Data Structures and Algorithms – OA25DSA01

Beginner to Expert Course

Course Description

Master data structures and algorithms in this all-in-one course, designed to transform you from a beginner to an expert in algorithmic problem-solving. Covering fundamental data structures, algorithm design, and advanced topics like graph algorithms and dynamic programming, this course uses Python for implementation and is perfect for 2025. The program bundles four levels of learning: Beginner, Intermediate, Advanced, and Expert with Projects.

What You'll Learn

- Understand and implement core data structures like arrays, linked lists, and trees
- Master algorithm design techniques such as recursion, greedy algorithms, and dynamic programming
- Solve real-world coding problems efficiently
- Optimize algorithms for time and space complexity
- Prepare for technical interviews and competitive programming

Course Requirements

- Basic programming knowledge (Python recommended, but concepts are language-agnostic)
- A computer with Python 3.x and a code editor (e.g., VS Code) installed
- Basic understanding of mathematics (e.g., algebra) recommended

Who This Course Is For

- Beginners starting their journey in data structures and algorithms
- Intermediate programmers aiming to master problem-solving and optimization
- Professionals preparing for technical interviews or competitive programming

Syllabus (35 Sessions)

1. Introduction to Data Structures and Algorithms

- Overview of data structures and algorithms
- Setting up Python and VS Code
- Understanding time and space complexity (Big-O notation)

- Writing simple Python scripts

2. Arrays and Lists

- Array basics and operations
- Python lists: Indexing, slicing, and methods
- Common array problems (e.g., find max, reverse)
- Time complexity analysis of array operations

3. Sorting Algorithms (Part 1)

- Introduction to sorting
- Bubble Sort and Selection Sort
- Insertion Sort
- Analyzing sorting complexity

4. Searching Algorithms

- Linear Search
- Binary Search (iterative and recursive)
- Time complexity of search algorithms
- Practical search problems

5. Linked Lists

- Singly linked list: Structure and operations
- Insert, delete, and traverse
- Doubly linked list basics
- Common linked list problems (e.g., reverse list)

6. Stacks and Queues

- Stack: Push, pop, peek
- Queue: Enqueue, dequeue
- Implementing with arrays and linked lists
- Applications (e.g., expression evaluation)

7. Recursion

- Understanding recursion and base cases
- Recursive vs. iterative solutions
- Common recursive problems (e.g., factorial, Fibonacci)
- Recursion tree and complexity analysis

8. Hash Tables

- Introduction to hash tables and hash functions
- Handling collisions: Chaining and open addressing
- Python dictionaries as hash tables
- Solving problems with hash tables

9. Practice Session

- Solve problems using arrays, linked lists, and hash tables
- Congratulations! You've built a foundation in data structures.

10. Trees: Binary Trees

- Binary tree structure and traversal (pre-order, in-order, post-order)
- Implementing binary trees in Python
- Common tree problems (e.g., height, diameter)
- Level-order traversal (BFS)

11. Binary Search Trees (BST)

- BST properties and operations
- Insert, delete, and search
- Balancing issues in BST
- Solving BST problems

12. Heaps

- Min and max heaps
- Heap operations: Insert, extract, heapify
- Implementing priority queues with heaps
- Heap sort and its complexity

13. Sorting Algorithms (Part 2)

- Merge Sort
- Quick Sort
- Comparing sorting algorithms
- Practical sorting applications

14. Graphs: Basics

- Graph representation: Adjacency list and matrix
- Graph traversal: DFS and BFS
- Implementing graphs in Python
- Common graph problems (e.g., connected components)

15. Graph Algorithms

- Shortest path: Dijkstra’s algorithm
- Minimum spanning tree: Kruskal’s and Prim’s algorithms
- Topological sorting
- Solving graph-based problems

16. Practice Session

- Solve tree and graph problems
- Congratulations! You’re an intermediate algorithm developer.

17. Greedy Algorithms

- Introduction to greedy algorithms
- Activity selection problem
- Huffman coding
- Analyzing greedy solutions

18. Dynamic Programming (DP)

- Introduction to DP: Memoization and tabulation
- Classic DP problems: Knapsack, Longest Common Subsequence
- Fibonacci with DP
- Optimizing recursive solutions

19. Advanced Dynamic Programming

- 0/1 Knapsack variations
- Longest Increasing Subsequence
- Matrix chain multiplication
- DP on trees and graphs

20. String Algorithms

- String matching: Naive and KMP algorithm
- Longest Palindromic Substring
- String manipulation with Python
- Solving string-based problems

21. Backtracking

- Introduction to backtracking
- N-Queens problem
- Subset sum and permutations

- Solving combinatorial problems

22. Algorithm Optimization

- Space-time tradeoffs
- Optimizing recursive algorithms
- Using memoization effectively
- Analyzing algorithm performance

23. Bit Manipulation

- Bitwise operators: AND, OR, XOR
- Common bit manipulation problems (e.g., count set bits)
- Bit masking techniques
- Applications in optimization

24. Practice Session

- Solve problems using DP and backtracking
- Congratulations! You've mastered core algorithms.

25. Project 1: Array-Based Analytics

- Build a data analytics tool using arrays
- Implement search and sort algorithms
- Analyze time complexity
- Visualize results with Python

26. Project 2: Linked List Application

- Create a music playlist manager with linked lists
- Implement add, remove, and shuffle operations
- Handle edge cases
- Test performance

27. Project 3: Binary Tree Analyzer

- Build a tree-based data analyzer
- Implement traversals and queries
- Solve problems like lowest common ancestor
- Visualize tree structure

28. Project 4: Graph-Based Route Planner

- Create a route planner using graphs

- Implement Dijkstra's algorithm
- Handle real-world map data
- Optimize for shortest paths

29. Advanced Graph Algorithms

- Bellman-Ford algorithm
- Floyd-Warshall algorithm
- Network flow: Ford-Fulkerson
- Solving advanced graph problems

30. Project 5: Competitive Programming Platform

- Build a problem-solving platform
- Implement algorithms for common problems
- Test with sample inputs
- Optimize solutions

31. Divide and Conquer

- Introduction to divide-and-conquer
- Problems: Merge Sort, Quick Sort, Closest Pair
- Strassen's matrix multiplication
- Analyzing divide-and-conquer algorithms

32. Data Structures for Big Data

- Introduction to tries and suffix trees
- Bloom filters for probabilistic data
- Using Python's collections for efficiency
- Applications in large-scale systems

33. Project 6: Text Search Engine

- Build a search engine using tries
- Implement autocomplete functionality
- Handle large text datasets
- Optimize for performance

34. Interview Preparation

- Solving LeetCode-style problems
- Common interview patterns (e.g., sliding window, two pointers)
- Time and space optimization strategies
- Mock interview practice

35. Final Exam and Wrap-Up

- Comprehensive exam on data structures and algorithms
- Review of key concepts and techniques
- Guidance on career paths (e.g., software engineering, competitive programming)
- Certificate of completion