Name - Riya Chaudhary
Section - I
Roll No - 7

Date
Page No. 1

Tutorial - 3

1. int linearSearch (int *arr, int n, int key)
for i <= 0 to n-1
if arr[i] = key
return i.
return -1

2. Insertion Sort
Iterative
for j = 2 to n
key = a[j]
i = j-1
while i > 0 and a[i] > key
a[i+1] = a[i]
i = i-1
a[i+1] = key

Recursive
if n > 1
then insertion(a, n-1)
key = a[n]
i = n-1
while i > 0 and a[i] > key
a[i+1] = a[i]
i = i-1
a[i+1] = key

It is online sorting because an online algorithm does not know the whole input it might take decisions that later turn out not to be optimal. It produces optimum result.

other sorting like Bubble sort, Selection Sort, merge sort, Heap Sort, Heap sort and Counting Sort.

3. Complexities of sort algorithms are -

1. Bubble Sort -
Best Case - $O(n)$
Worst Case - $O(n^2)$
Average Case - $O(n^2)$

2. Selection Sort -
Best Case - $O(n^2)$
Worst Case - $O(n^2)$
Average Case - $O(n^2)$

3. Insertion Sort -
Best Case - $O(n)$
Worst Case - $O(n^2)$
Average Case - $O(n^2)$

4. Merge Sort -
Best Case - $O(n \log n)$
Worst Case - $O(n \log n)$
Average Case - $O(n \log n)$

5. Quick Sort -
Best Case - $O(n \log n)$
Worst Case - $O(n^2)$
Average Case - $O(n \log n)$

6. Counting Sort -
Best Case - $O(n+K)$
Average Case - $O(n+K)$
Worst Case - $O(n+K)$

7. Heap Sort -
Best Case - $O(n \log n)$
Average Case - $O(n \log n)$
Worst Case - $O(n \log n)$

5. Iterative Pseudo Code -
Binary Search

```
Int binarySearch (int a[], int l, int r, int x)
{
  while (l <= r)
  {
```

```
int m ← (l+u)/2
if (a[m] = x)
  return m
if (a[m] < x)
   l ← m+1
else
   u ← m-1
}
return -1
}
```

Time Complexity -
Best case - O(1)
Worst case - O(log n)
Average case - O(log n)

Space complexity -
O(1)

Recursive -

```
int binarySearch (int a[], int l, int u, int n)
{ if (u >= l)
  { mid ← (l+u)/2
    if (a[m] = x)
      return m;
    else if (a[m] > x)
      return binarySearch (a, l, m-1, x)
    else
      return binarySearch (a, m+1, u, x)
  }return -1
}
```

Time Complexity -
BC - O(1)
WC - O(log n)
AC - O(log n)

Space Complexity -
BC - O(1)
WC -
AC -

Complexity of Linear Search-

Iterative Time Complexity — Space complexity - $O(1)$

Best case - $O(1)$

WC - $O(n)$

AC - $O(n)$

Recursive — Space Complexity - $O(1)$

BC - $O(1)$

WC - $O(n)$

AC - $O(n)$

6. $T(n) = T(\frac{n}{2}) + 1$

Put $n = \frac{n}{2}$

$T(\frac{n}{2}) = T(\frac{n}{4}) + 1$

$T(n) = T(\frac{n}{4}) + 1 + 1$

$T(\frac{n}{4}) = T(\frac{n}{8}) + 1$

$T(n) = T(\frac{n}{8}) + 1 + 1 + 1$

$T(n) = T(\frac{n}{2^k}) + k$

$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = \log n$

$T(n) = T(\frac{n}{2^{\log_2 n}}) + \log n$

$T(n) = T(\frac{n}{n}) + \log n$

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

$\boxed{T(n) = O(\log n)}$

8. Quick Sort is mostly used in practical used because it is the fastest general purpose sort. In most practical situations, quicksort is the method of choice. If stability is important and space is available mergesort might be best.

19. The worst case time complexity of a typical implementation of QuickSort is $O(n^2)$. The worst case occurs when the picked pivot is always an extreme element. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot and it gives the best case complexity when we will select pivot as a mean element it gives $O(n \log n)$ complexity.

12. Yes, we can write a version of stable selection sort like we if instead of swapping, the minimum element is placed in its position without swapping i.e., by placing the number in its position by pushing every element one step forward. It can be stable by changing key comparison operation so that comparison of two keys considers position as a factor with equal key means doesn't changes & it becomes stable as well.

