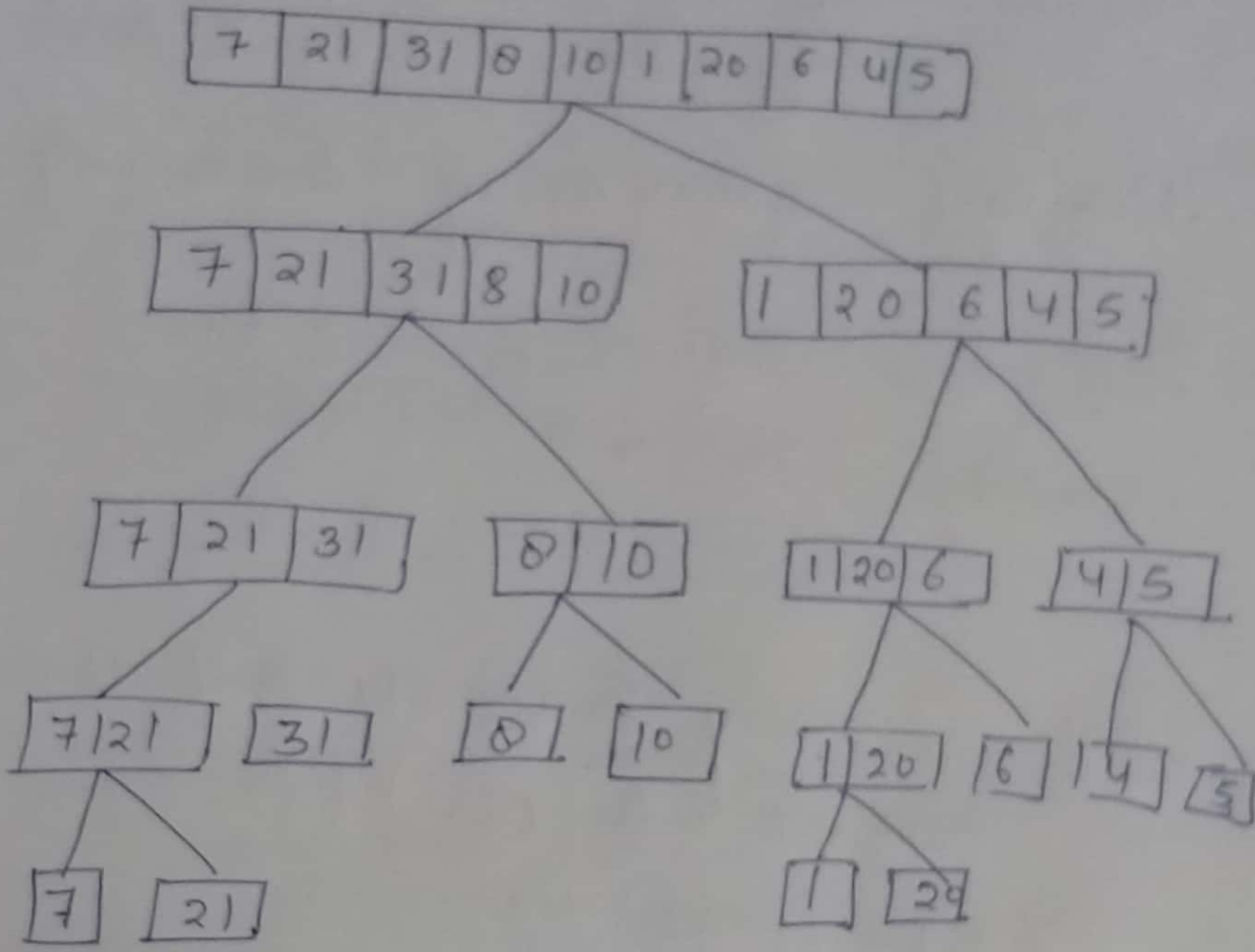
$$\boxed{T(n) = T(n/2) + 1}$$

Ans:>7

```
void (vector<int> arr, int key)
{
    sort(arr.begin(), arr.end());
    int low = 0, high = arr.size()-1;
    while(low < high)
    {
        if (arr[low] + arr[high] == key){
            print(low, high)
            Break;
        }
        else if (arr[low] + arr[high] > key)
            high--;
        else
            low++;
    }
    print("No found");
}
```

Ans:08 Quick Sort is the fastest general purpose Sort In most practical solution, quicksort is the Method of choice if stability is important and space is available, merge sort might be best.

Ans:09 Inversion indicates how far are close the array is from being sorted.



Inversion → 31

Ques→11 Merge sort

Best Case → $2T(n/2) + O(n)$

Worst Case → $2T(n/2) + n$

Quick sort.

Best Case → $2T(n/2) + n$

Worst Case → $T(n-1) + n$.

Basis	Quick sort	Merge sort.
· Position	Splitting done in any ratio.	Array divided in two equal parts.
· Works well on	smaller array.	fine on any size of array.
· Sorting Method	Internal	External.
· Stability	Not Stable	stable.

Ques→10 if array is already sort than Quick Sort will take $O(n^2)$ time. otherwise it takes $(n\log n)$ time.

Ans 12:-

```
Void StableSelectionSort(int arr[], int n)
{
    for(int i=0; i<n-1; i++)
    {   int Pos=i;
        int min = i;
        for(int j=i+1; j<n; j++)
        {   if (arr[j]<min)
            {
                min = arr[j];
                Pos = j;
            }
        }
        if(Pos!=i)
        while( Pos>i)
        {
            arr[Pos]=arr[Pos-1];
            Pos--;
        }
        arr[i]=min;
    }
}
```

Ans→13 We will use the merge sort because we can divide the 4GB data into 4 parts of 1 GB and sort them. Separately and combine them latter.

Internal sorting→ All the data is sorted in memory at all times sorting is in progress.

External sorting→ all the data is stored outside memory and only load in small parts.