SUBJECT CODE: CS301

SUBJECT NAME: DATA STRUCTURES AND ALGORITHMS

UNIT - 3 (MY NOTES)

SEMESTER: 3

Introduction to Data Structures

👉 **Data Structure:**
It is simply a way of **organizing and storing data** so that we can use it easily and efficiently inside programs.

🔹 Types of Data Structures

a) Primitive Data Structures

- Basic data types given by programming languages
- Examples: int, float, char, pointer

b) Non-Primitive Data Structures

✔️ **Linear Data Structures** – elements are arranged in order

- Array
- Stack
- Queue
- Linked List

✔️ **Non-Linear Data Structures** – elements are connected in hierarchy/network

- Tree
- Graph

🔹 Classification of Data Structures

✅ **Static Data Structure**

- Size is fixed before execution
- Example: Array

✅ **Dynamic Data Structure**

- Size can change during runtime
- Example: Linked List

🔹 Why Do We Need Data Structures?

- Helps use memory efficiently
- Makes searching and sorting faster
- Makes programs more organized and easier to manage

Algorithms

👉 **Algorithm:**

A step-by-step procedure used to solve a problem.

🔹 Characteristics of a Good Algorithm

- **Input** – takes data
- **Output** – gives result
- **Definiteness** – steps must be clear
- **Finiteness** – must end after some steps
- **Effectiveness** – easy to execute

🔹 Example Algorithm (Largest Number)

1. Start
2. Read A and B
3. If A > B print A
4. Else print B
5. Stop

Analysis of Algorithms

Used to check **how efficient** an algorithm is.

🔹 Types of Analysis

- **Best Case** → minimum time taken
- **Average Case** → normal situation
- **Worst Case** → maximum time taken

Time Complexity

👉 Time complexity tells us **how fast an algorithm runs** depending on input size n.

🔹 Common Complexities

- O(1) → Constant time
- O(log n) → Logarithmic
- O(n) → Linear
- O(n log n)
- $O(n^2)$ → Quadratic
- $O(2^n)$ → Exponential

Example:

```
for(i=0;i<n;i++)
    print(i);
```

➡ Runs n times → **O(n)**

Space Complexity

👉 Amount of memory used by an algorithm.

Includes:

- **Fixed part** → variables, constants
- **Variable part** → dynamic memory allocation

Asymptotic Notations

Used to describe how an algorithm grows when input increases.

🔹 Big-O (O)

- Shows upper bound (Worst Case)

🔹 Omega (Ω)

- Shows lower bound (Best Case)

🔹 **Theta (Θ)**

- Shows tight bound (Average behaviour)

Recursion (Basic Idea)

👉 Recursion means a function **calling itself** to solve smaller parts of a problem.

Parts of Recursion

- **Base Case** → stopping condition
- **Recursive Case** → function repeats

Example:

factorial(n):
if n==1 return 1
else return n * factorial(n-1)

Abstract Data Type (ADT)

👉 ADT explains **what operations** a data structure can do, not how it is implemented.

Example:

- Stack ADT → push(), pop(), peek()

Focus on:
✔️ Operations performed
❌ Internal implementation