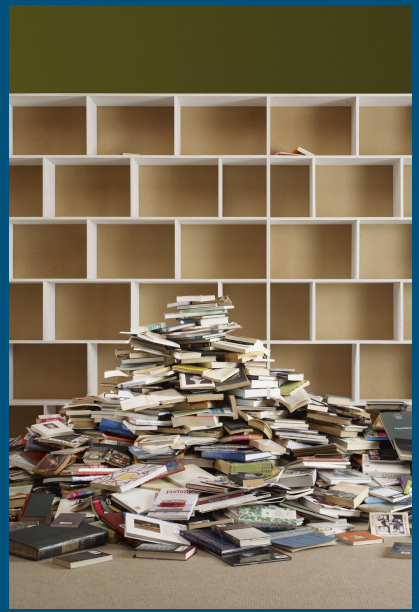
# Introduction to Data Structure & Algorithm

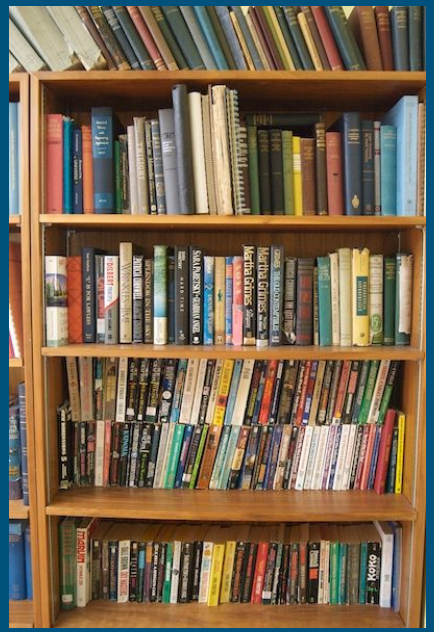
Coding Interview Prep @ Ostad

- Partharaj Deb -

# Introduction to Data Structures?

- A data structure is a way of organizing and storing data to perform operations efficiently.
- It defines a way to represent, store, and organize data to facilitate operations like insertion, deletion, and searching.





# Importance of Data Structures

Why are Data Structures Important?

- **Efficiency**: Efficient data structures lead to faster algorithms.
- **Organization**: Structured data allows for better organization and management.
- **Scalability**: Data structures impact how well a program can scale with increasing amounts of data.

# Common Data Structures

Examples of Common Data Structures

- **Arrays**: Ordered collection of elements.
- **Linked Lists**: Elements connected by pointers.
- **Stacks**: Last In, First Out (LIFO) structure.
- **Queues**: First In, First Out (FIFO) structure.
- **Trees**: Hierarchical structure.
- **Graphs**: Network of nodes and edges.
- **Hash Tables**: Key-value pairs.

# Types of Data Structures

Classification of Data Structures

- **Primitive Data Structures**: Basic building blocks (int, float, char).
- **Non-Primitive Data Structures**: Derived from primitive data structures (arrays, lists).

- **Linear Data Structures**: Elements are stored in a linear sequence (arrays, linked lists).
- **Non-Linear Data Structures**: Elements are not in a sequence (trees, graphs).

# Operations on Data Structures

Common Operations

- **Traversal**: Visiting each element.
- **Insertion**: Adding elements.
- **Deletion**: Removing elements.
- **Search**: Finding an element.
- **Sorting**: Arranging elements in a specific order.
- **Merging**: Combining two data structures.

# Introduction to Algorithms

An algorithm is a step-by-step procedure or formula for solving a problem.

In computer science, it refers to a set of well-defined instructions to perform a task or solve a problem.

# Characteristics of a Good Algorithm

- Input: Zero or more inputs are externally supplied.
- Output: At least one output is produced.
- Definiteness: Each step is precisely defined.
- Finiteness: The algorithm must terminate after a finite number of steps.
- Effectiveness: Each step is simple enough to be carried out in a finite amount of time.

# Why Algorithms Matter

- Efficiency: Optimize performance for speed and resource usage.
- Scalability: Handle increasing amounts of data effectively.
- Problem-Solving: Fundamental to solving complex programming problems.
- Interview Success: Core part of technical interview assessments.

# Types of Algorithms...

- Sorting Algorithms
  - Examples: Bubble Sort, Merge Sort, Quick Sort.
  - Importance: Organize data, used in other algorithms (e.g., binary search).
- Searching Algorithms
  - Examples: Linear Search, Binary Search.
  - Importance: Finding elements efficiently.
- Graph Algorithms
  - Examples: Depth-First Search (DFS), Breadth-First Search (BFS), Dijkstra's Algorithm.
  - Importance: Network analysis, pathfinding.

# Types of Algorithms...

- Dynamic Programming
  - Examples: Fibonacci Sequence, Knapsack Problem.
  - Importance: Optimizing recursive solutions.
- Divide and Conquer
  - Examples: Merge Sort, Quick Sort.
  - Importance: Breaking down problems into manageable parts.
- Greedy Algorithms
  - Examples: Huffman Coding, Prim’s Algorithm.
  - Importance: Making locally optimal choices for global solutions.

# Applications of Algorithms

- **Data Analysis**: Sorting and searching algorithms are essential for processing and analyzing data.
- **Networking**: Graph algorithms like Dijkstra's help in finding the shortest path in networks.
- **Resource Optimization**: Dynamic programming helps in making decisions that optimize resource usage.
- **Compression**: Greedy algorithms like Huffman coding are used in data compression techniques.

# Best Practices

Best Practices for Working with Data Structures & Algorithm

- **Choose Wisely**: Select the appropriate data structure and algorithm for the task.
- **Understand Operations**: Be familiar with the operations needed and their time complexity.
- **Memory Management**: Be mindful of memory usage.
- **Update and Maintain**: As the program evolves, revisit and update data structures & algorithm as needed.

# Question?