DAA

Tutorial - 3

Name: Karan Maurya

C. Roll no: 11

section: F

Ans. 1:

```
int binarySearch (int a[], int l, int b, int k)
{
    if ( l > b)
        return -1;
    else
    {
        int mid = l + (b-l)/2;
        if (a[mid] > k)
        {
            return binarySearch (a, l, mid-1, k);
        }
        else if (a[mid] < k)
        {
            return binarySearch (a, mid+1, b, k);
        }
        else
            return mid;
    }
}
```

$\boxed{T.C = O(\log n)}$

Ans. 2:

Iterative insertion sort:

```
void insSort( int a[], int n)
{
    for( int i=1; i<n; i++)
    {
        int k = a[i];
        int j = i-1;
        while( j>=0 && a[j] > k)
        {  a[j+1] = a[j]; j--; }
```

```
    }
    a[j+1] = K ~~a[j]~~;
  }
}
```

Recursive insertion Sort:

```
void insSort (int a[], int n)
{
    if (n<=1)
        return;
    insSort (a, n-1);
    int last = a[n-1];

    j = n-2

    while ( j>=0 && a[j] > last)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = last;
}
```

~~Ans 3~~

→ As in insertion sort the element is sorted at the the time of insertion, so it is called online sorting.

Ans. 3.

bubble sort	:	$O(n^2)$	$O(1)$
Insertion sort	:	$O(n^2)$	$O(1)$
sel sort	:	$O(n^2)$	$O(1)$
Merge sort	:	$O(n \log n)$	$O(n)$
Quick sort	:	best: $O(n \log n)$ Worst: $O(n^2)$	$O(1)$
Counting Sort	:	$O(n)$	$O(n+m)$
Heap Sort	:	$O(n \log n)$	$O(1)$

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

```
int binarySearch (int a[], int l, int h, int k)
{
    int m = (~~h~~ + h)/2;
    if(a[m] == key)
        return ~~true~~ m;
    else if( a[m] > k)
        h = mid-1;
    else
        l = mid +1;
}
```

O(log n) — Time Complexity

O(1) — Space Complexity

Recursive

T.C = O(log n)
S.C = O(1)

```
int binarySearch (int a[], int l, int h, int k)
{
    if(l > h)
        return -1;
    else
    {
        ~~if(a[~~ m = l + (h-l)/2;
        if(a[m] > k)
            return binarySearch (a, l, m-1, k);
        else if(a[m] < k)
            return binarySearch (a, m+1, h, k);
        else
            return m;
    }
}
```

Ans. 6

$$\boxed{T(n) = 2T(n-1) + 1}$$

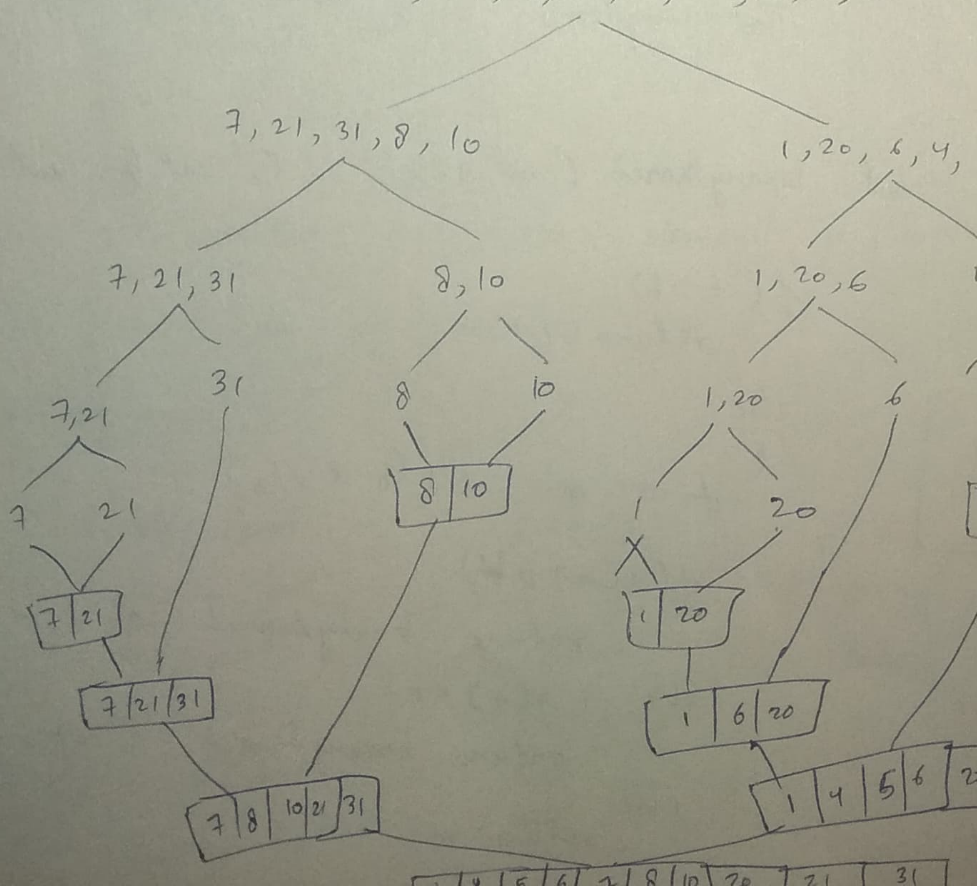
Ans. 7

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

Ans. 8 Quick Sort is the fastest generating purpose sort. In most practical situation, quicksort is the method of choice. If stability is important & space is available, mergesort might be best.

Ans. 9

7, 21, 31, 8, 10, 1, 20, 6, 4, 5



inversion: 3

Ans. 10 **Worst Case**: The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happen when input array is sorted or reverse sorted & either first or last element is picked as pivot $O(n^2)$.

Best case: It occur when Pivot element is the middle element or near to 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 parted into just 2 halves.
Works well on	Smaller array	fine on any size of array.
Additional space	$O(1)$	$O(n)$
Efficient	Inefficient for larger array.	more efficient.
Sorting Method	Internal	External
Stability	Not stable	Stable.

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;
}
```

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

-> External sorting : All the data is stored outside memory & only loaded into memory in small chunks.