Week 2

Q1What is Data Structure and Algorithms?

Data Structure:

A data structure is a way of organizing and storing data so that it can be accessed and used efficiently. It defines the layout or structure of the data in memory, which determines how data can be added, removed, or retrieved.

Types of Data Structures:

1. Linear Data Structures:
   - Array: A collection of elements stored at contiguous memory locations.
   - Linked List: A sequence of elements where each element points to the next.
   - Stack: A LIFO (Last In, First Out) structure.
   - Queue: A FIFO (First In, First Out) structure.
2. Non-Linear Data Structures:
   - Tree: A hierarchical structure with a root node and child nodes.
   - Graph: A collection of nodes connected by edges.

Algorithm**:**

An algorithm is a step-by-step procedure or set of rules for solving a specific problem or performing a computation. Algorithms are used to manipulate data in data structures efficiently.

Types of Algorithms:

1. Sorting Algorithms: Bubble Sort, Merge Sort, Quick Sort, etc.
2. Searching Algorithms: Binary Search, Linear Search, etc.
3. Graph Algorithms: Dijkstra's, BFS, DFS, etc.
4. Dynamic Programming: Solves problems by breaking them into subproblems (e.g., Fibonacci, Knapsack).
5. Greedy Algorithms: Makes locally optimal choices to achieve a global solution (e.g., Huffman Encoding).

PRACTICAL APPLICATIONS:

Real-World Examples:

1. Social Media Feed
   - Data Structure: Queue for posts
   - Algorithm: Sorting by relevance
2. Google Maps
   - Data Structure: Graph for roads

    - Algorithm: Dijkstra's for shortest path
3. Auto-complete
    - Data Structure: Trie for words
    - Algorithm: Pattern matching

Aspect	Linear	Non-Linear	Static	Dynamic
Structure	Sequential	Hierarchical/Graph-like	Fixed size	Flexible size
Memory	Contiguous or linked	Non-contiguous	Fixed during compile time	Allocated during runtime
Examples	Array, Stack, Queue	Tree, Graph	Static Arrays	Linked List, std::vector

Use case

1.Arrays in Java:
Use Cases:
Fixed-size collections where size is known beforehand
Direct/random access to elements (O(1) time complexity)
Memory-efficient storage of primitive data types
Matrix/grid representations (using 2D arrays)

```
// 1D Array
int[] scores = new int[5];  // Fixed size of 5
scores[0] = 95;  // Direct access


// 2D Array for matrix
int[][] matrix = new int[3][3];  // 3x3 grid
```

2.ArrayList:
Use Cases:
Dynamic size collections
Frequent read operations
When index-based access is needed
When you need to add elements at the end frequently

```
ArrayList<String> names = new ArrayList<>();
names.add("John");  // O(1) amortized
names.get(0);       // O(1) access
```

3.LinkedList:

Use Cases:
Frequent insertions/deletions in the middle
Implementing queues or stacks

When memory usage needs to be exactly proportional to size
When you don't need random access

```
LinkedList<String> tasks = new LinkedList<>();
tasks.addFirst("High Priority");  // O(1)
tasks.removeLast();             // O(1)
```

4.Stack(LIFO)

Browser History Management
Undo/Redo Operations
Expression Evaluation
Function Call Management

5.Queue
QUEUE (FIFO - First In First Out)
Print Spooler
Customer Service System
Message Processing System
Task Scheduling
Key Differences in Use Cases:

1. Stack:
   - When you need last-accessed elements first
   - When tracking state that needs to be unwound
   - When implementing recursive algorithms iteratively
   - When order needs to be reversed
2. Queue:
   - When processing needs to be fair (first-come-first-served)
   - When maintaining order of operations
   - When implementing breadth-first algorithms

6.Tree And Graphs
Tree
File System Structure
Binary Search Tree for Efficient Search
HTML DOM Structure

7. GRAPHS
Social Network Connections
Navigation System
Network Routing

Key Use Cases:
Trees:

- Hierarchical data representation
- Fast search operations (BST)
- File systems
- XML/HTML DOM
- Decision trees
- Company organization charts

Graphs:

- Social networks

- Road/transportation networks
- Network routing
- State machines