Tutorial-3 DAA

Q-1)
Ans:

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

Q-2)
Ans: Iterative Insertion Sort:

```
for(int i=1; i<n; i++){
        j= i-1;
        x= A[i];
        while( j>0 && A[j]>n)
        {
            A[j+1]= A[j]
            j--;
        }
        A[j+1]=n;
}
```

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;
}
```

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

Ans-3:
Bubble sort : $O(n^2)$
Insertion sort : $O(n^2)$
Selection sort : $O(n^2)$
~~selec~~ Merge 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 → Merge, Insertion, Bubble sort
Inplace sorting → Bubble, Insertion, Selection sort.

Ans-5:
Iterative Binary Search

$O(\log n)$

```
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

$O(\log n)$.

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

Ans-6: $T(n) = T(n/2) + T(n/2) + C$.

Ans-8:

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

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

Total inversions in this case is 31 (using mergeSort)

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 as reverse and either first or last element is picked as pivot. $O(n^2)$.

Best case: Best case occurs when Pivot element is the middle element as new to the middle element.

$O(n \log n)$

Ans-11:

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

Quick Sort: $T(n) = 2T(n/2) + n+1$.

	Quick Sort	Merge Sort
• Partition	Splitting done in any ration.	Must split in two halves.
• Works well	Smaller array	Fine on any size.
• Efficient	Large space	More efficient
• Sorting method	Internal	External
• Stability	Not stable	Stable.