# Tutorial - 3

Unzila
Sec-I

Ans 1 →

```
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";
```

Ans 2 → Iterative :-

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

Insertion Sort is called Online Sorting because Insertion Sort considers one input element per iteration and produces a partial solution without considering future elements. But other sorting algorithm requires access to the entire input, thus considered as offline algorithm.

Ans 3→

| Algorithm | Time Complexities | | |
|---|---|---|---|
| | Best | Average | Worst |
| Bubble Sort | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| Selection Sort | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Insertion Sort | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| Count Sort | $O(n+k)$ | $O(n+k)$ | $O(n+k)$ |
| Quick Sort | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ |
| Merge Sort | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ |
| Heap Sort | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ |

Ans 4→

| Algorithm | Inplace | Stable | Online |
|---|---|---|---|
| Bubble Sort | ✓ | ✓ | ✗ |
| Selection Sort | ✓ | ✗ | ✗ |
| Insertion Sort | ✓ | ✓ | ✓ |
| Count Sort | ✗ | ✓ | ✗ |
| Merge Sort | ✗ | ✓ | ✗ |
| Quick Sort | ✓ | ✗ | ✗ |
| Heap Sort | ✓ | ✗ | ✗ |

Ans 5→ 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;
}
```

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;
}
```

| Algorithm | Time Complexity | | Space Complexity | |
|---|---|---|---|---|
| | Recursive | Iterative | Recursive | Iterative |
| Linear Search | $O(n)$ | $O(n)$ | $O(1)$ | $O(1)$ |
| Binary Search | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(1)$ |

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

Ans 7→

```
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++;
    }
    printf (i, j);
}
```

Here, sort function has O(nlogn) complexity and for while it is O(n).

So, Time Complexity = O(nlogn)

Ans 8→ Quick sort is the fastest general purpose sort in most practical situation. But we mostly prefer Merge Sort because of its stability and it would be best for very large data. Further, time complexity of Merge Sort is same in all cases that is O(nlogn).

Ans 10→ When the array is already sorted or sorted in reverse order, quick sort gives the worst case time complexity i.e $O(n^2)$ but when the array is totally unsorted it will give best case time complexity i.e O(nlogn)