Tutorial-3

1. Write linear search pseudocode to search an element in a sorted array with minimum comparisions.

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

2. Write pseudo code for iterative and recurrsive insertion sort. Insertion sort is called Online sorting. Why? What about other sorting algorithms that has been discussed in lectures?

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

```
        j--;
    }
    arr[j+1] = nth;
}

for(i=1 to n)
{
    key ← A[i]
    j ← (i-1)

    while(j>=0 and A[j]>key)
    {
        A[j+1] ← A[j]
        j ← j-1
    }

    A[j+1] ← key
}
```

Insertion sort is online sorting because it doesn't know the whole input, more input can be inserted with the insertion sorting is running.

3. Complexity of all the sorting algorithms that has been discussed in lectures.

Name	Best	Worse	Average
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	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$

4. Divide all the sorting algorithms into inplace/stable/online sorting.

Inplace sorting	Stable sorting	Online sorting
Bubble	Merge	Insertion
selection	Bubble	
Insertion	Insertion	
Quick	Count	
Heap		

5. Write recursive/iterative pseudo code for binary search. What is the Time and space complexity of Linear and Binary search (Recursive and Iterative)?

```
int Binary(int a[], int l, int r, int x)
{
    if(r>=l)
    {
        int mid = l+(r-l)/2;

        if(arr[mid]==x)
            return mid;
        else if(arr[mid]>x)
            return Binary(arr, l, mid-1, x);
        else if(arr[mid]<x)
            return Binary(arr, mid+1, h, x);
    }
    return -1;
}

int Binary(int arr[], int l, int r, int x)
{
    while(l<=r)
    {
        int mid = l+(r-l)/2;

        if(arr[mid]==x)
            return mid;
        else if(arr[mid]>x)
            r=mid-1;
        else
            l= mid+1;
    }
    return -1;
```

Date / /

Time complexity of
Binary Search ⇒ O(log n)
Linear search ⇒ O(n)

6. Write recurrence relation for Binary recursive search.

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

where, T(n) is the time required for Binary search in an array of size 'n'.

7. Find two indexes such that A[i] + A[j] = k minimum time complexity.

```
int find(A[], n, key)
{
   Sort(A, n)
   for(i=0 to n-1)
   {
      x = Binarysearch(A, 0, n-1, key-A[i])
      if(n)
         return 1;
   }
   return -1;
}
```

Time complexity = $O(n \log n) + n \times O(\log n)$
= $O(n \log n)$

8. Which sorting is best for practical cases? Explain

- Quick sort is the fastest general purpose sort.
- In most practical situations, quicksort is the method of choice. If stability is important and space is available, merge sort might be best.

10. In which cases Quick sort will give the best and the worst case time complexity?

The worst case time complexity of Quick sort is $O(n^2)$. This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted.

The Best case of Quick sort is when we will select pivot as a mean element.

Time complexity = $O(n \log n) + n \times O(\log n)$
= $O(n \log n)$

9.

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. If stability is important and space is available, merge sort might be best.

10.

10. In which cases Quick sort will give the best and the worst case time complexity?

The worst case time complexity of Quick sort is $O(n^2)$. This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted.

The Best case of Quick sort is when we will select pivot as a mean element.

9. What do you mean by number of inversions in an array? Count the number of inversions in an 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 of a[i] > a[j]
In arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}
Total no. of inversion are 31, using merge sort.

11. Write Recurrence Relation of Merge and Quick sort in best and worst case? What are the similarities and differences between complexities of two algorithms and why?

Recurrence relation of
Merge sort → $T(n) = 2T(n/2) + n$
Quick sort → $T(n) = 2T(n/2) + n$

- Merge sort is more efficient and works faster than quick sort in case of large array size or data sets.
- Worst case complexity for quick sort is $O(n^2)$ whereas $O(n \log n)$ for merge sort.

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

Stable selection sort

```
void stableselection(int arr[], int n)
{
    for(int i=0; i<n-1; i++)
    {
        int min = i;
        for(int j=i+1; j<n; j++)
        {
            if(arr[min] > arr[j])
                min = j;
        }

        int key = arr[min];

        while(min > i)
        {
            arr[min] = arr[min-1];
            min--;
        }
        arr[i] = key;
    }
}
```

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.

```
void Bubble(int arr[], int n)
{
    for(int i=0; i<n; i++)
    {
        int swaps = 0;
        for(int j=0; j<n-i-1; j++)
        {
            if(arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                swaps++;
            }
        }
        if(swaps==0)
            break;
    }
}
```