Data Structure

Here's a comprehensive Data Structures course syllabus with references to books and online resources.

1. **Introduction to Data Structures**
   - Overview of data structures and their importance
   - Types of data structures: Linear and nonlinear
   - Complexity analysis: Time and space complexities

   References:
   Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein (Chapter 1)
   Online: GeeksforGeeks - Data Structures Basics

2. **Arrays and Strings**
   - Arrays: Definition, operations, applications
   - Strings: Basic operations, pattern matching (Naive, KMP algorithm)

   References:
   Data Structures and Algorithm Analysis in C by Mark Allen Weiss (Chapters 1-2)
   Online: TutorialsPoint - Arrays

3. **Linked Lists**
   - Singly linked list, doubly linked list, circular linked list
   - Operations: Insertion, deletion, traversal
   - Applications: Implementing stacks, queues, and polynomial manipulation

   References:
   Data Structures Through C by Yashavant Kanetkar
   Online: GeeksforGeeks - Linked Lists

4. **Stacks and Queues**
   - Stack: LIFO principle, operations, applications (e.g., expression evaluation, backtracking)
   - Queue: FIFO principle, operations, circular queues, priority queues, dequeues

   References:
   Data Structures and Algorithmic Thinking with Python by Narasimha Karumanchi
   Online: Programiz - Stacks and Queues

5. **Recursion**

   Concepts and implementation
   Tail recursion
   Applications: Factorial, Fibonacci, Tower of Hanoi

   References:
   Algorithms by Robert Sedgewick and Kevin Wayne
   Online: Khan Academy - Recursion

6. **Trees**

   Binary trees: Representation, traversal (in-order, pre-order, post-order)
   Binary search trees (BST)
   Balanced trees: AVL trees, red-black trees
   Heaps: Min-heap, max-heap, heap sort

   References:
   Data Structures and Algorithms in Java by Robert Lafore
   Online: GeeksforGeeks - Trees

7. **Graphs**

   Representations: Adjacency matrix, adjacency list
   Graph traversals: BFS, DFS
   Applications:
   Shortest path (Dijkstra, Floyd-Warshall),
   Minimum spanning tree (Prim's, Kruskal's)

   References:
   Graph Theory with Applications by Bondy and Murty
   Online: GeeksforGeeks - Graphs

8. **Hashing**

   Hash tables: Hash functions, collision handling (chaining, open addressing)
   Applications: Dictionary implementation, LRU cache

   References:
   Data Structures and Algorithms in C++ by Adam Drozdek
   Online: Programiz - Hashing

9. **Searching and Sorting**

    Searching: Linear search, binary search

    Sorting: Bubble sort, selection sort, insertion sort, merge sort, quicksort

    Advanced sorting algorithms: Counting sort, radix sort, heap sort

    References:

    Algorithms, Part I by Robert Sedgewick and Kevin Wayne

    Online: Sorting Visualizations - VisuAlgo

10. **Advanced Topics**

    Trie: Prefix trees and their applications

    Segment trees: Range queries

    Disjoint set (Union-Find): Applications in graph algorithms

    References:

    Competitive Programming 3 by Steven Halim and Felix Halim

    Online: Codeforces and Competitive Programming Resources

Online Resources

MIT OpenCourseWare - Introduction to Algorithms

Coursera: Data Structures and Algorithms by UC San Diego

LeetCode - Data Structures Practice

FreeCodeCamp: Data Structures Tutorials

Algorithm

Here's a comprehensive Algorithm Course Syllabus, along with references to books and online resources.

Module 1: Introduction to Algorithms

- Definition and Importance of Algorithms
- Characteristics of a Good Algorithm
- Algorithm Design Process
- Complexity Analysis: Time and Space Complexity, Asymptotic Notations (Big-O, Big-Θ, Big-Ω)

References:
Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein (Chapter 1)
Online: GeeksforGeeks - Algorithm Basics

Module 2: Sorting and Searching Algorithms

- Sorting: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort
- Advanced Sorting: Heap Sort, Radix Sort, Counting Sort
- Searching: Linear Search, Binary Search, Interpolation Search

References:
Algorithms, Part I by Robert Sedgewick and Kevin Wayne
Online: VisuAlgo - Sorting and Searching

