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 define 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 - $O(n)$

Ans 4 Online sorting - Insertion

Stable sorting -> Merge Sort, Insertion sort, 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

$O(\log n)$

```
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
  BinarySearch(arr, mid+1, high)
}
return false;
```

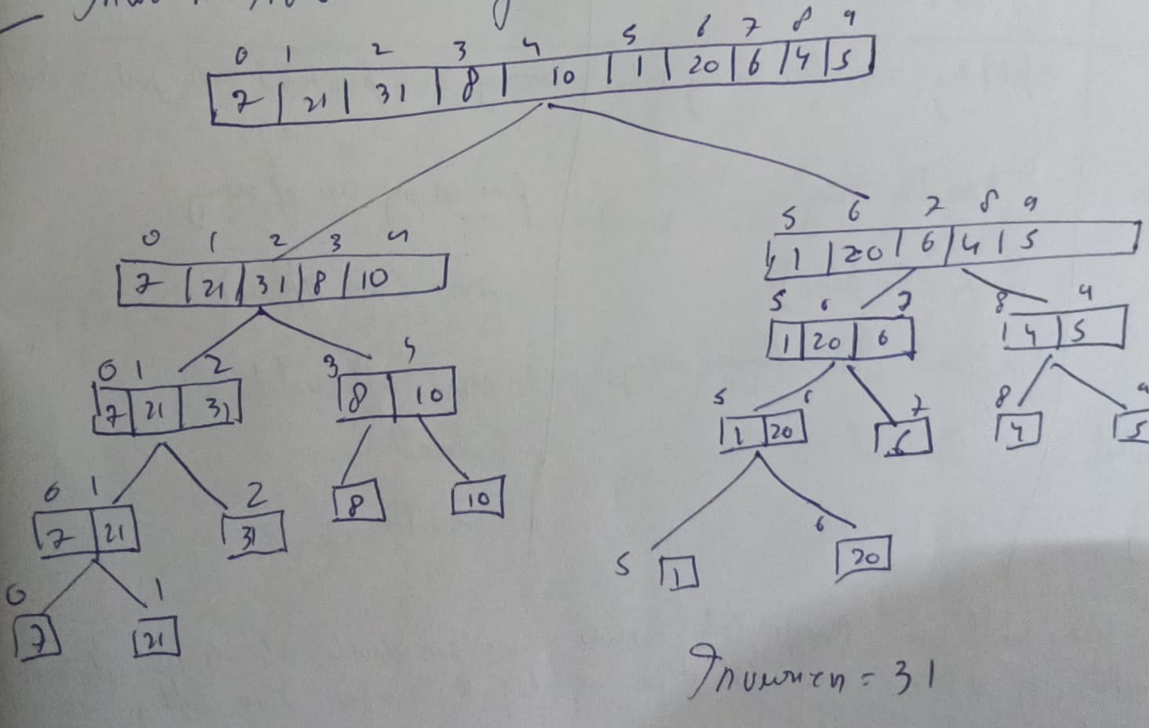
Ans 6 $T(n) = T(n/2) + T(n/2) + c$

7 map<int, int> mp;

```
for(int i=0 ; i<arr.size() ; i++)
{ if (mp.find (target - arr[i]) = m.end())
     m[arr[i]] = i;
  else {
     cout << i << " " << mp[arr[i]];
  } }
```

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

9) Inversion Indicates - how far or 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 or largest) element. This happens when I/P array is sorted or reverse sorted & 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 $O(n \log n)$

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

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

Basis	Quick Sort	Merge Sort
• Partition	splitting is done in any ratio	array is partitioned into just 2 halves
• works well on	smaller array	fine on any size of array
• Additional space	Less (In place)	More (not inplace)
• Efficient	inefficient for larger array	More efficient.
• Sorting method	Internal	External
• Stability	Not stable	stable

Ans 12 we will use Merge Sort because we can divide the 4GB data into 4 packets of 1GB and sort them separately & 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 m/m & only loaded into memory in small chunks.