Tutorial-3 (DAA)

Name- Shubham Gupta
Section- F
Roll No. -28
Uni Roll NO. - 2017050

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 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 comes, insertion sort define its right place.

Ans-3

Bubble Sort → O(n²)
Insertion Sort → O(n²)
Selection Sort → O(n²)
Merge Sort → O(n*logn)
Quick Sort → O(nlogn)
Counting Sort → O(n)
Bucket Sort → O(n)

Ans-4

Online Sorting → Insertion Sort
Stable Sorting → Merge Sort, Insertion Sort, Bubble So
Inplace Sorting → Bubble, Insertion, 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(logn)

```
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;
```

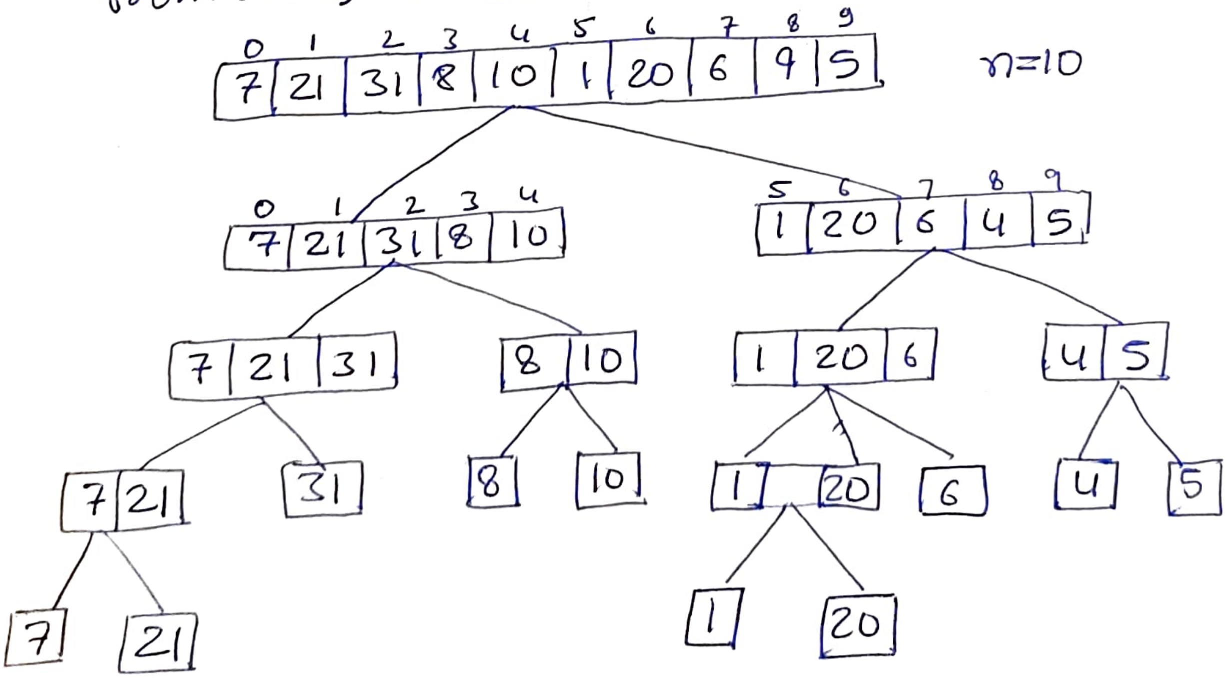
Ans-6 $T(n) = T(n/2) + T(n/2) + C$

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

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

Ans-9 Inversion indicated how fast 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 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 :** $T(n) = 2T(\frac{n}{2}) + O(n)$

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

Basis	Quick Sort	Merge Sort
• Partition	splitting is done in any ratio	array is parted into just two halves
• works well on	smaller arrays	fine on any size of array
• Additional space	Less (in-place)	more (not-in-place)
• efficient	inefficient for large arr.	more efficient
• Sorting method	Internal	External
• Stability	Not Stable	Stable

Ans-14 We will use merge sort because we can divide the 4GB data into 4 packets of 1GB & sort them separately & combine them latter

→ Internal Sort: All the data to sort is stored in memory at all time while sorting in progress

→ External Sort: All the data is stored outside memory & only loaded into memory in small chunks.

Ans-12

```
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;
      }
   }
```

Ans-13

```
void bubbleSort(int a[], int n)
{ int f=0;
  for(int i=0;i<n-1;i++)
  {
     f=0;
    for(int j=0; j<n-1-i ; j++)
    {
      if (a[j]>a[j+1])
       {
         int t = a[j];
         a[j]=a[j+1];
         a[j+1]=t;
         F=1;
       }
     }
    if (F==0)
       break;
   }
}
```