NAME - VARSHA TYAGI , ROLL NO - 11 , CE

DAA assignment - 3

Ques1- Pseudo Code for Linear Search

```
for ( i = 0 to n )
  {
      if ( arr [i] = = Value )
        // element found
  }
```

Ques2. Void recursive Insertion (int arr [ ] , int n)

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

for i = 1 to n
{
    key ← A[i]
    j ← i-1
  while ( j ≥ 0 and A[i] > key ]
  {
     A [j+1] ← A [j]
     j ← j-1
  }
```

```
    A[j+1] ← key
}
```

Ques3- Complexity of all sorting Algorithm -

		Best	Worst	Average
(a.)	Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
(b.)	Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
(c.)	Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
(d.)	Heap Sort	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
(e.)	Quick Sort	$O(n\log(n))$	$O(n^2)$	$O(n\log(n))$
(f)	Merge Sort	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$

Ques4-

Implace Sorting	Stable Sorting	Online Sorting
Bubble	Merge Sort	Insertion.
Selection	Bubble	
Insertion	Insertion	
Quick Sort	Count	
Heap Sort		

Ques5- Recursion Binary Search

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

Itterative

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

The time Complixity recursive => O(logn)

Binary Search -

Linear Search => O(n)

Q6- Recurrence relation for binary search

$T(n) = T(n/2) + 1$ — (I)

$T(n/2) = T(n/4) + 1$ — (II)

$T(n/4) = T(n/8) + 1$ — (III)

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

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

$\vdots$

$= T(n/2k) + 1 \ (k \text{ times})$

Let $2^k = n$

$k = \log n$

$\therefore T(n) = T(n/2) + \log n$

$T(n) = T[1] + \log n = o(\log n)$

Ques8- Quick Sort is the fastest general purpose sort in most practical situations, ~~ge~~ is the method of choice. if stability is important and space is available, merge sort might be best.

Ques9- A pair $(a[i], a[j])$ is said to be inversion if $a[i] > a[j]$

arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

Total number of inversion are 31 using merge sort.

Ques10- Worst case in Quick Sort :-

Ans - The worst case time complexity of a Quick Sort is $O(n^2)$ if the picked pivot element is always an extreme (smallest or largest element)

Or the given array is sorted and we pick either first or last element.

Best case in Quick Sort :-

The best case is $O(n \log(n))$ when we will select pivot element as & mean element.

Ques11

Ans - Quick Sort -

Worst case

$$T(0) = T(1) = 0 \text{ (base)}$$

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

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

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

$$T(n-2) = (n-2) + T(n-3)$$

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

$$T(n) = n + n-1 + n-2 + T(n-3)$$

$$T(n) = n(k \text{ times}) - (k) + T(n-k)$$

Let $k = n$

$$\therefore \quad T(n) = n \times n + n + T(n-n)$$

$$= n^2 + n + T(0)$$

$$\therefore \quad T(n) = O(n^2)$$

Best Case -

$T(0) = T(1) = 0$ (Base)

$T(n) = 2T(n/2) + n$ — ①

$T(n/2) = 2T(n/4) + \frac{n}{2}$ — ②

$T(n/4) = 2T(n/8) + \frac{n}{4}$ — ③

$\therefore T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$

$T(n) = 2\left[2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right] + w$

$= 4T\left[\frac{n}{2^3}\right] + n + n + w$

$\vdots$

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

Let $2^k = n$

$k = \log n$

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

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

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

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

	Quick Sort	Merge Sort
1.	Splitting of a array of elements is in any ratio, not necessary divided into half.	In the Merge Sort an Array is parted into just two halfes.
2.	Worst complexity O(n)	O(n.logn)
3.	It works will on small array.	It operates fine on any size of array.
4.	It works faster than other Sorting algo for small data eg- selection Sort.	It has consistent speed on any size of Data.
5.	Internal Sorting method.	Internal Sorting Method.

Ques12 Ans- Stable Selection Sort -

```
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-i];
      min--;
  }
      a[i] = key;
}
```

Ques 13
Ans – A better version of bubble sort, known as modified bubble sort, includes a flag that is set if an exchange is made after an entire pass ones the array. It no exchange is made, then it should be class the array is already order because no two elements need to be switched in that case the end.

```
Void bubble (int a[], int n)
  {
      for (int i=0; i<n; i++)
      {     int swaps =0;
      for(int j=0; j<n-i-1; j++)
       {     if (a[j]>a[j+1])
        {
              int t = a[j];
              a[j] = a[j+1];
              a[j+1] = t;
              swap++;
        }
      }
      if (swaps = = 0)
            break;
     }
}
```