DAA

Tutorial → 3

1-

```
search (arr, n, x)
        if arr[n-1] == x
            return "True"
        temp = arr[n-1].
        arr[n-1] = x

        for i=0, i++ ...
            if arr[i]== x
                // —
            arr[n-1] = temp
            return (i<n-1)
```

=> The idea is to copy x (element to be searched) to last location so that one last comparision when x is not present in arr[] is saved.

2 - Iterative Insertion Sort-

```
InsertSort(arr, n)
        for (i=1, i<n, i++)
            value = arr[i]
              j = i
            while j>0 && arr[j-1] > value
                arr[j] = arr[j-1]
                j --
            arr[j] = value
```

Recursive

```
InsertSort(arr[], i, n)
    value = arr[i]
      j = i
    while(j>0 && arr[j-1]>value)
        arr[j] = arr[j-1]
        j--
    arr[j] = value
  if (i+1<=n)
      InsertSort (arr. i+1, n)
```

⇒ Insertion sort, by contrast, is online because it does not need to Know anything about what values it will sort & the information is requested WHILE the Algorithm is running. Simply put, it can grab new values at every iteration

3-

Sorting Algo	T. C.			S.C
	B.C	A.C	W.C	W.C
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$

4- 1. Insertion sort and Quick sort are inplace sort.

2- Insertion, merge and bubble sorts are stable.

3- Selection and insertion sort are online sort.

5- **Recursive Binary Search**

```
BS(arr[], l, r, x)
    if r>=l
        mid = l+(r-l)/2
    if arr[mid] == x
        return mid
```

```
if (arr[mid] > x)
    return BS(arr, l, mid -1, x)
return BS(arr, mid +1, r, x)
```

Iterative

```
BS(arr[], l, r, 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
```

Linear Search

T.C S.C

1) Recursive → O(n) , O(n)

2) Iterative → O(n) ; O(1)

Binary Search

T.C S.C

1) Recursive → O(logn) , O(logn)

2) Iterative → O(logn) . O(1)

7-

```
void findsum(int a[], int n), int K)
{
        int i=0, j=n-1;
        int sum=0;
        while(i<j)
        {
              if ( a[i] + a[j] == K)
              {    cout<<i<<j<<endl;
                   return;
              }
              if ( a[i] + sum < K )
                   sum+= a[i++];
              if ( a[j] + sum < K )
                   sum+= a[j--];
        }
}
```

8- In most practical situations, Quicksort is the fastest general-purpose sort.
If stability is important and space is available, mergesort might be best. In some performance-critical applications, the focus may be on just sorting numbers, so it is reasonable to avoid the costs of a using references and sort primitive types instead.

9- ~~Indic~~ Inversion count for an arry indicates how far (or close) the array is from being sorted. If the array is already sorted, then inversion count is 0, but if the array is sorted in the reverse order, the ~~re~~ inversion count is maximum.

$\boxed{a[i] > a[j] \text{ but } i<j}$

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

=> inversions using merge sort => 31

10- Best Case occurs when partition process always picks the middle element as pivot.

$$\boxed{T(n) = 2T(n/2) + n}$$

Worst Case occurs when partition process always picks great

or smallest element as pivot

$$\boxed{T(n) = T(n-1) + n}$$

11- Merge Sort

Best → $T(n) \Rightarrow 2T(n/2) + \Theta(n)$

Worst → $T(n) \Rightarrow 2T(n/2) + \Theta(n)$

Quick sort

Best → $T(n) \Rightarrow 2T(n/2) + \Theta(n)$

Worst → $T(n) \Rightarrow T(n-1) + \Theta(n)$

12- Quick Vs Merge Sort

1- merge sort, array is parted into 2 halves ($n/2$), and in Quick sort into any ratio.

2- Worst time Complexity of Quick Sort is $O(n^2)$ whereas as $O(n \log n)$ for Merge sort.

3- Quick sort works with small data whereas merge sort can work on any type of data.

4- Merge sort is efficient in case of large data & Quick sort is efficient in case of small data.

5- Merge sort is stable & ~~inplace~~ not inplace but Quick Sort is inplace but unstable.

6- Quick Sort is preferred for arrays & M.S for Linked Lists

12- Any comparision based sorting algorithm which is not stable by nature can be modified to be stable by changing the key comparision operation so that the comparison of two keys considers position as a factor for objects with equal key or by tweaking it in a ways such that its meaning doesn't change and it becomes stable as well.

```
void stableSelectionSort (int a[], int n)
{
      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-1];
              min--;
          }
          a[i] = Key;
      }
}
```

13-

```
void OptimizedBubbleSort (int a[], int n)
{
      int i,j;
      bool swapped;
      for (i=0; i<n-1; i++)
      {
          swapped = false;
          for (j=0; j<n-i-1; j++)
          {   if (a[j] > a[j+1])
                 {   swap(&a[j], &a[j+1]);
                 }   swapped = true;
          }
```

```
  {
     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-1];
          min--;
      }
      a[i] = Key;
   }
}
```

13-

```
void OptimizedBubbleSort(int a[], int n)
{
    int i,j;
    bool swapped;
    for(i=0; i<n-1; i++)
    {
        swapped = false;
        for(j=0;j<n-i-1; j++)
        {   if(a[j]>a[j+1])
             {  swap(&a[j], &a[j+1]);
                swapped = true;
        }    }

    if(swapped == false)
        break;
    }
}
```

14- We will divide 4GB file into 2GB chunk of RAM Size usin
external sorting. Then sort these temporary files one bye on
using the ram individually (any sorting algorithm : Quick/m

→ If the input data is such that it can be adjusted in the main memory at once, it is called internal sorting.

→ If the input data is such that it cannot be adjusted in the
memory entirely at once, it needs to be stored in a hard
floppy disk, or any other storage device is called External Sor