Tutorial 3 { Jagtesh Kumar Dimri (Sec: K)
Roll no = 08 Sem = 4th }

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++)
{
    i = 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 adjust into 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 Sort, Bubble Sort.

Inplace Sorting : Bubble Sort, Insertion sort, Selection sort.

Ans 5 : Iterative Binary Search :

```
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;
}
```

$T(n) = O(\log n)$

Recursive Binary Search.

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

$T(n) = O(\log n)$

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

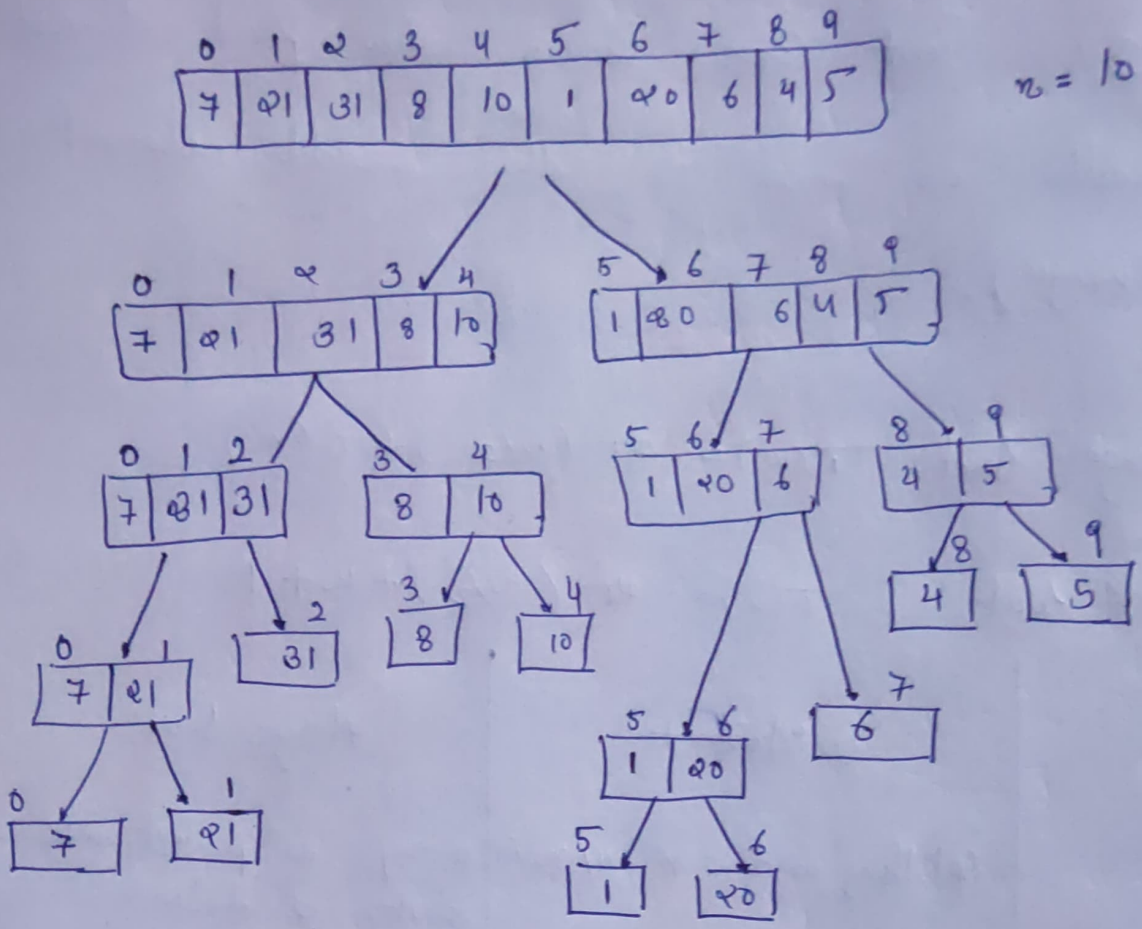
Ans 7

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

Ans 8 Quick sort is the fastest general purpose sort. So most practical solution. quicksort is the method of choice.

If stability is important & space is available, mergesort might be the best.

Ans 9 : Succession Indicates : How far as close the array is from being sorted.



Inversions = 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 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 middle or near to the middle element. O(nlogn).

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

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

Basis	QuickSort	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
• 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. Selection Sort can be made stable if instead of swapping, the minimum element is placed in its position without swapping i.e. by placing number in its position by pushing every element one step forward.

Ans 13. We can set a flag one & if after any pass there is no swap. It means, the array has been sorted & we can break out of the loop.

Ans 14. We will use Merge sort because we can divide the 4 GB data into 4 packets of 1 GB & sort them separately & 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 & only loaded into only in small chunks.