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]>n)
    {
        A[j+1]=A[j];
        j--;
    }
    A[j+1]=n;
}
```

P.T.O →

Anushka

```
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.

Ans3.

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: $O(n)$

Ans4.

Online sorting → Insertion Sort
Stable Sorting → Merge Sort, Insertion Sort, Bubble Sort
Inplace Sorting → Bubble sort, Insertion Sort, Selection Sort.

Anushka

Ans 5. Iterative Binary Search:

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

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

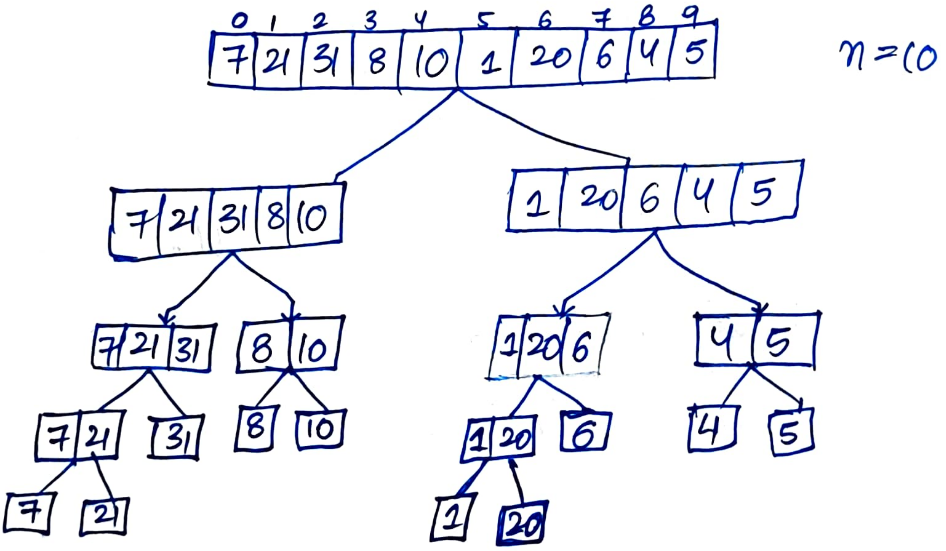
Anushka

Ans7.

```
map <int,int> m;
for (int i=0; i<arr.size(); i++)
  {
    if(m.find(target-arr[1])== m.end())
          m[arr[i]] =1;
     else {
        cout <<i<<" "<<m[arr[i]];
      }
  }
```

Ans8. Quicksort is the fastest general purpose sort. In most practical solution, quicksort is the method of choice. If stability is important and space is available, merge sort might be best.

Ans9. Inversion indicates - how far or close the array is from being sorted



Inversion = 31

Anushka

Ans 10.

Worst case: The worst case occurs when the picked pivot is always an extreme (smallest or largest) elements. 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. $O(n\log n)$

Ans 11. Merge Sort: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

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

Basis	QuickSort	MergeSort
• 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 and space	Less (in-place)	More (not in-place)
• Efficient	ineafficient for larger arrays	More efficient
• Sorting Method	Internal	External
• Stablity	Not Stable	Stable

Anushka

Ans12. Selection sort can be made stable if instead of swapping, the minimum element is placed in its position without swapping i.e., by placing the number in its position by pushing every element one stop forward.

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

Ans13. We will use Merge Sort because we can divide the 4GB data into 4 packets of 1GB & sort them separately and combine them latter.

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

External Sorting:- all the data is stored outside memory and only loaded in small chunks.

Anushka