Ques-1. Write linear search pseudo code to search an element in a sorted array with minimum no. of comparison.

Sol^n:

```
void linearSearch(int A[], int n, int key)
{   int flag=0
    for(int i=0; i<n; i++)
        { if (A[i]==key)
              { flag=1;
                 break;
              }
        }
    if(flag==0)
        cout<<"Not found"
    else
        cout<<"found";
}
```

Ques.2 Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting that has been discussed.

Sol^n:

Iterative:

```
for i=1 to n-1
        t=A[i], j=i-1
    while(j>=0 && A[j]>t)
        { if (A[j+1] = A[j])
              j--
        }
        A[j+1] =t;
}
```

Recursive:

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

Insertion Sort is an Online algorithm because insertion sort considers one input element per iteration and produces a partial solution without considering future elements.
But in case of other sorting algorithm, we require access to the entire input, thus they are offline algorithm.

Ques-3: Complexity of all sorting algorithm that has been discussed.
Soln,

Algorithm	Worstcase	BestCase	Average case.
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Count Sort	$O(n+K)$	$O(n+K)$	$O(n+K)$
quick Sort	$O(n^2)$	$O(n\log(n))$	$O(n(\log n))$
merge sort	$O(n(\log n))$	$O(n\log n)$	$O(n\log n)$
heap sort	$O(n(\log n))$	$O(n(\log n))$	$O(n(\log n))$

Date————.

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

Ans:

Algorithm	Inplace	Stable	Online
Bubble Sort	✓	✓	✗
Selection Sort	✓	✗	✗
Insertion Sort	✓	✓	✓
Count Sort	✗	✓	✗
Merge Sort	✗	✓	✗
quick Sort	✓	✗	✗
Heap Sort	✓	✗	✗

Que-5: Write Recursive/Iterative pseudo code for binary search. What is the time and space complexity linear/Binary search (Recursive and Iterative)

Sol^n: Recursive

```
int binarySearch (int arr[], int l, int r, int key)
{ if (r>=l)
    { int mid= l+(r-l)/2;
        if ('arr[mid] == key) return mid;
      if (arr[mid] > key)
     return binarySearch (arr, l, mid-1, key);
    return binarySearch (arr, mid+1, r, key)
  }
  return -1;
}
```

Date

Iterative :

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

	Time complexity		Space complexity	
	Recursive	Iterative	Recursive	Iterative
Linear search.	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Binary Search.	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$

Que. 6 Write Recurrance Relation for binary recursive search.

Ans.

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

Que.7: find two indices such that $A[i]+A[j]=K$ in minimum time complexity.

Solⁿ.

```
void Sum (int A[], int K, int n)
{   sort (A, A+n);
        int i=0, j=n-1;
    while (i<j)
    { if (A[i]+A[j]==K)
            break;
      else if (A[i]+A[j]>K)
            j--;
      else
            i++;
    }
    print (i, j).
}
```

Here sort function has $O(n(\log n))$ complexity and for while loop it is $O(n)$

$\therefore$ Overall complexity $= O(n(\log n))$

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

Ans: In practical uses, we mostly prefer merge sort because of its stability and it can best for very large data. further more, the time complexity of merge sort is same in all cases that is $O(n(\log n))$.

TeacEeHs Sign

Ques-11. Write Recurrance relation of merge sort and Quick Sort in best and worst case. What are the similarties between complexities of two algorithms and why?-

Sol^n:

Algorithm	Recurrance Relation	
	Best case	Worst Case
Quick Sort	$T(n) = 2T(n/2) + n$	$T(n) = T(n-1) + n$
Merge Sort	$T(n) = 2T(n/2) + n$	$T(n) = 2T(n/2) + n$

Both the algorithms are based on the divide and conqueor algorithm. Both the algorithms have the same time complexity in the best case and average because both the algorithm divides array into subparts, sort them and finally merge all the sorted parts.

Ques 12: Selection sort is not stable by default but can you write a version of stable selection sort.

Sol^n: As the selection sort is not stable because - it changes the relative position of same elements after sorting.
Selection sort can be made stable if instead of swapping the minimum element is placed in its position without swapping i.e by placing the number in its position by pushing every element one step forward.
In simple words use insertion sort technique which means inserting element in its correct place.

i.e O(nlogn).

Pseudo code for stable Selection sort:

```
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;
   }
}
```

Que-13: Bubble Sort scans whole array even when array is sorted. Can you modify the bubble sort so that it doesn't scan the whole array once it is sorted.

Soln: We can modify bubble sort by placing a flag variable. If array is already sorted we can halt the process by checking the flag variable if its value changes or not.

Pseudo code for Modified bubble sort

```
void bubble (int A[], int n)
{  for (int i=0; i<n; i++)
   {  int swaps = 0;
```

```
for (int j=0 ; j<n-i-1 ; j++)
  { if (A[j] > A[j+1])
     { Swap(A[j], A[j+1]);
        Swaps++;
     }
  }
  if (swaps == 0)
     break;
 }
}
```

Que-14: Your computer has a RAM of 2 GB and you are given an array of 4 GB of sorting. Which algorithm you are going to use for this purpose and why. Also explain the concept of External and Internal sorting.

Ans: For the array of 4 GB, we use the External sorting because array size is greater than the RAM of our computer.

→ External Sorting: These are sorting algorithms that can handle large data amounts which cannot fit in the main memory. Therefore only a part of the array resides in the RAM during execution.

Example: K-way Merge Sort.

→ Internal Sorting: These are sorting algorithms where the whole array needs to be in the RAM during execution.

Ex: Bubble sort, Selection sort etc.

Teacher's Sign