Tutorial -3

Answer 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;
```

Answer 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 sort:

```
void insertion sort (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 online sorting because whenever a new element come, insertion sort define its right place.

Answer 3) → 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 − $O(n)$.

Answer 4) → Online Sorting → Insertion Sort
Stable Sorting → Merge Sort, Insertion Sort, Bubble Sort.
Inplace Sorting → Bubble Sort, Insertion Sort, Selection Sort.

Answer 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)
  Binary Search (arr, low, mid-1);
  else
  Binary - Search (arr, mid+1, High);
} return false;
```

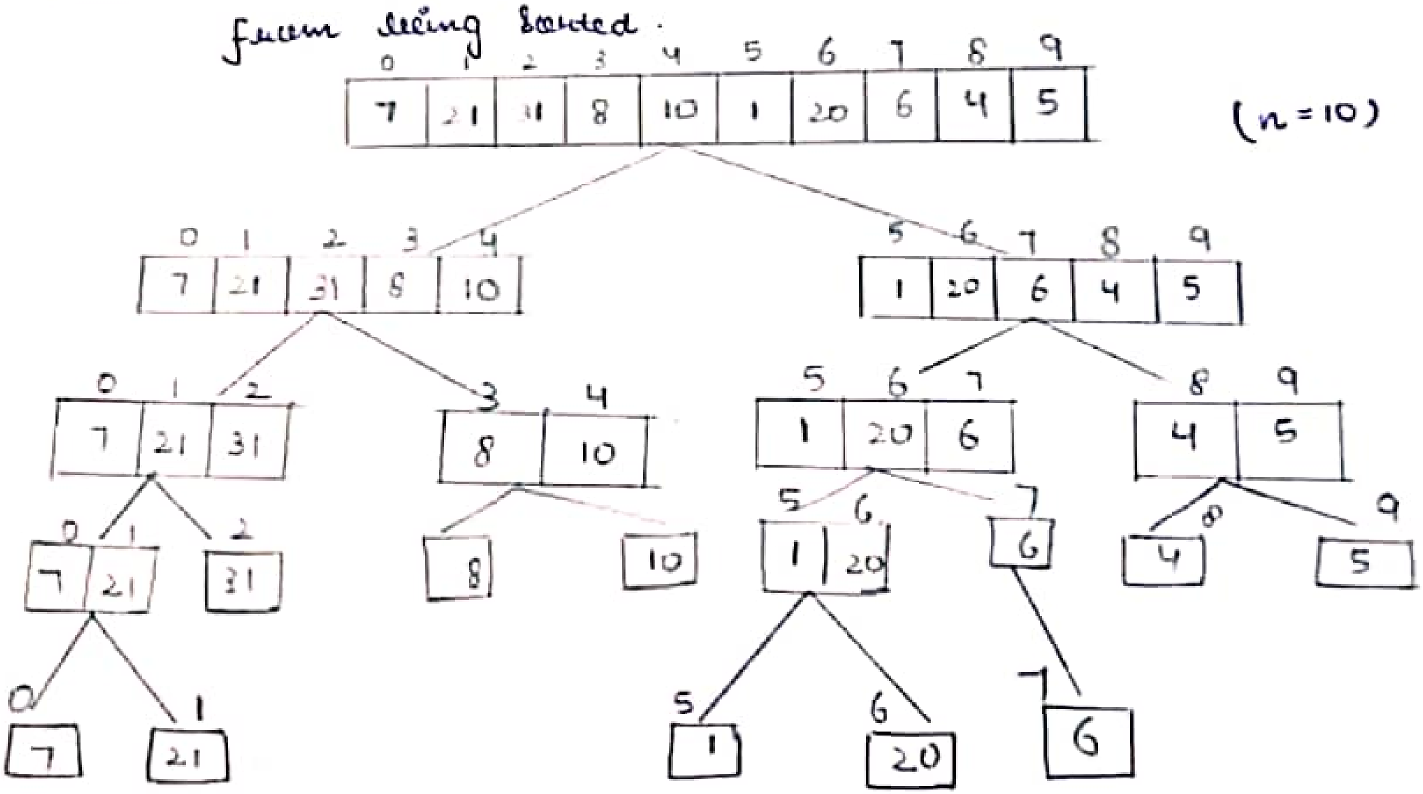
Answer 6) → $T(n) = T(n/2) + T(n/2) + c$.

Answer 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]];
    }
}
```

Answer 8) → Quick sort is the fastest general Purpose sort. In most - Practical situation, quicksort is the method of choice. It stability is important and space is available, merge sort might be best.

Answer 9) → Inversion Indicates → How far close the array is from being sorted.



(n=10)

Inversion : 31.

Answer 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 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 or near to the middle element.

$O(n \log n)$.

Answer 11) → Merge Sort $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

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	Less (In-place)	More (Not in-place).
⊙ Efficient	inefficient for larger array.	More efficient.
⊙ Sorting Method	Internal	External.
⊙ Stability	Not Stable	Stable.

Answer 14) → We will use Merge Sort because we can divide the 4 GB data into 4 Packets of 1GB and Sort them separately and combine them later.

⊙ Internal Sorting: all the data to Set 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 Smaller chunks.