Name → Aman Dwivedi
Section → C.S.T
Roll No → 07

classmate
Date
Page

Tutorial - 3

1- Pseudo Code for Linear Search.

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

2-

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

Iterative:

```
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+2] ← key
}
```

3- Complexity of all sorting Algorithm..

	Best	Worst	Average
→ Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble "	$O(n)$	$O(n^2)$	$O(n^2)$
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.
Bubble	Merge	Insertion
Selection	Bubble	
Insertion	Insertion	
Quick	Count	
Heap		

5. Recursive Binary Search

```
int binarysearch (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 binarysearch(arr, l, mid-1, x);
      return binarysearch(arr, mid+1, r, x);
   }
   return -1 ;
}
```

Iterative:

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

Time complexity recursive: $O(\log n)$
Binary Search

Linear Search: $O(n)$

6- Recursive Relation for Binary Search.

$T(n) = T(n/2) + 1$ - ①

$T(n/2) = T(n/4) + 1$ - ②

$T(n/4) = T(n/8) + 1$ - ③

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

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

$\Rightarrow T(n/2^k) + 2$ (k times)

Let $2^k = n$

$k = \log n$

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

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

8- Quick sort is the ~~fastest~~ & most efficient sorting algorithm which makes ~~it to~~ makes it most used as well. It is faster as compared to other sorting algorithm. Also, its time complexity is $O(n \log n)$. But in case of a larger array, Merge sort is preffered.

9- Inversion in an array basically defines how far or close an array is from being sorted.
If array is already sorted, Inversion count → 0,
if array is in reverse order, inversion count → max.

| 7 | 21 | | 31 | 8 | | 1 | 10 | | 20 | 6 | | 4 | 5 |

$31 > 8$ $\quad$ $10 > 1$ $\quad$ $20 > 6$ $\quad$ $c = 3$

| 7 | 8 | 21 | 31 | | 1 | 6 | 10 | 20 | | 4 | 5 |

$21 > 8$ $\quad$ $10 > 6$ $\quad$ $c = 5$

Inv.

| 1 | 6 | 7 | 8 | 10 | 20 | 21 | 31 | | 4 | 5 |

| 1 | 4 | 5 | 6 | 7 | 8 | 10 | 20 | 21 | 31 |

Count → 14

so $14 + 17 = 31$

Total inversion = 31

10- Best Case: If pivot element is in the middle.
Time complexity = $O(n \log n)$

Worst Case: If pivot element is at extensive position and array is reverse sorted.
Time complexity = $O(n^2)$.

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

Merge Sort: $T(n) = 2T(n/2) + n$

* In Merge sort, the array is divided into two equal halves
T. C = $O(n \log n)$

* In quick sort, the array is divided into any ratio depending on position of pivot element.
T. C range $O(n^2) - O(n \log n)$.

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

13- void bubble(int a[], int n)

```
{
  for (i=0 to n)
  {
     flag = 0;
  for (j=0 to n-1-i)
   if (a[j] > a[j+1])
      swap (a[j], a[j+1]);
         flag = 1;
    if (flag == 0)
      break;
    }
 }
```

14- In that case, external sorting algorithm such as k-way merge sort is used that can handle large data amount and sort it which can't fit into main memory.

A part of array resides in RAM during the execution whereas in internal sorting process takes place ~~entity~~ entirely within the main memory. Ex- Bubble, Selection, etc.