ASSIGNMENT :- 03

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(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 a[], int n)
 { if (n <= 1)
    return;
  insertion (a, n-1);
  int last = a[n-1];
  i = n-2;
  while (j >=0 &&
      arr[j] > last)
  { a[j+1] = a[j];
     j--
  }
  a[j+1] = last;
  }
```

It is a kind of ONLINE SORTING because whenever a new element comes, insertion sort defines its right place.

Ans 3.)

Sorting	Time Complexity
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	$O(n+k)$
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;
}
```

] - O(logn).

Recursive Binary Search:-

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

} - O(log n)

Ans 6) Required eqⁿ :-

$$T(n) = T(n/2) + T(n/2) + C$$

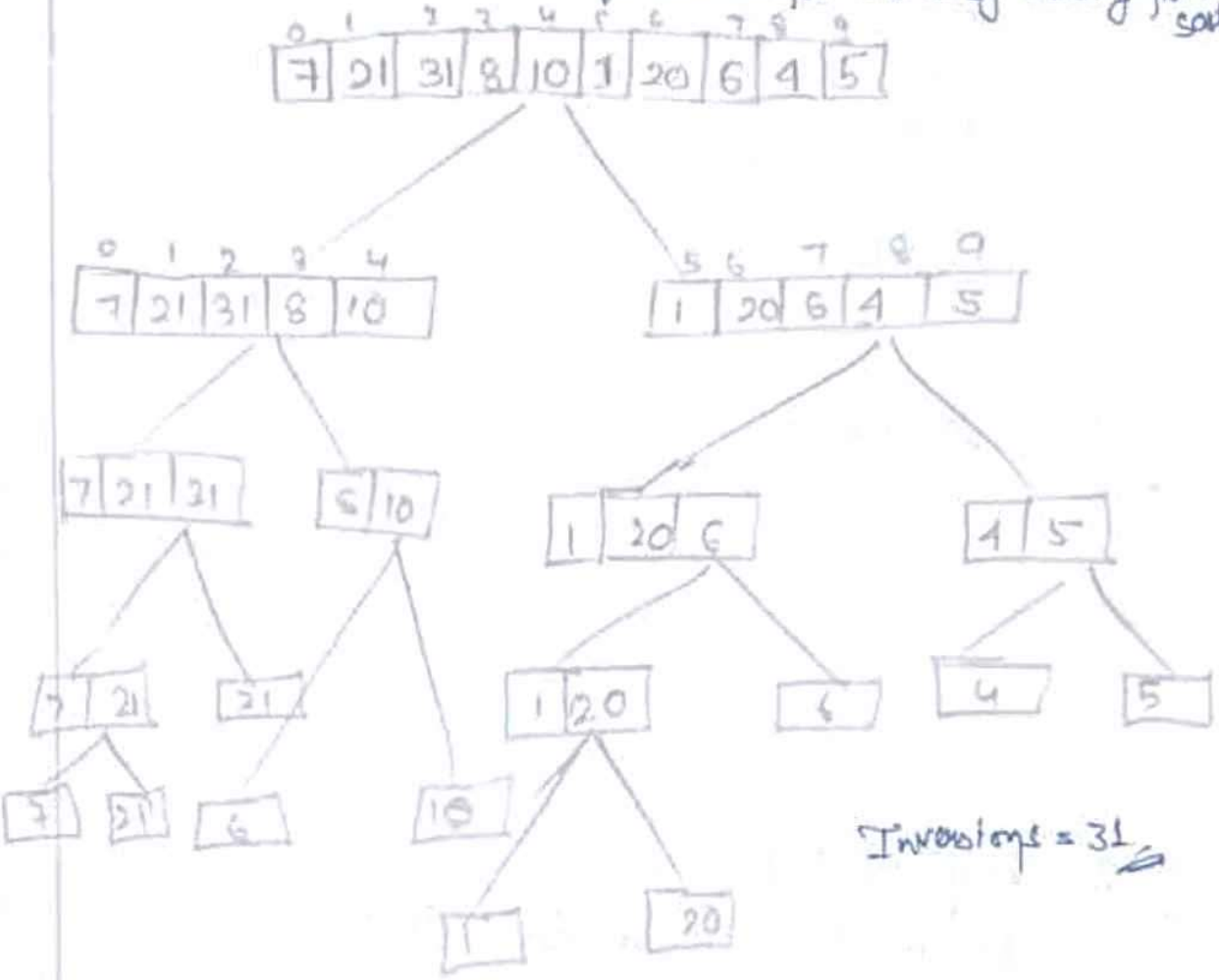
$$\boxed{T(n) = 2T(n/2) + C}$$

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 << " " << map[arr[i]];
  }
}
```

Ans 8) Quick sort is the fastest general purpose. In most practical situations, Quick sort is the method of choice. If stability is important and space is available, Merge sort is best.

Ans 9) Inverse indicates :- It indicates how close or far the array is being from sorted.



Inversions = 31

Ans 10) Worst case :-

It occurs when the pivot element is always an extreme (smallest or largest) element. This happens when input array is sorted and reverse sorted and either first or last element is picked as pivot.

$O(n) = n^2$

Best case :-

It occurs when pivot element is the middle element or near to the middle element.

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

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

Parameter	Quick sort	Merge Sort
Partition	Splitting is done in any ratio.	Array is parted into just two halves.
Working	Smaller array	Fine on any size of array
Add'l space	less (inplace)	More (not inplace).
Efficiency	Ineffecient on larger array.	effective on all types of array.
Sorting Method	Internal	External.
Stability	Not stable	Stable.

Ans 12) Stable selection sort :-

```
void StableSelectionSort (int a[], int n)
{
for (i = 0 to n-1; i++)
{
int min = i;
for (j = i+1 to n; j++)
  if (a[min] > a[j])
    min = j;
  int key = a[min]
  while (min > i)
{
a[min] = a[min-1];
  min--;
}
```

Ans 13)

```
void BubbleSort (int arr[], int n)
{
int i,j;
bool swapped;
for (i=0 to n-1; i++)
{
 swapped = false;
for (j= 0 to n-i-1; j++)
 {if (arr[j] > arr[j+1])
 { swap(arr[j], arr[j+1]);
   swapped = true;
 }
}
  if (swapped==false)
     break;
}
```

Ans 14)

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

Internal Sorting:-
All the data is sorted in memory at all times during sorting is in progress.

External Sorting:-
All the data is sorted is outside memory and loaded into memory in small chunck's.