Date

Tutorial - 03

1.
```
bool linear_search(int a[], int e, int n){
    bool flag = false;
    for(int i=0; i<n; i++){
        if(a[i]==e){
            flag = true;
        }
        else flag = false;
    }
    return flag;
}
```

2. Iterative:
```
void insertion_sort(int a[], int n){
    for(int i=1; i<n; i++){
        int value = a[i];
        int j = i;
        while(j>0 && a[j-1]>value){
            a[j] = a[j-1];
            j--;
        }
        a[j] = value;
    }
}
```

Recursive -
```
void insertionSort(int a[], int i, int n){
    int value = a[i];
    int j = i;
    while(j>0 && a[j-1]>value{
        a[j] = a[j-1];
```

```
            j--;
        }
        a[j] = value;
        if (i+1 <= n) {
            insertionSort(arr, i+1, n);
        }
    }
```

Insertion sort is called an online sorting algorithm because insertion sort considers an input element per iteration and produces a partial solution without considering future elements.

Date ___/___/___

Saathi

3.

Sorting Technique	Best	Average	Worst
• Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
• Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
• Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
• Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
• Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
• Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
• Count Sort	$O(n + range)$	$O(n + range)$	$O(n + range)$

4.

Sorting Technique	Inplace	Stable	Online
• Bubble Sort	✓	✓	×
• Selection Sort	✓	×	×
• Insertion Sort	✓	✓	✓
• Quick Sort	✓	×	×
• Merge Sort	×	✓	×
• Heap Sort	✓	×	×

5. Iterative -

```
int binarySearch(int a[], int x){
    int low = 0, high = a.length-1;
    while(low <= high){
        int mid = (low+high)/2;
        if (x == a[mid])
            return mid;
        else if (x < a[mid])
            high = mid-1;
        else low = mid+1;
    }
    return -1;
}
```

Date

Saathi

Recursive -

```
int binarySearch(int a[], int low, int high, int x){
    if( low > high)
        return -1;
    int mid = (low + high)/2;
    if (x == a[mid])
        return mid;
    else if (x < a[mid])
        return binarySearch(A, low, mid-1, n);
    else
        return binarySearch(A, mid+1, high, n);
}
```

Time Complexity → Iterative - O(log n)
Recursive - O(log n)

Space Complexity → Iterative - O(1)
Recursive - O(log n)

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

using masters method,

$a = 1, \quad b = 2, \quad f(x) = 1$

$c = \log_b a$

$= \log_2(1) = 0$

$n^c = n^0 = 1$

$f(x) = n^c$

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

$\boxed{T(n) = O(\log n)}$

Page No.

7. vector <int> find(A[], k, n) {
    vector <int> sol;
    for i=0 to n-1
        for j=0 to n
            if A[i] + A[j] = k
                sol.push_back(i);
                sol.push_back(j);
    return sol;

8. Quick sort is the fastest general purpose sort. In most practical situation quick sort is a method of choice. If stability is important and space is available, merge sort might be best. In some performance-critical applications, the focus maybe on just sorting numbers, so it is reasonable to avoid the costs of using reference and sort primitive types instead.

9. Inversion count for an array indicates how far or close the array is from being sorted. If the array is already sorted then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is the maximum

Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

Pair of inversion array arr[] = {(7,1), (7,6), (7,4), (7,5), (21,8), (21,10), (21,1), (21,20), (21,6), (21,4), (21,5), (31,8), (31,10), (31,1), (31,20), (31,6), (31,4), (31,5), (8,6), (8,4), (8,5), (10,1), (8,1), (10,6), (10,4), (10,5), (20,6), (20,4), (20,5), (6,4), (6,5)}

=> count = 31

Date ___/___/_____

Saathi

10. Best case - The best case occurs when the ~~positio~~ partition process always picks the middle element as pivot. Following is the best case for recurrence relation -

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

Worst Case - The worst case occurs when the partition process always picks greater or smallest element as pivot. If above partition strategy is considered where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order.

11. Quick sort,

Recurrence Relation : Best case - $T(n) = 2T(n/2) + O(n)$

Worst case = $T(n) = T(n-1) + O(n)$

Merge sort,

Recurrence Relation : Best case - $2T(n/2) + O(n)$

Worst case - $2T(n/2) + O(n)$

Time Complexities

	Best case	Worst case
Quick Sort	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$

12.
```
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;
    }
}
```

13.
```
void bubbleSort(int a[], int n){
    int temp;
    bool flag=false;
    for(int i=0; i<n-1; i++){
        flag=false;
        for(int j=0; j<n-i-1; j++){
            flag=true;
            if(a[j]>a[j+1]){
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
        if(flag==false)
            break;
    }
}
```

Saathi

Date

14. As the size of given array exceeds the size of RAM therefore, we will use k-way merge sort as sorting technique. It takes a part of array and sort it, therefore, whole array is not loaded into main memory altogether.

External Sorting: This algorithms loads a part of array and sort it, whole array is not loaded into the RAM, especially used to sort arrray of large size.
Eg → k-way merge sort.

Internal Sorting: These algorithm needs whole array altogether in RAM during execution.
Eg → Bubble sort.