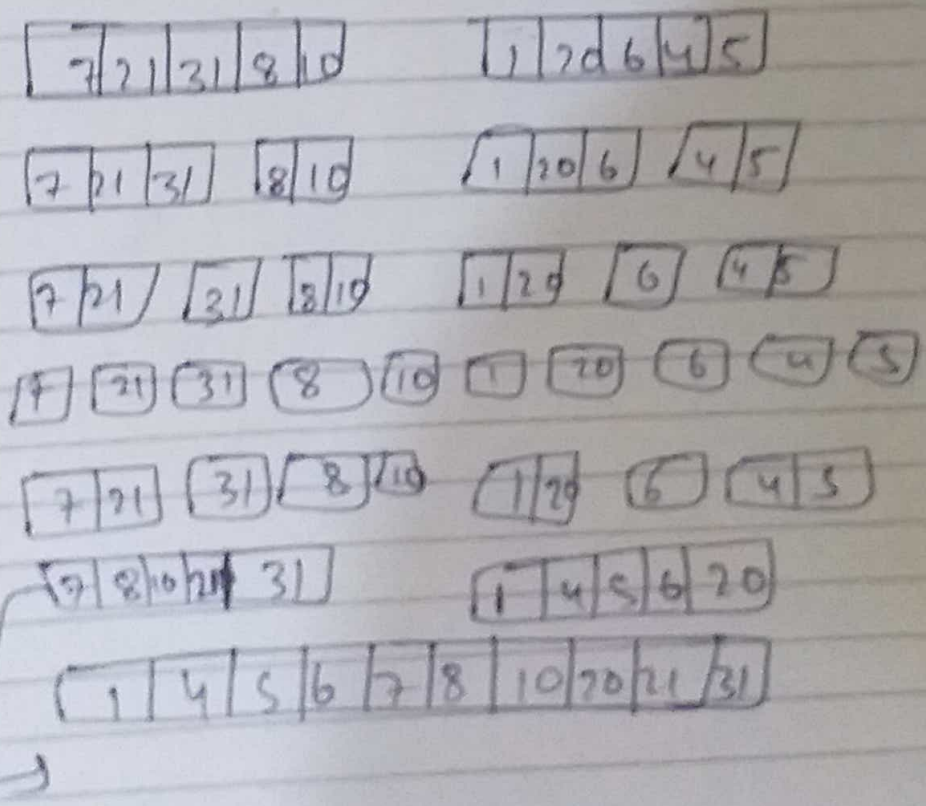
4. Inplace Sorting - Bubble Sort, Selection Sort, Insertion Sort, Heap Sort, Quick Sort

Online Sorting - Selection Sort, Insertion Sort

Stable Sorting - Bubble Sort, Insertion Sort, Merge Sort, Counting Sort.

9. Inversion Count for an array indicates how far the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if array is sorted in reverse order, inversion count is the maximum. Two elements a[i] & a[j] form an inversion if a[i] > a[j] & i < j.

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



7	8	10	21	31

1	4	5	6	20

(7,1), (8,1), (10,1), (21,1), (31,1), (7,4), (8,4), (10,4), (21,4), (31,4), (7,5), (8,5), (10,5), (~~21,5~~), (~~10,20~~), (31,5), (7,6), (8,6), (10,6), (21,6), (31,6), (21,20), (31,20)

Count Inversion = 22

11. Best case of Quick sort

$T(n) = 2T\left(\frac{n}{2}\right) + n$

$a = 2 \quad f(n) = n$

$b = 2$

$= n^{\log_b a} = n$

$= n^{\log_2 2} = n$

$f(n) = n$

$f(n) = \Theta(n \log_b a \log^k n)$

$f(n) = \Theta(n \log^k n)$

$\boxed{f(n) = \Theta(n \log n)}$

Worst case of Quick Sort

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

Put $n = n-1$

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

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

Put $n = n-2$

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

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

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

$T(k) = T(n-k)$

& $n = k+1$

$k = n-1$

$T(n) = 1+2+3+\cdots n$

$T(n) = \frac{n(n+1)}{2} = O(n^2)$

$T(n) = O(n^2)$

Best case of Merge Sort -

$T(n) = 2T(\frac{n}{2}) + n$

$a = 2, b = 2$ $f(n) = n$

$n^{\log_b a} = n^{\log_2 2} = n$

$f(n) = \Theta(n \log_b a \log^k n)$

$f(n) = \Theta(n \log^k n)$

$f(n) = \Theta(n \log n)$

Same as worst case.

Similarities is Best case complexities of merge Sort & quick Sort both are same $T(n) = \Theta(n \log n)$ But worst case complexities of both the sorting are different merge Sort $= \Theta(n \log n)$ & quick Sort $= O(n^2)$