Tutorial -3

Ans1)

```
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;
```

Ans2) Iterative insertion sort:

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

Recursive Insertion sort

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

```
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;
}
```

Ans3) Bubblesort — $O(n^2)$

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 — $O(n)$

Ans4) Online sorting → Insertion sort

Stable sorting → Merge sort

Inplace sorting → Bubble sort, Insertion sort, Selection sort

Ans5) 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 :

$O(\log n)$

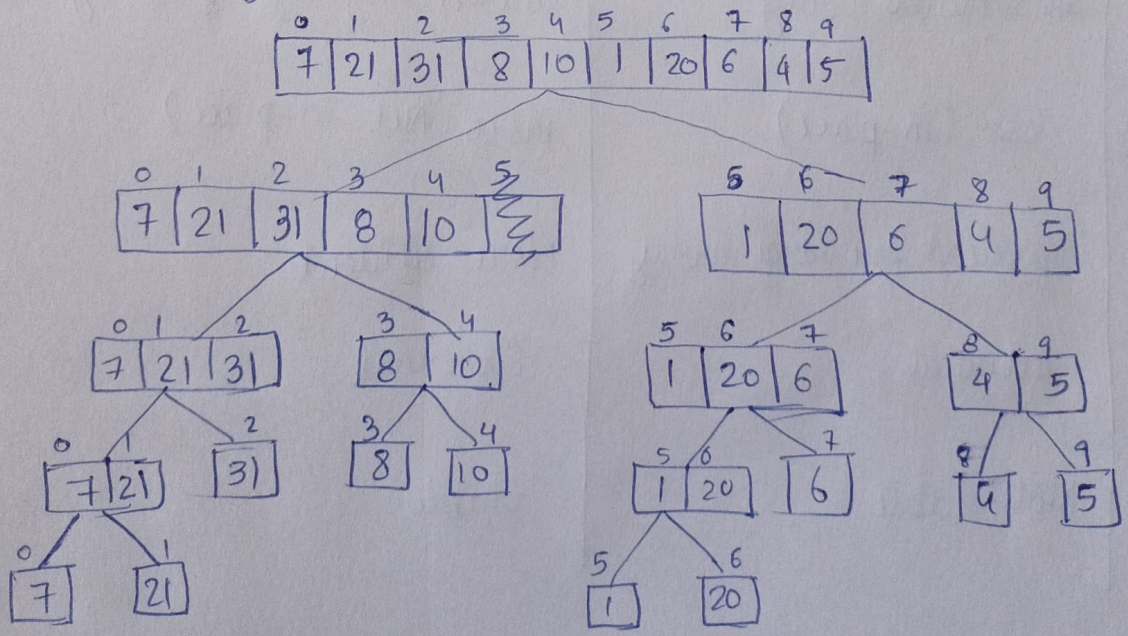
```
while (low <= high)
{ int mid = (low + high) / 2
  if (arr[mid] == key)
      return true;
  else if (arr[mid] > key)
      Binary search (arr, low, mid-1);
  else
      Binary search (arr, mid+1, high);
}
return false;
```

Ans6) $T(n) = T(n/2) + T(n/2) + C$

Ans7)
```
map<int, int> m;
for (int i=0; i<arr.size(); i++)
{
  if (m.find(target - arr[i]) = m.end())
      m[arr[i]] = 1;
  else
  {
    count<< i<< "  " << mp[arr[i]];
  }
}
```

Ans8) Quicksort is the fastest general purpose sort. In most practical situation, quicksort is the method of choice. If stability is important and space is available, mergesort might be best sort.

Ans9) Inversion indicates — how far or close the array is from being sort



Inversion = 31

Ans10) 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 sorted 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 near to the middle element.

$O(n \log n)$

Ans11) Merge sort: $T(n) = 2T\left(\frac{n}{2}\right) + O(n^2)$

Quick sort: $T(n) = 2T\left(\frac{n}{2}\right) + n + 1$

Basis	Quick sort	Merge sort
• Partition	Splitting is done in any ratio.	Array is parted into just 2 halves
• Works well on	smaller array	fine on any size of array.
• Addition of space	less (in-place)	More (Not in-place)
• Efficient	inefficient for large array	More efficient
• Sorting Method	Internal	External
• Stability	Not stable	Stable

Ans14) We will use Merge sort because we can divide the 4GB data into 4 Packets of 1 GB and sort them separately and 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 and only loaded into memory in small chunks.