Module 3: Divide and Conquer Algorithms

- Concept and Application of Divide-and-Conquer
- Examples: Merge Sort, Quick Sort, Binary Search
- Matrix Multiplication (Strassen's Algorithm)

References:
Introduction to the Design and Analysis of Algorithms by Anany Levitin (Chapter 4)
Online: Khan Academy - Divide and Conquer

Module 4: Greedy Algorithms

- Greedy Algorithm Design Paradigm
- Applications:
  - Activity Selection Problem
  - Huffman Encoding
  - Minimum Spanning Tree (Prim's and Kruskal's)
- Knapsack Problem (Greedy Version)

References:
Algorithm Design by Jon Kleinberg and Éva Tardos (Chapter 4)
Online: GeeksforGeeks - Greedy Algorithms

Module 5: Dynamic Programming

Principles of Dynamic Programming

Applications:

- Longest Common Subsequence (LCS)
- Matrix Chain Multiplication
- Knapsack Problem
- Floyd-Warshall Algorithm (All-Pairs Shortest Path)

Difference between Greedy and Dynamic Programming

References:

Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein (Chapters 15-16)

Online: Dynamic Programming by Tushar Roy (YouTube)

Module 6: Graph Algorithms

Representation of Graphs: Adjacency Matrix, Adjacency List

Graph Traversals: Breadth-First Search (BFS), Depth-First Search (DFS)

Shortest Path Algorithms: Dijkstra's, Bellman-Ford

Minimum Spanning Tree: Kruskal's and Prim's Algorithms

References:

Introduction to Graph Theory by Douglas B. West

Online: GeeksforGeeks - Graph Algorithms

Module 7: Backtracking and Recursion

Backtracking Principles

Applications:

- N-Queens Problem
- Subset Sum Problem
- Sudoku Solver

Relation to Recursion

References:

The Algorithm Design Manual by Steven Skiena

Online: Backtracking by Programiz

Module 8: Divide and Conquer vs. Dynamic Programming

Comparing the Two Approaches

Case Studies:

- Longest Increasing Subsequence (Dynamic Programming)
- Maximum Subarray Sum (Kadane's Algorithm)

References:

Introduction to Algorithms by Cormen, Leiserson, Rivest (Sections on Divide-andConquer and DP)

Online: FreeCodeCamp - Algorithms

Module 9: Advanced Topics

Computational Geometry: Convex Hull, Closest Pair of Points
Network Flow: Ford-Fulkerson Algorithm
String Matching: Naïve String Matching, Rabin-Karp, KMP, Boyer-Moore

References:
Algorithms on Strings, Trees, and Sequences by Dan Gusfield
Online: Stanford Algorithms Specialization on Coursera

Module 10: NP-Completeness and Approximation Algorithms

NP and NP-Complete Problems
Polynomial-Time Reductions
Approximation Algorithms: Vertex Cover, Traveling Salesman Problem

References:
Computational Complexity: A Modern Approach by Arora and Barak
Online: MIT OpenCourseWare - Advanced Algorithms

Online Resources

Coursera: Algorithms Specialization by Stanford University
edX: Algorithm Design and Analysis LeetCode: Algorithm Practice Problems
HackerRank: Algorithm Challenges

Algorithm Design and Engineering

Here is a comprehensive Algorithm Design and Engineering Course Syllabus, designed to cover the theoretical and practical aspects of algorithm design and its application in engineering problems.

Module 1: Introduction to Algorithm Design

Definition and Importance of Algorithms in Engineering
Algorithm Design Principles: Correctness, Efficiency, Scalability
Models of Computation: RAM Model, Parallel Models
Complexity Analysis: Time and Space Complexity

References:
Algorithm Design by Jon Kleinberg and Éva Tardos (Chapter 1)
Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein (Chapter 1)
Online: GeeksforGeeks - Algorithm Basics

Module 2: Algorithm Design Paradigms

Divide and Conquer:
Merge Sort, Quick Sort, Binary Search
Strassen's Matrix Multiplication
Greedy Algorithms:
Huffman Coding, Activity Selection, Minimum Spanning Trees
Dynamic Programming:
Longest Common Subsequence, Knapsack Problem
Matrix Chain Multiplication
Backtracking:
N-Queens Problem, Subset Sum, Sudoku Solver

