Ques 1 → Write linear Search pseudoCode to search an element in a Sorted array with minimum Comparisons.

Ans →

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

}
```

Ques 2 → Write pseudo Code for iterative and recursive ~~sort~~ insertion sort. Insertion Sort is Called online sorting. Why? What about other Sorting algorithms that has been discussed in lectures?

Ans Iterative →

```
Void insertionsort (int A[], int n)
{
  for (int i=1; i<n; i++)
   {  j = i-1;
      x = A[i];
     While( j>-1 && A[j]>x)
       {   A[j+1] = A[j]
           j--;
       }
     A[j+1] = x;
   }
}
```

Recursive ⇒

Dheeraj

```
Void insertionSort (int arr[], int n)
{
    if (n <= 1)
        return;

    insertionSort (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 the information is requested WHILE the algorithm is running.

Other Sorting algorithm:-

- Bubble Sort
- Quick Sort
- Merge Sort
- Selection Sort
- Heap Sort

Dheeraj

Ques 3 ⇒ Complexity of all the Sorting algorithm that has been discussed in lectures.

Ans

	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	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

Ques 4 ⇒ Divide all the sorting algorithms into implace/stable/online sorting.

Ans ⇒

Implace Sorting	Stable Sorting	Online Sorting
• Bubble	• Merge Sort	- Insertion
• Selection	• Bubble	
• Insertion	• Insertion	
• Quick Sort	• Count	
- Heap Sort		

Dheeraj

Ques 5 → Write recursive/ iterative pseudo Code for binary Search. What is the Time and Space Complexity of Linear and Binary Search (Recursive and Iterative

Ans → Iterative ⇒

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

Recursive ⇒

```
int binarySearch (int arr[], int l, int r, int key
 {
      while (l<=r)
      {
           int m = ((l+r)/2);
           if (key == arr[m])
                return m;
```

Dheeraj

```
    else if ( key < arr[m])
        return binarySearch(arr, l, mid-1, key);
    else
        return binarySearch(arr, mid+1, r, key);

    }
    return -1;
}
```

Time Complexity ⇒

- Linear Search – $O(n)$
- Binary Search – $O(\log n)$

Ques 6 ⇒ Write recurrence relation for binary recursive Search.

Ans →

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

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

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

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

$\Rightarrow T(n/4) + 1 + 1$ (from eqn 2)

$= T(n/8) + 1 + 1 + 1$ (from eqn 3)

$\vdots$

$T(n/2^k) + 1$ (k times)

Let $2^k = n$, $k = \log n$

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

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

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

Dheeraj

Ques 7 ⇒ find two indexes such that $A[i] + A[j] = k$ In minimum time Complexity

Ans ⇒

```
for(int i=0; i<n; i++)
  {
     for(int j=0; j<n; j++)
       {
          if(a[i]+a[j]==k)
             printf("%d%d", i,j);
       }
  }
```

Ques 8 ⇒ Which Sorting is best for practical Uses? Explain.

Ans ⇒ QuickSort 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.

Ques 9 ⇒ What do you mean by number of Inversions in an array? Count the number of inversions in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} Using merge sort.

Ans ⇒ • A Pair (A[i], A[j]) is said to be inversion if
- A[i] > A[j]
- i < j

• Total no. of inversion in given array are 31 Using merge sort.

Dheeraj

Ques 10 ⇒ In which Cases Quick Sort will give the best and the worst case time Complexity?

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

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

Ques 11 ⇒ Write Recurrence Relation of Merge and Quick Sort in best and wrost Case? What are the similarities and differences between Complexities of two algorithm and why?

Ans ⇒ **Merge Sort ⇒**

Best Case :- $T(n) = 2T(n/2) + O(n)$

Wrost Case :- $T(n) = 2T(n/2) + O(n)$

$O(n\log n)$

Quick Sort ⇒

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

Wrost Case :- $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

Dheeraj

In Quick Sort ~~the~~ the array of elements is divided into parts repeatedly until it is not possible to divide it further. It is not necessary to divide half.

In Merge Sort the elements are split into two sub-array (n/2) again and again until only one element is left.

Ques 12 ⇒ Selection Sort is not stable by default but can you write a version of Stable Selection

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-j];
           min--;

        }
     a[i] = key;

  }
```

Ques 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 modified bubble sort, includes a flag that is set if a exchange is made after an entire pass over the array. If no exchange is made, then it should be clear the array is already order because no two elements need to be switched. In that case sort is ended.

```
Void bubble(int a[], int n)
{
    for (int i=0; i<n; i++)
    {
        int swaps=0;
        for(int j=0; j<n-i-j ; 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;
    }
}
```