)AA

Tutorial - 3

Q1-> Pseudo code for linear Search.

```
for (i=0 to n)
{
    if (arr[i] == key)
        Print "element found";
}
```

Q2-> recursive

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

iterative

```
for (i=2 to n)
    key = A[i]
    j = i-2
```

```
While (j >= 0 and A[j] > key)
{
    A[j+1] = A[j]
    j = j-1;
}

A[j+1] = key.
```

Insertion is online sorting because it insert know the whole input move input can be inserted with the insertion sorting is running

Q3→ Complexity of different Sorting algorithms.

Name	Best case	worst case	Average
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

4→ Inplace Sorting Stable Sorting online Sorting.

Inplace	Stable	online
Bubble	Merge	Insertion
Selection	Bubble	
Insertion	Insertion	
Quick Sort		
Heap		

05→ Resursive →

```
int Binary (int arr[], int l, int r,
            int x)
  if (r >= l)
      int mid = l + (r-l)/2;
    if (arr[mid] == x)
      return mid;
   else if (arr[mid] > x)
    return binary (arr, l, m-1, x);
  else
      return binary (arr, m+1, r, x);
  }
   return -1;
}
```

itrative :-

```
int binary (int arr[], int l, int r,
            int x)
{
    while (l <= r)
    {
        int m = l + (r-l) / 2;
        if (arr[m] == x)
            return m;
        else if (arr[m] > x)
            r = m - 1;
        else
            l = m + 1;
    }
    return -1;
}
```

Time complexity :

Binary search -> O(logn)

Linear search -> O(n)

Q6→ Recurrence relation for binary resursive search.

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

$T(n)$ = time required for binary search in an array of n.

Q8→ Which sorting is best for practical uses? Explain

Quick sort is fastest general purpose sort. If Satisfy is important and space is available the merge sort can be best Sorting.

Q inversion in an array

The inversion count for any array is the number of steps it will take for the array to be sorted, or how far away any array is from being sorted. If we are given an array sorted in reverse order, the inversion count will be the maximum number in that array.

```
#include <iostream>
using namespace std;

int getInvCount(int arr[], int n)
{
    int inv_count = 0;
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (arr[i] > arr[j])
                inv_count++;

    return inv_count;
}
int main()
{
    int arr[] = {7, 21, 31, 8, 10, 1, 20,
                 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Number of inversions are "
         << getInvCount(arr, n);
    return 0;
}
```

Ans 10 ⇒ The worst case time complexity of quick sort is $O(n^2)$. This occurs when the picked pivot element is an extreme (smallest or largest) element. This happens when inputs array is sort or reverse sorted.

The best case of quick sort is when pivot element is mean element of that array.

Q 11 ⇒ Merge sort is more efficient and works faster than quick sort in case of larger array.

Worst case: complexity of quick sort is $O(n^2)$ and for merge sort is $O(n \log n)$.

Q 12 ⇒ Stable Selection Sort

```
Void stable selection (int arr[], int n)
{
    for (int i=0 ; i<n-1 ; i++)
    {
        int min = i;
```

```
for (int j = i+1 ; j < n ; j++)
{
  if (arr[min] > arr[j]
      min = j;
  }
int key = arr[min];
while (min > i)
{
  arr[min] = arr[min-1];
    min--;
  }
  arr[i] = key;
}
}
```

13) Modified Bubble Sort

```
void Bubble (int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int swaps = 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;
                swaps ++;
            }
        }
        if (swaps == 0)

            break;
    }
}
```