References:
Algorithm Design by Jon Kleinberg and Éva Tardos (Chapters 2–4)
The Algorithm Design Manual by Steven Skiena
Online: MIT OpenCourseWare - Design and Analysis of Algorithms

Module 3: Graph Algorithms

Graph Representations: Adjacency Matrix, Adjacency List
Graph Traversals: BFS, DFS
Shortest Path Algorithms: Dijkstra's, Bellman-Ford, Floyd-Warshall
Minimum Spanning Tree Algorithms: Kruskal's, Prim's
Applications in Engineering: Routing, Scheduling, Network Optimization

References:
Introduction to Graph Theory by Douglas B. West
Graph Algorithms by Shimon Even
Online: GeeksforGeeks - Graph Algorithms

Module 4: Optimization and Approximation Algorithms

Linear Programming and its Applications
Approximation Algorithms for NP-Hard Problems:
Vertex Cover, Traveling Salesman Problem (TSP), Set Cover
Engineering Applications:
Resource Allocation, Job Scheduling

References:
Approximation Algorithms by Vijay Vazirani
Online: Stanford Online - Approximation Algorithms

Module 5: Advanced Topics in Algorithm Design

Randomized Algorithms:
Monte Carlo vs. Las Vegas Algorithms
Applications: Randomized Quick Sort, Min-Cut
Parallel and Distributed Algorithms:
MapReduce Model, Parallel Graph Algorithms
External Memory Algorithms:
B-Trees, Cache-Aware Algorithms

References:
Randomized Algorithms by Motwani and Raghavan
Algorithms in C++ by Robert Sedgewick (Parallel Computing Section)
Online: Khan Academy - Parallel Algorithms

Module 6: Engineering Applications of Algorithms

Computational Geometry:
Convex Hull, Closest Pair of Points, Voronoi Diagrams
Network Flow:
Ford-Fulkerson Algorithm, Max-Flow Min-Cut Theorem
String Matching:
KMP Algorithm, Rabin-Karp Algorithm, Boyer-Moore Algorithm

References:
Computational Geometry: Algorithms and Applications by Mark de Berg et al.
Algorithms on Strings, Trees, and Sequences by Dan Gusfield
Online: Topcoder Tutorials - Advanced Algorithm Applications

Module 7: Algorithm Engineering

Principles of Algorithm Engineering:
Design, Implementation, Testing, and Optimization
Memory Management in Algorithms
Profiling and Performance Optimization
Tools for Algorithm Engineering:
Profilers, Debuggers, Benchmarking Tools

References:
Algorithm Engineering by Michael T. Goodrich
Online: Programming Techniques by Tushar Roy (YouTube)

Module 8: Algorithms for Big Data

Overview of Big Data and its Challenges
Streaming Algorithms:
Count-Min Sketch, Bloom Filters
Graph Processing at Scale:
PageRank, Connected Components
Applications: Social Networks, Search Engines

References:
Mining of Massive Datasets by Anand Rajaraman and Jeffrey Ullman
Online: Coursera - Big Data Algorithms

Module 9: Complexity Theory and NP-Completeness

P, NP, and NP-Complete Problems
Reductions between Problems
Engineering Heuristics for NP-Hard Problems
Real-World Applications in Engineering

References:
Computational Complexity: A Modern Approach by Arora and Barak
Introduction to Algorithms by Cormen, Leiserson (Chapter on NP-Completeness)
Online: MIT OpenCourseWare - Computational Complexity

Online Resources

Coursera: Algorithms Specialization by Stanford University edX: Algorithm Design and Analysis
Khan Academy: Algorithms Course
LeetCode: Practice Problems

Classification of Data Structure

Data structures are broadly classified into two categories: Linear and Non-Linear, based on how data is organized and accessed. Here's a detailed classification:

1. **Linear Data Structures**
   Data elements are arranged sequentially, and each element is connected to its previous and next element.
   Types:

   Arrays:
   Fixed-size, contiguous memory locations.
   Example: [1, 2, 3, 4]

   Linked Lists:
   Sequential collection of nodes, where each node contains data and a reference to next node.
   Types: Singly Linked List, Doubly Linked List, Circular Linked List.

