Q1. Write Linear Search Psuedo code to search an element in a sorted array with minimum comparison.

=)

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

Q2. Write Psuedo code for iterative & recursive insertion sort. Insertion sort is called Online sorting. Why? What about other sorting algorithms that has been discussed?

⇒ 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];
            j--;
        }
        arr[j+1] = x;
    }
}
```

Recursive :

```
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 used to know anything about what values it will sort and information is required while algorithm is running.

Other Sorting Algorithms :-

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

Q3. Complexity of all sorting algorithm that has been discussed in lecture.

⇒

Sorting Algorithms	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	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

Q4. Divide all sorting algorithms into Inplace | stable | Online sorting.

Ans -

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		

Q5. Write recursive / iterative Psuedo code for linear search, What is the Time & Space Complexity of Linear & Binary Search.

Ans - Iterative:

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

Time complexity:-

1) Linear search – O(n)
2) Binary search – O(logn)

Q6. Write recurrence relation for binary recursive search.

⇒ $T(n) = T(n/2) + 1$ – ①

$T(n/2) = T(n/4) + 1$ – ②

$T(n/4) = T(n/8) + 1$ – ③

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

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

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

$\vdots$

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

Let $g^k = n$

$k = \log n$

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

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

$T(n) = O(\log n)$ → Answer.

Q7. Find two indexes such that $A[i] + A[j] = K$ in minimum time complexity.

⇒

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

Q8. Which sorting is best for practical uses? Explain.

⇒ 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.

Q9. What do you mean by inversions in an array? Count the no. of inversions in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

⇒ A Pair (A[i], A[j]) is said to be inversion if

- A[i] > A[j]
- i < j
- Total no. of inversions in given array are 31 using merge sort.

Q1 Q10. In which cases Quick Sort will give least & worst case time complexity.

A ⇒ Worst Case $O(n^2)$ → The worst case occurs when the pivot element is an element (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

Q11. Write Recurren Relation of Merge / Quick Sort in best & worst case. What are the similarities & difference between complexities of two algorithm & why?

⇒ Merge Sort :-

Best Case → $T(n) = 2T(n/2) + O(n)$

Worst Case → $T(n) = 2T(n/2) + O(n)$

Quick Sort :-

Best case → $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$

Worst case → $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

In quick sort, array of elenert is divided into 2 parts respectively until it is not possible to divide it further.

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

Q12. 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;
}
```

Q13. Bubble sort scans array even when array is sorted. Can you modify the bubble sort so that it doe snot scan the whole array once it is sorted.

⇒ A letter version of bubble sort, known as m bubble sort includes a flag that is set of a exchange is made after an entire pass over. If no exchange is made then it should be called the arrays is already order because no two elements need to be switched.

```
void bbbobble (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;
                  swap ++;
          }
      }
      if (swap == 0)
      break;
    }
}
```