Tutorial - 3

```
1. int linearSearch( int *arr, int n, int key)
     for i=0 to n-1
         if (arr[i] == key)
               return i.
     return -1
```

```
2. Iterative insertion sort
     void insertionSort ( int arr[], int n )
     {
          int i, j, temp;
          for( i=1 , to n
            {
              temp = arr[i]
              j = i-1
          while  j >= 0     AND  arr[j] > temp
          {
               arr[j+1] = arr[j];
                 j=j-1;
             }
           arr[j+1]= temp.
        } }
```

```
Recursive insertion sort
  void insertion sort( int arr[], int n)
  { if(n <= 1)
       return
    insertionsort (arr, n-1)
      int temp = arr[n-1]
           i = n-2
    while( i >=0  AND  arr[i] > temp)
```

```
            arr[i+1] = arr[i]
                i = i-1
        }
    arr[i+1] = temp;
    }
```

2. Insertion sort considers one input element per iteration and produces partial solⁿ without hindering future. Thus insertion sort is called inline sorting.

3. Time Complexities :-

	Bestcase
Bubble sort	$O(n^2)$
Selection Sort	$O(n^2)$
Insertion Sort	$O(n)$
Merge Sort	$O(n\log n)$
Quick Sort	$O(n\log n)$
Heap Sort	$O(n\log n)$

4)

	Inplace	Stable	Online
Bubble sort	✓	✓	
Selection sort	X	X	X
Insertion sort	✓	✓	✓
Merge sort	✓	✓	
Quick sort	X	X	
Heap sort	X	✓	

Date.
Page No.

5) Iterative Binary Search

```
int BinarySearch(int arr[], int l, int r, int key)
{
    while(l<=r)
    {   int  mid = (l+r) /2
        if (arr[mid] == key)
            return mid;
        if (arr [mid] < key)
            l= mid + 1;
        else
            r= mid - 1;
    }
    return -1
}
```

Time Complexity
Best = O(1)
Average = O(log n)
Worst = O(log n)

Space Complexity = O(1)

Recursive Binary Search

```
int Binary Search (int arr[], int l, int r, int key)
{
    if (l≤r) {  int mid = (l+r)/2
            if (arr[mid] == key)
                return mid;
        else if (arr [mid] <key) return Binarysearch(arr, mid+1, r, key)
        else
          return Binarysearch (arr, l, mid-1, key)
    }
    return -1;
}
```

6. Recurrence relation for binary search

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

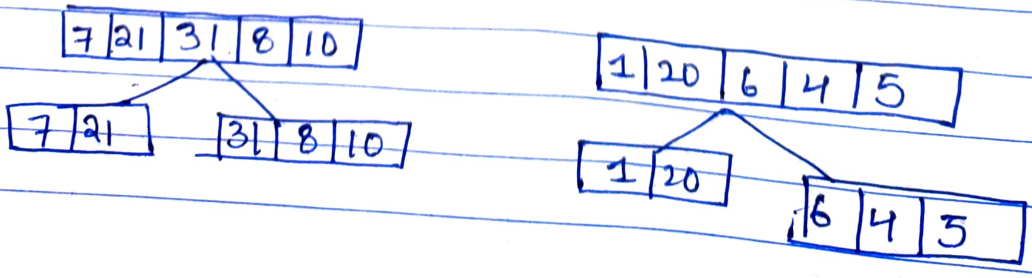
7.
```
void FindPairs(int arr[], int n, int k)
{
      unordered_set<int> s;
   for(int i=0; i<n; i++)
    {
        int t = ~~sum~~ k- arr[i];
        if(s.find(t) != s.end())
          cout<< arr[i] << t;
       s.insert(arr[i]);
    }  }
```

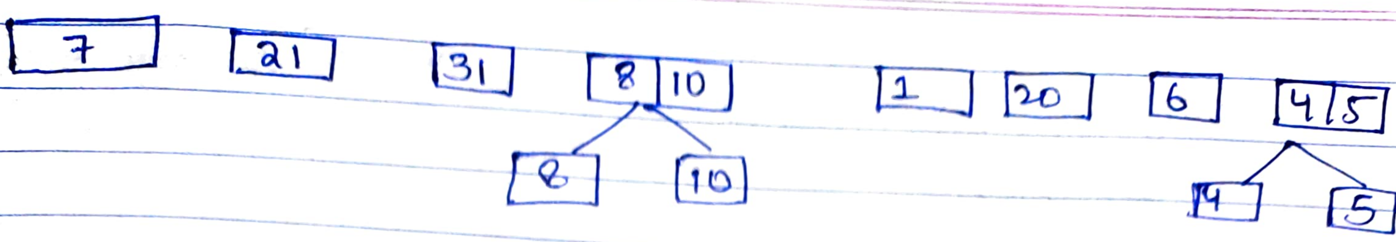
8. Quicksort is the fastest general purpose sort. In most practical situations, quick sort is the method of choice. If stability is important and space available merge sort might be best.

9. Inversion count for an array indicates how far or close the array is from being sorted. If the array is already sorted then inversion count is 0 and if array is reverse sorted, then inversion count is maximum.

Cond^n $a[i] > a[j]$ and $i<j$

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





Number of inversions = 31

10) Best case occurs when the array is completely random manner, time complexity $O(n\log n)$
Worst case occurs when array is already sorted either asending or descending order.
Time Complexity $O(n^2)$

11) Recurrence relation merge sort in Best and Worst case
$= 2T\left(\frac{n}{2}\right)+n$

Recurrence relation of Quick sort in
Best case $= 2T\left(\frac{n}{2}\right)+n$

Worst case $= T(n-1)+n$

Similarity -
Best case time complexity $= n\log n$
Difference
Worst case Time Complexity of merge sort $O(n\log n)$ and of quick sort $O(n^2)$

Merge sort array is parted into 2 halves and it operates fine on any size of array whereas in quicksort the splitting of an array of elements is in any ratio, not necessary divided into half. it works well on smaller array.

12) Selection sort works by finding the minimum element and then inserting it in its correct position by swapping with the element which is in the position of this minimum elements. This is what makes it unstable.

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

13) If our computer has a RAM of 2GB and we are given an array of 4GB for sorting, then we divide our source file into temporary files of size equal to the size of RAM and first sort these files by merge sort or quick sort.

Internal sorting - If the input data is such that it can be adjusted in the main memory at once.

External sorting - If the input data is such that it cannot be adjusted in memory entirely at once, it needs to be stored in the Harddisk, floppy disk or any other storage device.