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 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 online sorting because whenever a new element come, insertion sort defines its right place.

Ans-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 – $O(n)$

Ans-4:
- Online sorting → Insertion sort
- Stable sorting → Merge sort, Insertion, Bubble sort
- Inplace sorting → Bubble sort, Insertion sort, 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:

```
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
```

O(log n)

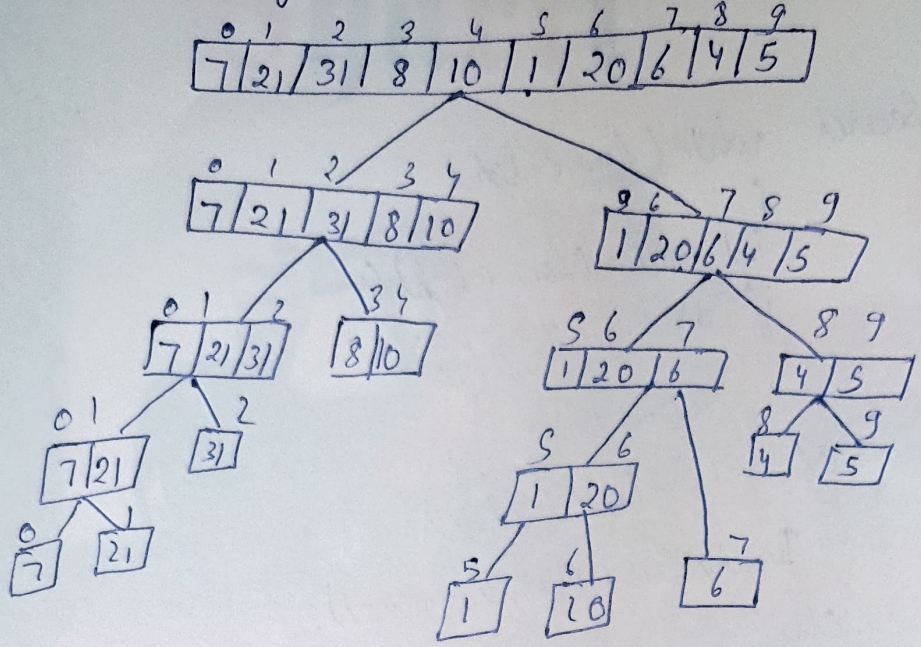
Ans-6: $T(n) = T(n/2) + T(n/2) + C$

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]] = 1
  else
  {
  cout << i << " " << mp[arr[i]];
  }
}
```

Ans-8: Quick sort is the fastest generates purpose sort. In most practical selection, quick sort is method of choice. If stability is important and space is available, merge sort might be best.

Ans-9: Inversion indicates - how far 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/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

$O(n \log n)$

Ans-11: Merge Sort: $T(n) = 2T(\frac{n}{2})$ to n

Quick Sort: $T(n) = 2T(\frac{n}{2}) + n + 1$

Ans-11: Merge Sort: $T(n) = 2T(n/2)$ to n

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

Basics	Quick Sort	Merge Sort
• Partition	Splitting is done in array ratio	array is parted into 2 halves
• Works well on	Smaller array	fine on any size of array.
• Addition Space	Less (Inplace)	• More (Not Inplace)
• Efficient	Inefficient for larger array	More efficient.
• Sorting Method	Internal	External
• Stability	Not Stable	Stable.

Ans-14: We will use Merge sort 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 loads into memory in small chunks.