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 a new element come, insertion sort defines 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

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 - $O(n)$

Ans4

Online Sorting → Insertion sort
Stable sorting → Merge sort, Insertion sort, Bubble 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)
 Binarysearch (arr, low, mid-1)
else
Binary-Search (arr, mid+1, high);
}
return false;
```

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

Ans7

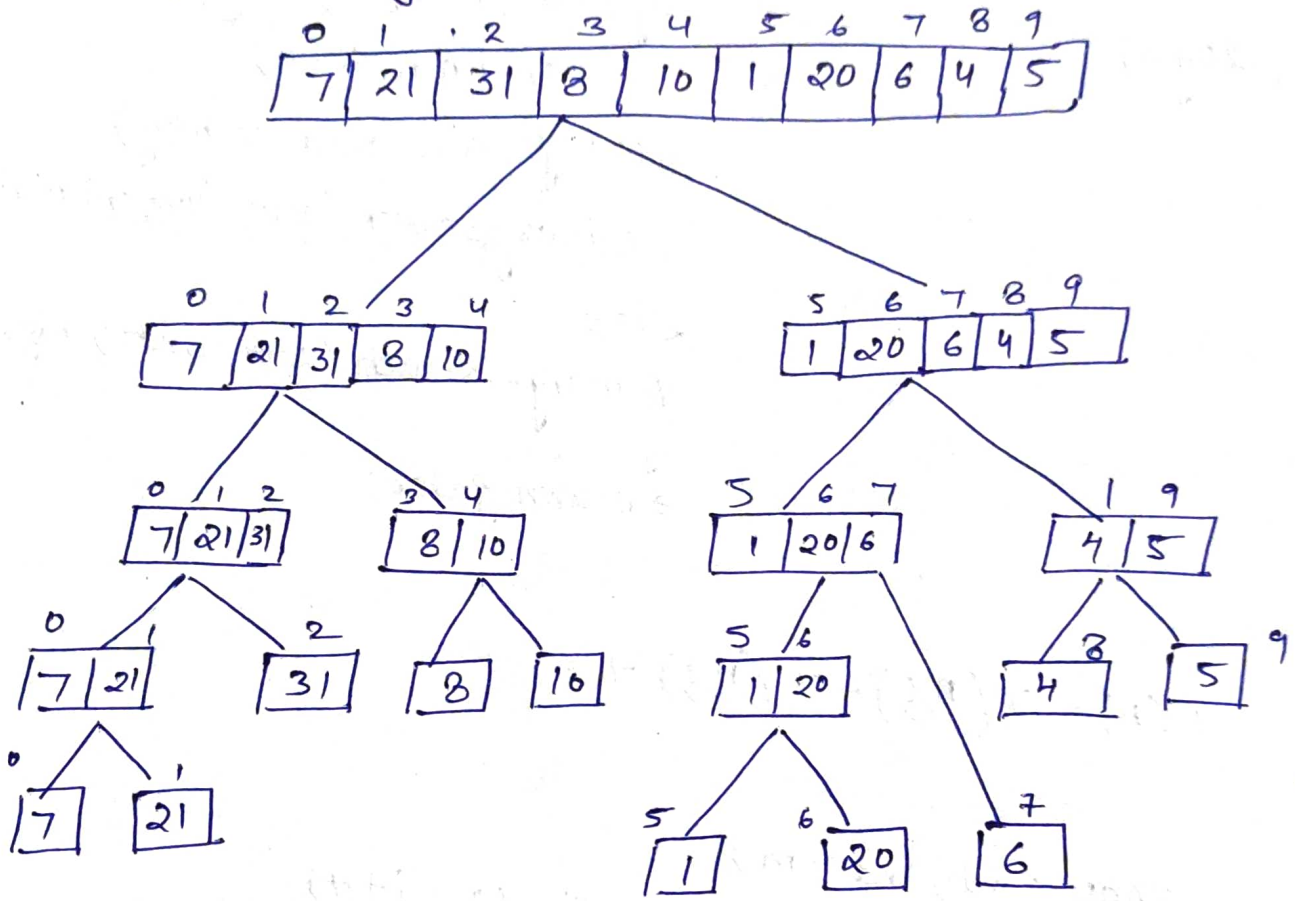
```
map<int, int> m;
for (int i=0; i< arr.size() ; i++)
{ if (m.find (target - arr[i]) = m.end())
        m[arr[i]] = i

  else
     cout << i << " " << mp[arr[i]];
  }
}
```

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

Ans-9

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



Inversion = 31

Ans-10

Worstcase :- The worstcase 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 :- Bestcase occurs when Pivot element in the middle element as near to the middle element. $O(n \log n)$

Ans 11

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

Quick sort :- $T(n) = 2T(n/2) + n + 1$

Basic	Quicksort	Mergesort
• 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.
• Additional Space	Less (in-place)	More (Not-in-place)
• Sorting Method	inefficient for larger array	More efficient
• Stability	Internal	External
	Not stable.	Stable

Ans 14 We will use Mergesort because we can divide the 4 GB data into 4 packets of 1 GB and ~~sort~~ them seperately and combine them latter.

- 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.