Date:

Tutorial 3

(a)
```
int linearsearch(int *arr, int n, int key)
    for i>=0 to n-1
        if arr[i] = key
            return;
    return -1
```

(b) iterative insertion sort
```
void insertionsort(int arr[], int n)
{   int i, temp, j;
    for i <- 1 to n
        temp <- arr[i]
        j <- i-1
        while (j>=0 AND arr[j] > temp)
            arr[j+1] <- arr[j]
            j <- j-1
        arr[j+1] <- temp
```

→ recursive insertion sort
```
void insertionsort(int arr[], int n)
    if (n<=1)
        return
    insertionsort(arr, n-1)
    last = arr[n-1]
    j = n-2
    while (j>=0 AND arr[j]>last)
        arr[j+1] = arr[j]
        j--
    arr[j+1] = last.
```

→ Insertion sort is called as online sorting as it does not need to know anything about what values it will sort and the information is requested while the algo. is running

(c) Selection Sort
time complexity = best case → $O(n^2)$
worst case → $O(n^2)$

- insertion sort
time complexity = best case → $O(n)$
worst case → $O(n^2)$

→ merge sort
time complexity = best case → $O(n\log n)$
worst case → $O(n\log n)$

→ quick sort
time complexity = best case → $O(n\log n)$
worst case → $O(n^2)$

→ heap sort
time complexity = best case → $O(n\log n)$
worst case → $O(n\log n)$

→ bubble sort
time complexity = best case → $O(n^2)$
worst case → $O(n^2)$

(d)

sorting	inplace	stable	online
selection	✓	✓	✓
insertion	✓	✓	
Merge			
Quick	✓		
Heap	✓	✓	
Bubble	✓		

(e) iterative binary search

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

time complexity
B.C → O(1)
avg → O(log n)
worst → $O(\log_2 n)$

→ Recursive Binary Search

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

time complexity
Best Case = O(1)
avg = O(logn)
worst = O(logn)

(f) Recurrence Relation for binary search

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

(g) $A[i] + A[j] = K$

(h) Quick sort is the fastest general purpose sort. In most ~~pa~~ practical situation, quick sort is a method of choice. if stability is important & space is available then merge sort is good

(i) Inversion count is a measure of how far or how close the array is from being completed sorted. For a completed sorting the inversion count is 0, but if array is reversely sorted then the inversion count is max.

(j) The worst t.c of quick sort is $O(n^2)$. It occurs when the pivot element is either first or last. This happens if the array is sorted or reversly sorted.

(k) Recurrence Relation of

merge sort → $T(n) = 2T(n/2) + n$

quick sort → $T(n) = 2T(n/2) + n$.

→ merge sort works faster than quick sort in case of large array

→ Worst case time complexity of Q.S is $O(n^2)$ & M.S is $O(n\log n)$