Name- Abhishek Singh

Section- CST-SPL-2

class Rollno.- 09

University rollno- 2017447

Ans.1) Pseudocode for linear search

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

Ans.2)

```
void insertion(int arr[], int n)
{
    if (n<=1)
        return;
    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[j]> key)
        {
            A[j+1] ← A[j]
            j ← j-1
        }
        A[j+1] ← key
}
```

Insertion sort is online sorting because it doesn't know the whole input, more input can be inserted with the insertion sorting is running

Ans.3> Complexity->

Selection sorting ->

Best -> $O(n^2)$

Worst -> $O(n^2)$

Average -> $O(n^2)$

Bubble sort->

Best -> $O(n)$

worst -> $O(n^2)$

Average -> $O(n^2)$

	Best	Worst	Average
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 →	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

Ans. 4)

Inplace Sorting	stable sorting	Online sorting
Bubble	Merge sort	Insertion
Selection	Bubble sort	
Insertion	Insertion	
Quick sort	Count sort	
Heap sort		

Ans. 5)

```
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;
}

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 of

linear search => $O(n)$

Binary search => $O(\log n)$

Ans. 6> $T(n) = T(n/2)+1$

where $T(n)$ is the time required for binary search in an array of size 'n'

Ans. 7>

```
int find (A[], n, k)
{
    sort (A, n)
    for (i = 0 to n-1)
    {
        n = binary search (A, 0, n-1, k-A[i])
        if (n)
            return 1
    }
    return -1
}
```

Time complexity = $O(n \log n) + n \cdot O(\log n)$

$= O(n \log n)$

Ans. 8> - Quick sort is the fastest ~~sort~~ general purpose sort

- In most practical ~~sited~~ situation, quick sort is the method of choice. If stability is important and space is available, merge sort might be best

Ans. 9> A pair (a[i], a[j]) is said to be inversion if a[i] > a[j]

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

total no. of inversion are 31, using merge sort.

Scanned with CamScanner

Ans. 10> The worst case time complexity of quick sort is $O(n^2)$.

This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted.

- The best case of quick sort is when we will select pivot as a mean element.

Ans. 11> Reccurrence relation of

Merge sort $\rightarrow T(n) = 2T(n/2) + n$

Quick sort $\rightarrow T(n) = 2T(n/2) + n$

- Merge sort is more efficient and works faster than quick sort in case of larger array size or data set.

- Worst case complexity for quick sort is $O(n^2)$ whereas $O(n \log n)$ for merge sort.

Ans. 12) Stable selection sort $\rightarrow$

```
void stableselection (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;
}

}
```

Ans. 13) Modify 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;
  }
}
```

Scanned with CamScanner