Q1.) Write linear Search pseudocode to search an elemnt in sorted array with minimum comperisons

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

Q2.) Write pseudo Code for iterative and recursive insertion sort. Insertion sort is celled online sorting. Why? What about other sorting algorithm that has been discussed in lectures?

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 -

```
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 & the information is requested WHILE the algorithm is running.

Other Sorting algorithm

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

Q3) Complexity of all the sorting algorithm that has been discussed in lectures.

	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 the sorting algorithm into inplace/stable/online sorting

Inplace Sorting
- Bubble
- Selection
- Insertion
- Quick sort
- Heap sort

Stable Sorting
- Merge sort
- Bubble
- Insertion
- Count

Online sorting
- Insertion

Q5) Write recursive/iterative pseudo code for binary search. What is the Time & Space Complexity of Linear & Binary Search

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;
    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 - $O(\log n)$

Q1) Write recurrence relation for binary recursive search.

Ans $T(n) = T(n/2) + 1$ — ①

$T(n/2) = T(n/4) + 1$ — ②

$T(n/4) = T(n/8) + 1$ — ③

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

$= T(n/4) + 1 + 1$ (From eqn 2)

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

⋮

$= T(n/2^K) + 1$ (K Times)

Let $2^K = n$

$K = \log n$

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

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

$T(n) = O(\log n)$

Q7) Find two indexs 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);
    }
}
```

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

Ans Quicksort is fastest general-purpose sort. In most practical situation quicksort is the method of choice. If stability is important & space is available, mergesort might be best.

Q9) What do you mean by number of inversion in an array? Count the number of inversion 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 is
• A[i] > A[j]
i > j
• Total no of inversion in given array are 31 using merge sort

Q10) 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 one extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted & 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.

Q11) Write Recurrence relation of merge sort & quick sort in best & worst case. What are similarities & differences b/w

complexities of two algorithm & why?

Ans Merge Sort-

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

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

$O(n \log n)$

Quick Sort-

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

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

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

In Merge Sort the elemnts are split into two sub array $(n/2)$ again & again until enter only one element is left.

(Q12) Selection sort is not stable by default but you can 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;
}
```

(Q13.) 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, include a flag that is set of an 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 element need to be switched. In that case sort is exchanged.

```
void bubble (int a[], int n)
{
    for (int i=0; i<n; i++)
    {
        int swap = 0;
        for (int j=0; j<n-i-1; j++)
        {
            if (a[j] > a[j+1])
            {
                int t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                swap++;
            }
        }
        if (swaps == 0)
            break;
    }
```