Q1) Write linear search pseudo code to search an element in a sorted array with minimum comparisons.

TUTORIAL-1

```
for (i=0 to n)
{
    if (arr[i] == key)
        return key;
}
return -1;
```

Q.2 Write Pseudo code for iterative and recursive insertion sort. Insertion sort is called online sort. Why? What about other sorting algorithms that has ~~been~~ 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 need ~~used~~ to know anything about values it will sort and information is requested while algorithm is running.

Other ~~online~~ sorting algos:-
Bubble, selection, merge, quick, heap.

Akash

lecatures.

Sorting Algo	Best	Worst	Avg.
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q.4) Divide all sorting algorithms into inplace / stable / online.

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

Q.5) Write iterative / recursive pseudocode for binary search. What is the time and space complexity of linear and binary search.

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;
        else if (key < arr[m])
            r = m-1;
        else
            l = m+1;
    }
    return -1;
}
```

Akanksha.

Recursive:-

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

	Time Complexity	Space Complexity
Linear search	$o(n)$	$o(1)$
Binary search	$o(\log n)$	$o(n)$

Q.6) Write recurrence relation for binary search (recursive).

$T(n) = T(n/2) + 1$ — ①

$T(n/2) = T(n/4) + 1$ — ②

$T(n/4) = T(n/8) + 1$ — ③

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

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

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

$\vdots$

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

let $2^k = n$

$k = \log n$

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

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

$T(n) = \cancel{\log n}\ o(\log n)$

Akash

Q.7) Find two index 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 %d", i,j);
    }
}
```

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

Quick Sort is the fastest general-purpose sort. In most practical situations, quicksort is the method of choice. ~~As~~ stability is important and if space is available, merge sort 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, 10, 1, 20, 6, 4, 5}. using merge sort

A pair (A[i], A[j]) is said to be inverted if
- A[i], > A[j]
- i < j

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

Q.10) In which case, quick sort will give least and worst case time complexity.

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 select pivot element as a mean element.

Q.11) Write recurrence relation of merge/quicksort in best & worst case. What are the similarities and differences b/w complexities of two algorithms and why?

Merge Sort :- Best case - $T(n) = 2T(n/2) + O(n)$
Worst Case - $T(n) = 2T(n/2) + O(n)$ $\Big\}$ $O(n\log n)$

Quick Sort :- Best case - $T(n) = 2T(n/2) + O(n)$ — $O(n\log n)$
Worst case - $T(n) = T(n-1) + O(n)$ — $O(n^2)$

Akanksha

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

In mergesort, the elements are split into 2 subarrays (n/2) again & 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 code?

```
for(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;
}
```

Q.13) Bubble sort scans ~~every~~ 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.

```
void bubblesort (int arr[], int n)
{
for(int i=0; i<n-1 ; 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;
        swap++;
      }
    }
    if(swap ==0) return;
  }
}
```

Akanksha