# TUTORIAL-3

**Ans→1)**

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

**Ans-2)** Iterative Insertion Sort :

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

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

Recursive Insertion Sort :

```
void insertionsort (int arr[], int n)
{
    if (n<=1) return;
    insertionsort (arr, n-1);
    int last = arr [n-1] ;
    j=n-2 ;
    while (j>=0 && arr[j] > last)
    {
        arr[j+1] = arr [j];
        j--;
    }
    arr [j+1] = last
}
```

Ans→ 3)

Bubble Sort → $O(n^2)$

Insertion " → $O(n^2)$

Selection " → $O(n^2)$

Merge " → $O(n\log n)$

Quick " → $O(n\log n)$

Count Sort → $O(n)$

Bucket Sort → $O(n)$

Ans-4)

Online Sorting → Insertion Sort

Stable Sorting → Merge Sort, Insertion, Bubble Sort

Inplace Sorting → Bubble, Insertion, Selection sort

Ans → 5) BINARY SEARCH → O(log n)

**Iterative**

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

**Recursive**

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

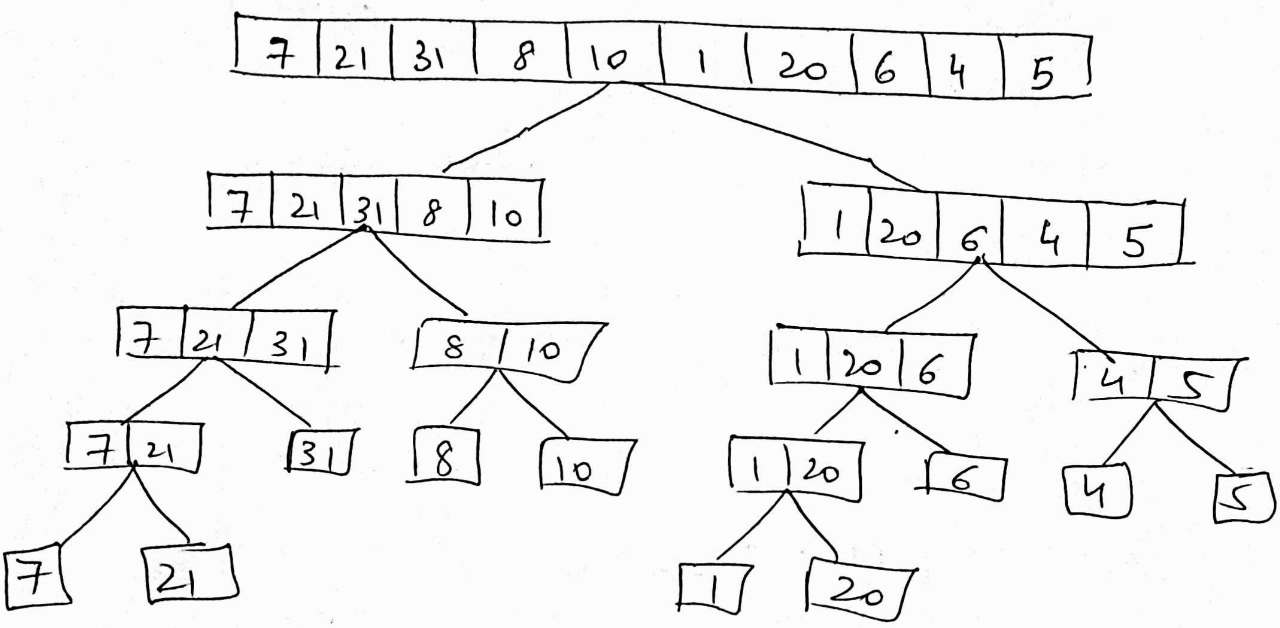
Ans → 6) $T(n) = T(n/2) + T(n/2) + c$

Ans → 7)

```
map <int, int> m;
for (int i=0; i< arr.size(); i++) {
    if (m.find (target-arr[i])= m.end())
        m[arr [i]]=i;
    else
        cout <<i<< "  " <<mp[arr[i]];
}
```

Ans→8) Quicksort is the fastest general purpose sort. In most practical situation it is the method of choice. If stability is imp. & space is available mergesort might be best.

Ans-9) Inversion indicates how far or close the array is from being sorted. (n=10)



Inversions = 31

Ans-10) Worst Case → It occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted as reverse sorted & either first or last element is picked as pivot.

$O(n^2)$

Best Case → Pivot middle or near to middle element

$O(n \log n)$

Ans-11) Merge sort → $T(n) = 2T(n/2) + O(n)$

Quick sort → $T(n) = 2T(n/2) + n + 1$

| Basis | Quick Sort | Merge Sort |
|---|---|---|
| • Partition | Any ratio | Equal 2 halves |
| • Works well on | Smaller array | Any size |
| • Additional Space | Less (in-place) | More |
| • Efficient | Inefficient for larger array | More efficient |
| • Sorting Method | Internal | External |
| • Stability | No | Stable |

Ans→14) We'll use merge sort ∵ we'll divide the data (4GB) into 4 pockets of 1GB & sort them separately & combine them later.

Internal Sorting → all the data to sort is stored in memory at all times while sorting is in progress.

External Sorting → all the data is stored outside memory & only loaded into memory in small chunks.