Data Structure:

- Structure of Data
  - The organization of data in memory.
  - Operations performed on that data.

- Types of data structures:
  - Linear - Array, Stack, Queue, Linked List.
  - Non-Linear - Tree, Graph
  - Associative:- data is stored in key value pair e.g. HashTable

- Usually data structures are represented as "Abstract Data Types".
-
- Example Data Structures:
  - Array, Stack, Queue, Linked List, Tree, Graph, Hashtable, etc.

Time Complexity:

- Approximate measure of time required for completing any algo/operation.
- Example: Factorial:

```
res = 1;

for(i=1; i<= num; i++)
    res = res * i;

printf("factorial : %d\n", res);
```

- Time complexity of any algorithms depends on number of iterations in that algorithms .

- In above program, time is proportional to number whose factorial is to be calculated.

- T α n . Hence time complexity is O(n) -> Order of n.

- If time required for any algo is constant (not dependent on any factor)
  i.e. T = k. Then time complexity is represented as O(1).

- Time complexity is represented in "Big O" notation.

Selection Sort:

```
for(i=0; i<n-1; i++)
{
    for(j=i+1; j<n; j++)
    {
       if(a[i] > a[j])
           SWAP(a[i], a[j]);   // macro
    }
}
```

- Number of iterations = 1 + 2 + 3 + ... + n-1 = n (n-1) / 2
- T α n (n-1) / 2
- T α n^2 - n
- If n >> 1, then n^2 >>>> n => Hence all lower order terms can be ignored.
- T α n^2
- Time complexity = O(n^2)

Bubble Sort:

```
for(i=0; i<n-1; i++)
{
   for(j=0; j<n-1; j++)
   {
      if(a[j] > a[j+1])
         swap(a[j], a[j+1]);  // macro
   }
}
```

- Number of iterations = (n-1)(n-1)
- T α n^2 - 2n + 1
- If n >> 1, then n^2 >>>> n => Hence all lower order terms can be ignored.
- T α n^2
- Time complexity = O(n^2)

Linear Search:

```
for(i=0; i<n; i++)
{
  if(key==a[i])
   return i; // element found at index i
}
return -1; // ele not found
```

- Best case : "key" is found at index 0.
  - Number of iterations = 1
  - T = k
  - O(1)
- Worst case : "key" is found at last index (n-1).
  - Number of iterations = n
  - T α n
  - O(n)
- Average case : "key" is found in middle.
  - Number of iterations (average) = n / 2
  - T α n / 2
  - T α n
  - O(n)

Binary Search:

1 start
2. accept Arr (asc sorted) ,Accept Key(element to search)
3. assign size =10 left=0, right=size-1
4. check left<=right
    if yes
       mid=(left+right)/2
       check(key==arr[mid]
          if yes
             return mid
        check (key>arr[mid]
           if yes
             left=mid+1
          check (key<arr[mid]
            if yes
               right=mid-1
          go to 4
5   if not
        return -1
6. stop

```
while(left <= right)
{
    mid = (left+right) / 2;
    if(key == arr[mid])
      return mid; // element found at index "mid"
   if(key <arr[mid])
     right = mid - 1;
   else
   left = mid + 1;
}
return -1;
```

- Array must be sorted to use binary search.
- $2^i = n$ -> where 'n' is number of elements, 'i' is number of iterations
- i log 2 = log n
- i = log n / log 2
- T α log n / log 2
- T α log n
- Time complexity : O (log n)

Most common time complexities:

- O(1)
- O(log n)
- O(n)
- O(n log n)
- $O(n^2)$

Recursion:

- Function calling itself is called as "recursive function".
- From programming perspective two things are important:
  - Explain process in terms of itself.
  - Terminating condition.
- Advantages:
  - Simplifies / reduce the code.
- Disadvantages:
  - Increased space complexity (function activation records are created per call).
  - Increased time (time required to create activation record and call the function).