   Stacks:
   LIFO (Last In, First Out) structure.
   Operations: push, pop, peek.
   Example: Undo operations in a text editor.

   Queues:
   FIFO (First In, First Out) structure.
   Variants: Circular Queue, Deque (Double-Ended Queue), Priority Queue.

2. **Non-Linear Data Structures**
   Data elements are not arranged sequentially, forming a hierarchy or network.
   Types:

   Trees:
   Hierarchical structure with a root node and child nodes.
   Types: Binary Tree, Binary Search Tree, AVL Tree, Heap, B-Trees.

   Graphs:
   Set of nodes (vertices) connected by edges.
   Types:
   Based on direction: Directed and Undirected.
   Based on weight: Weighted and Unweighted.
   Applications: Social networks, GPS systems.

3. **Other Classifications**

Static Data Structures:
Size is fixed during compilation.
Example: Arrays.

Dynamic Data Structures:
Size can grow or shrink during runtime.
Example: Linked Lists, Dynamic Arrays.

Linear vs. Hierarchical:
Linear: Arrays, Stacks, Queues.
Hierarchical: Trees, Graphs.

Homogeneous vs. Heterogeneous:
Homogeneous: All elements are of the same type (e.g., Arrays).
Heterogeneous: Elements can be of different types (e.g., Classes, Structures).

Choosing the Right Data Structure

The choice of data structure depends on:
Data size: Small or large datasets.
Operations: Searching, inserting, or deleting.
Speed requirements: Time complexity of operations.
Memory usage: Fixed or dynamic memory allocation.

Classification of Algorithm

Algorithms can be classified based on their purpose, design paradigm, or the type of problem they solve. Here's a structured overview:

1. **Classification Based on Purpose**

1.1 Search Algorithms
Used to find an element in a data structure.
Examples:
Linear Search, Binary Search,
Depth-First Search (DFS), Breadth-First Search (BFS) (for graphs)

1.2 Sorting Algorithms
Arrange elements in a specific order (ascending/descending).
Examples: Bubble Sort, Quick Sort, Merge Sort, Heap Sort, Insertion Sort

1.3 Optimization Algorithms
Find the best solution among many possible options.
Examples: Dynamic Programming, Greedy Algorithms, Genetic Algorithms

1.4 Graph Algorithms
Solve problems related to graph structures.
Examples:
Dijkstra's Algorithm (shortest path)
Kruskal's and Prim's Algorithms (minimum spanning tree)
Floyd-Warshall Algorithm (all-pairs shortest path)

1.5 String Algorithms
Work with string data for pattern matching or manipulation.
Examples:
KMP (Knuth-Morris-Pratt) Algorithm
Rabin-Karp Algorithm
Longest Common Subsequence

1.6 Cryptographic Algorithms
Secure data using encryption or hashing.
Examples:
RSA Algorithm
AES (Advanced Encryption Standard)
SHA (Secure Hash Algorithms)

2. Classification Based on Design Paradigm

2.1 Divide and Conquer
Break a problem into smaller sub-problems, solve them, and combine results.
Examples: Merge Sort, Quick Sort, Binary Search

2.2 Dynamic Programming
Solve problems by breaking them into overlapping sub-problems and storing results.
Examples:
Fibonacci Sequence
Knapsack Problem
Matrix Chain Multiplication

2.3 Greedy Algorithms
Make the locally optimal choice at each step, aiming for a global solution.
Examples:
Huffman Encoding
Kruskal's Algorithm
Activity Selection Problem

2.4 Backtracking
Explore all possibilities by building solutions incrementally and undoing invalid choices.
Examples:
N-Queens Problem
Sudoku Solver
Subset Sum Problem

2.5 Brute Force
Try all possible solutions and pick the best.
Examples:
Travelling Salesman Problem (naive solution)
String Matching

2.6 Randomized Algorithms
Use random numbers to make decisions within the algorithm.
Examples:
Quick Sort (randomized pivot)
Monte Carlo Methods
Randomized Primality Testing

3. Classification Based on Problem Domain

3.1 Numerical Algorithms
Solve mathematical problems.
Examples:
Euclid's Algorithm (GCD)
Newton-Raphson Method (root finding)

3.2 Machine Learning Algorithms
Extract patterns or insights from data.
Examples:
Linear Regression
Decision Trees
Neural Networks

3.3 Parallel Algorithms
Designed to run on multiple processors simultaneously.
Examples:
Parallel Sorting
Matrix Multiplication

4. Classification Based on Complexity

4.1 Time Complexity
Based on execution time.
Examples: O(n),O(log n),O(n2)O(n),O(logn),O(n2)

4.2 Space Complexity
Based on memory usage.
Examples: In-place sorting (low space), recursive algorithms (higher space)

Choosing the Right Algorithm:
Depends on the problem type, data structure used, and required efficiency in time and space.

