Name - Ayush Chauhan
Sec - B
Roll No. - 37
Date : ___/___/___

Tutorial-3

1.
```
for (i=0 to n)
{ if (a[i]==value)
      // element from found
}
```

2. Iteration:
```
void insertionsort(int a[], int n)
{ 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;
  }
}
```

Recursion:
```
void insertionsort (int a[], int n)
{   if (n<=1) return;
    insertionsort (a, n-1);
    int last=a[n-1];
     int j=n-2;
   while, (j>=0 && a[j]>last)
    {  a[j+1]=a[j];
       j--;
    }
   a[j+1]=last;
}
```

3. Sorting Algorithms Best Worst Average

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

4.

Inplace	Stable	Online
Bubble Sort	Merge Sort	Insertion Sort
Selection Sort	Bubble Sort	
Insertion Sort	Insertion Sort	
Quick Sort	Count Sort	
Heap Sort		

5. Iterative;

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

TC: Linear Srch; $O(n)$
Binary Srch; $O(\log n)$

Date : ___/___/___

6. $T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/4) + 1$ — (2)

$T(n/4) = T(n/8) + 1$ — (3)

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

$= T(n/4) + 1 + 1$

$= T(n/8) + 1 + 1 + 1$

$\vdots$

$T(n/2^k) + 1$ (k times)

Let $2^k = n$

$k = \log n$

$T(n) = T(n/n) + \log n$

$T(n) = T(1) + \log n$

$T(n) = O(\log n)$

7.
```
for (i=0; i<n; i++)
{ for (int j=0; j<n; j++)
    { if (a[i]+a[j]==k)
        printf ("%d %d", i, j);
    }
}
```

8. Quick sort is fastest general purpose sort. In most practical situation, quicksort is the method of choice as stability is important and space is available, mergesort might be best.

9. A pair $(A[i], A[j])$ is said to be inversion of $A[i] > A[j]$, $i < j$

Total no. of inversions in given array are 31.

(UPCP)

10. Worst case $O(n^2)$: It occurs when the pivot element is an extreme element. This happens when input array is sorted or reverse sorted and either first or last element is selected as pivot.

Best Case $O(n \log n)$: The best case occurs when we will select pivot element as a mean element.

11. Merge Sort:
Best Case – $T(n) = 2T(n/2) + O(n)$
Worst Case – $T(n) = 2T(n/2) + O(n)$

Quick Sort:
Best Case – $T(n) = 2T(n/2) + O(n)$
Worst Case – $T(n) = T(n-1) + O(n)$

12.
```
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 - j];
        min - - j }
    a[i] = key;
}
```