

## Tutorial-3

Name - Deepak Kohli

Section - F

Roll no. - 53

Q-1 Write linear search Pseudo code to search an element in a sorted array with minimum comparisons.

Ans-1

```

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

```

Q-2 Write Pseudo Code for iterative and recursive insertion sort. Insertion sort is called Online sorting. Why? What about other sorting algorithms that has been discussed?

Ans-2 Iterative

```

void insertion-sort (int arr[], int n)
{
    for (int i = 1; i < n; i++)
    {
        j = i - 1;
        x = arr[i];
        while (j > -1 && arr[j] > x)
        {
            arr[j+1] = arr[j];
        }
    }
}

```

```

    }
    arr[j+1] = x;
}
}

```

Recursion:-

```

void insertion_sort(int arr[], int n)
{
    if (n <= 1)
        return;
    insertion_sort(arr, n-1);
    int last = arr[n-1];
    int j = n-2;
    while (j >= 0 && arr[j] > last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last;
}

```

Insertion sort is called 'Online Sort' because it does not need to know anything about what values it will sort and information is requested while algorithm is running.

## Other Sorting Algorithms :-

- ) Bubble sort
- ) Quick sort
- ) Merge sort
- ) Selection sort
- ) Heap sort

Q-3 Complexity of all sorting algorithm that has been discussed in lectures.

Ans-3	Sorting Algorithm	Best	Worst	Average
	Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
	Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
	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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q-4 Divide all sorting algorithms into inplace / stable / Online sorting.

Inplace sorting	Stable sorting	Online sorting
Bubble sort	Merge sort	Insertion sort
Selection sort	Bubble sort	
Insertion sort	Insertion sort	
Quick sort	Count sort	
Heap sort		



Q-5 Write recursive / iterative Pseudo Code for binary search. What is the Time and Space complexity of Linear and Binary Search.

Ans-5 Iterative -

```

int bsearch (int arr[], int l, int h, int key)
{
    while (l <= h) {
        int m = ((l + h) / 2);
        return m;
        else if (key < arr[m])
            h = m - 1;
        else
            l = m + 1;
    }
    return -1;
}

```

Recursive -

```

int bsearch (int arr[], int l, int h, int key)
{
    while (l <= h) {
        int m = ((l + h) / 2);
        if (key == arr[m])
            return m;
        else if (key < arr[m])
            return bsearch (arr, l, mid - 1, key);
        else
            return bsearch (arr, mid + 1, h, key);
    }
    return -1;
}

```

}

- \* Time Complexity :-
- Linear Search -  $O(n)$
- Binary Search -  $O(\log n)$

Q-6 Write recurrence relation for binary recursive search.

$$\text{Ans-6} \quad T(n) = T(n/2) + 1 \quad \text{--- (1)}$$

$$T(n/2) = T(n/4) + 1 \quad \text{--- (2)}$$

$$T(n/4) = T(n/8) + 1 \quad \text{--- (3)}$$

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 1 + 1 \\ &= T(n/8) + 1 + 1 + 1 \\ &\vdots \end{aligned}$$

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

$$\text{let } 2^k = n$$

$$k = \log n$$

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

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

$$T(n) = O(\log n) \quad \text{Ans.}$$

Q-7 Find two indexes such that  $A[i] + A[j] = k$  in minimum time complexity.

for (int i = 0; i < n; i++)

{  
for (int j = 0; j < n; j++)

{  
if (A[i] + A[j] == k)  
printf("0%d", i, j);

}

}



Q-8 Which sorting is best for practical uses? Explain.

Ans- Quick Sort is fastest general-purpose sort. In most practical situations quicksort is the method of choice as stability is important and space is available, mergesort might be best.

Q-9 What do you mean by inversions in an array? Count the number of inversions in Array  
arr[] = {7, 21, 31, 8, 16, 1, 20, 6, 4, 53}  
using merge sort.

Ans- A Pair  $(A[i], A[j])$  is said to be inversion of  $A[i] > A[j]$   
 $i < j$

Total no. of inversions in given array are 31 using merge sort.

Q-10 In which cases Quick Sort will give best and worst case time complexity.

Ans- Worst Case  $O(n^2)$  → The worst case occurs when the pivot element is an extreme (smallest/largest) element. This happens when input array is sorted or reverse sorted and either first or last element is selected as pivot.

Best case  $O(n \log n)$  - The best case occurs when we will select pivot element as a mean element.

Q-11 Write Recurrence Relation of Merge/Quick sort in best and worst case. What are the similarities and differences between complexities of two algorithm and why?

Ans- Merge sort

$$\text{Best Case - } T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$$

$$\text{Worst Case - } T(n) = 2T(n/2) + O(n)$$

Quick sort

$$\text{Best Case - } T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$$

$$\text{Worst Case - } T(n) = T(n-1) + O(n) \rightarrow O(n^2)$$

In quick sort, array of element is divided into 2 parts repeatedly until it is not possible to divide it further,

In merge sort the elements are split into 2 subarray  $(n/2)$  again and again until only one element is left.



Q-12 Selection sort is not stable by default but can you write a version of stable selection sort?

Ans-

```

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;
}

```

Q-13 Bubble sort scans array even when array is sorted. Can you modify the bubble sort so that it does not scan the whole array once it is sorted.

Ans- A better version of bubble sort, known as improved bubble sort, includes a flag that is set if an exchange is made after an entire pass over the array. If no exchange is made then it should be called that the array is already sorted.



because no two elements had to be switched.

```
void bubble (int arr[], int n)
```

```
{
```

```
    for (int i = 0; i < n; i++)
```

```
    {
```

```
        int swaps = 0;
```

```
        for (int j = 0; j < n - 1 - i; j++)
```

```
        {
```

```
            if (arr[j] > arr[j+1])
```

```
            {
```

```
                int t = arr[j];
```

```
                arr[j] = arr[j+1];
```

```
                arr[j+1] = t;
```

```
                swaps++;
```

```
            }
```

```
        }
```

```
        if (swaps == 0)
```

```
            break;
```

```
    }
```

```
}
```