Application of Algorithm

Algorithms play a crucial role in various domains, enabling efficient problem-solving and optimization across a wide range of applications. Here are some key applications of algorithms in different fields:

1. Computer Science and Software Development

1.1 Sorting and Searching
Example Applications:
Databases: Sorting and indexing for faster queries.
E-commerce: Sorting products by price, rating, or popularity.
Search Engines: Search algorithms like PageRank to rank results.

1.2 Cryptography
Example Applications:
Data Security: RSA and AES for encrypting sensitive information.
Online Transactions: SSL/TLS protocols use cryptographic algorithms for secure communication.
Password Storage: Hashing algorithms like SHA to securely store passwords.

2. Networking

2.1 Routing Algorithms
Example Applications:
Internet Routing: Algorithms like Dijkstra's and Bellman-Ford are used in routers to find the shortest paths in a network.
Traffic Management: Algorithms optimize the routing of data to avoid congestion in computer networks.

2.2 Load Balancing
Example Applications:
Web Servers: Distribute incoming network traffic across multiple servers to ensure optimal performance and prevent overload.

3. Machine Learning and Artificial Intelligence

3.1 Classification and Clustering
Example Applications:
Spam Detection: Naive Bayes or Decision Trees used to classify emails as spam or not.
Customer Segmentation: K-means clustering to group customers based on purchasing behavior.

3.2 Optimization Algorithms

Example Applications:

Recommendation Systems: Algorithms like Collaborative Filtering suggest products based on user preferences.

Training AI Models: Backpropagation and Gradient Descent optimize weights in neural networks.

3.3 Natural Language Processing (NLP)

Example Applications:

Text Processing: Algorithms like the KMP or Rabin-Karp are used in text search and pattern matching.

Speech Recognition: Algorithms enable speech-to-text applications, such as virtual assistants (e.g., Siri, Alexa).

4. Robotics and Control Systems

4.1 Pathfinding Algorithms

Example Applications:

Autonomous Vehicles: Algorithms like A* and Dijkstra's are used for route planning and navigation.

Robotic Arm Movement: Optimization algorithms help to plan the path of robotic arms to avoid obstacles and reach target positions.

4.2 Motion Planning

Example Applications:

Industrial Robots: Algorithms control robots in manufacturing plants to pick up items from one location and place them at another efficiently.

5. Finance and Economics

5.1 Algorithmic Trading

Example Applications:

Stock Market Algorithms: Machine learning and optimization algorithms help in making quick trading decisions based on market data.

5.2 Financial Modeling

Example Applications:

Risk Assessment: Monte Carlo simulations and linear programming are used to predict market movements and calculate risk.

6. Healthcare

6.1 Medical Imaging

Example Applications:

Image Segmentation: Algorithms like K-means clustering or region growing are used to identify and segment regions of interest in medical images (e.g., CT scans, MRIs).

6.2 Disease Prediction

Example Applications:

Predictive Modeling: Machine learning algorithms, such as Logistic Regression or Neural Networks, are used for predicting diseases based on patient data.

7. Social Media and Online Platforms

7.1 Content Recommendation

Example Applications:

Streaming Platforms (e.g., Netflix, YouTube): Collaborative Filtering and Content-Based Filtering algorithms suggest movies and videos based on user preferences.

7.2 Social Network Analysis

Example Applications:

Friend Recommendations: Graph algorithms help to identify potential friends or connections on platforms like Facebook, based on mutual connections.

8. Computer Graphics and Gaming

8.1 Rendering Algorithms

Example Applications:

3D Rendering: Algorithms such as Ray Tracing and Rasterization are used in computer graphics to generate realistic images and animations.

