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;
}
```

Recursive insertion:-

```
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];
}
arr[j+1] = last;
}
```

Ans-3 Bubble sort - $O(n^2)$, Insertion sort - $O(n^2)$
selection sort - $O(n^2)$, Heap sort - $O(n \log n)$
Quick sort - $O(n \log n)$, Count sort - $O(n)$
Bucket sort - $O(n)$

Ans-4 Online sorting - Insertion sort
Stable sorting - mergesort, Insertion sort, Bubble sort
Inplace sorting - Bubble sort, Insertion sort, Selection sort.

Ans-5 Iterative Binary search :-

```
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;
}
```

Recursive Binary search :-

```
while( low<= high)
{
  int mid = (low+high)/2;
  if (arr[mid] == key) return true;
```

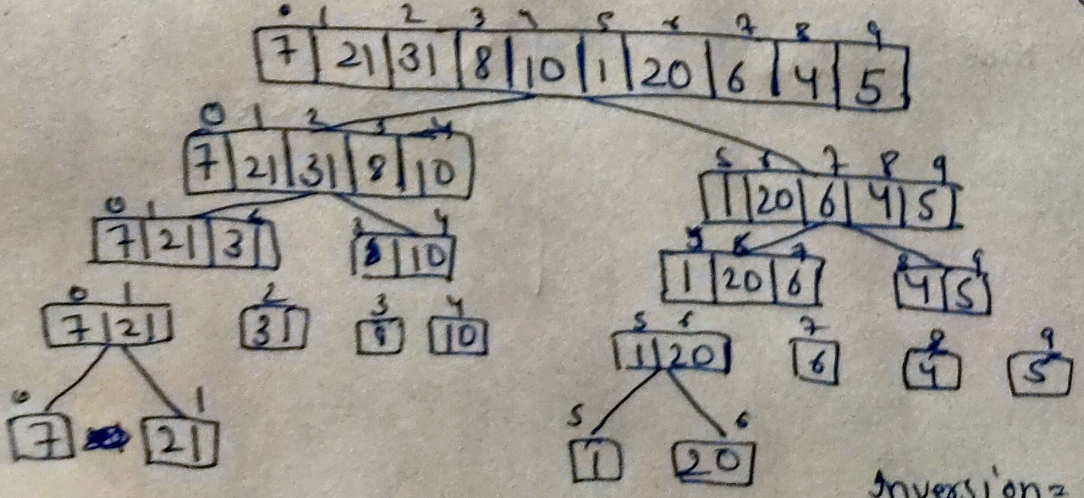
```
    else if (arr[mid] > key)
        high = mid - 1;
    else
        low = mid + 1;
}
return false;
```

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 << " " << m[arr[i]];
}
```

Ans-8 Quicksort is the fastest general purpose sort in most practical situation, quick sort is the method of choice. If stability is important & space is available merge sort might be best.

Ans-9 Inversion indicates - how far or close the array is from being sorted :-



Inversion = 31

Ans-10 Worst Case :- The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted & either first or last element is picked as pivot $O(n^2)$

Best case :- Best case occurs when pivot element is the middle element or near to the 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	Splitting is done in any ratio	array is parted into just 2 halves.
• Works well on	smaller array	live on any sized array.
• Additional space	less (inplace)	more (not inplace).
• Efficient	inefficient for larger array	more efficient
• Sorting method	Internal	External
• Stability	Not stable	Stable