8.2 Game AI

Example Applications:

Pathfinding: A* algorithm is used in video games to control characters' movements through game levels.

Decision Making: Minimax algorithm helps in decision-making for AI in strategy games (chess).

9. Bioinformatics and Computational Biology

9.1 DNA Sequence Alignment

Example Applications:

Gene Sequencing: Algorithms like Needleman-Wunsch and Smith-Waterman are used to compare DNA sequences for gene identification or mutation analysis.

9.2 Protein Structure Prediction

Example Applications:

Molecular Modeling: Algorithms help predict the 3D structure of proteins from their amino acid sequences.

10. Computational Geometry

10.1 Convex Hull Algorithms

Example Applications:

Geographical Information Systems (GIS): Used in map applications to find the boundary of a set of points (e.g., a region or city boundary).

10.2 Intersection Algorithms

Example Applications:

Computer Graphics: Used to detect intersections between objects (e.g., in collision detection for gaming).

Conclusion

Algorithms are fundamental in every area of modern computing, from solving simple problems like sorting and searching to complex tasks like machine learning, cryptography, and robotics. The choice of algorithm directly affects the efficiency, scalability, and success of systems in these domains.

Design Process of Data Structure and Algorithm

The data structure and algorithm design process involves several systematic steps to ensure efficient problem-solving and optimized solutions. Below is an organized guide to this process:

1. **Understand the Problem**
   Clarify Requirements: Understand the inputs, outputs, constraints, and edge cases.
   Ask Questions: Identify ambiguities or incomplete information.
   Examples: Work through examples to grasp the problem dynamics.

2. **Analyze Constraints**
   Time Complexity: Determine how fast the solution should be (e.g., O(log n), O(n), etc.).
   Space Complexity: Evaluate how much memory can be used.
   Special Conditions: Handle edge cases (e.g., empty inputs, large datasets).

3. **Explore Solutions**
   Brute Force: Start with the simplest, most obvious solution.
   Optimize: Think about patterns, redundancies, and unnecessary calculations.
   Choose a Strategy:
   - Divide and conquer
   - Dynamic programming
   - Greedy approach
   - Backtracking
   - Graph traversal (BFS/DFS)

4. **Select Appropriate Data Structures**
   Arrays: For sequential storage and index access.
   Linked Lists: For dynamic, sequential data.
   Stacks/Queues: For LIFO/FIFO operations.
   Hash Tables: For fast key-value lookups.
   Trees/Graphs: For hierarchical or networked data.
   Heaps: For priority-based operations.
   Tries: For efficient string manipulation.

5. **Design the Algorithm**
   Pseudo-code: Write a high-level step-by-step description of the solution.
   Iterative vs Recursive: Decide between iterative or recursive implementation.
   Break into Subproblems: Modularize the problem for easier implementation.

6. **Analyze Complexity**
   Time Complexity:
   Analyze the number of operations relative to input size.
   Use Big-O notation to represent complexity.
   Space Complexity:
   Track memory usage, including auxiliary space.

7. **Implement**
   Start Simple: Write code for a basic version.
   Test Cases: Verify correctness using examples, edge cases, and constraints.
   Debug: Resolve errors incrementally.

8. **Optimize**
   Identify Bottlenecks: Analyze runtime or space issues in implementation.
   Advanced Techniques: Apply memoization, caching, or data structure optimizations.
   Simplify: Reduce unnecessary calculations or redundant operations.

9. **Test Thoroughly**
   Automated Tests: Write unit tests for different input cases.
   Stress Testing: Test with large inputs or extreme edge cases.
   Validation: Confirm performance against constraints.

10. **Document and Refactor**
    Code Readability: Use meaningful variable names and comments.
    Refactor: Simplify or restructure for maintainability.
    Explain: Document the reasoning and performance of the solution.

Tools to Master Data Structures & Algorithms

Books:

Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein.
Algorithms by Robert Sedgewick and Kevin Wayne.
Cracking the Coding Interview by Gayle Laakmann McDowell.

Online Resources:

Platforms: LeetCode, HackerRank, Codeforces, GeeksforGeeks.
Videos: "Abdul Bari Algorithms Playlist" on YouTube, MIT OpenCourseWare.

Practice:

Solve problems daily to strengthen your algorithmic thinking and apply these concepts.