Week 1: Foundations of Data Structures and Algorithms

Day 1: Revise arrays and strings. Solve 5 problems on arrays from LeetCode or HackerRank.
Day 2: Learn about linked lists. Implement a singly and doubly linked list in C++ or Python.
Day 3: Implement basic operations on stacks and queues (push, pop, enqueue, dequeue).
Day 4: Solve 5 stack/queue problems from LeetCode or HackerRank.
Day 5: Study recursion and write recursive functions (factorial, Fibonacci, etc.).
Day 6: Solve 3 problems involving recursion (e.g., generating subsets, permutations).
Day 7: Review the week's progress, reattempt any unsolved problems.

Week 2: Advanced Data Structures and Sorting

Day 8: Learn binary search trees (BST). Implement a basic BST with insert, delete, and search operations.
Day 9: Learn heaps. Implement a max-heap and a min-heap.
Day 10: Study sorting algorithms: implement selection sort, bubble sort, and insertion sort.
Day 11: Implement merge sort and quicksort.
Day 12: Solve 5 problems on sorting and binary search from LeetCode or HackerRank.
Day 13: Study hash maps and sets. Implement them using arrays or linked lists.
Day 14: Solve problems on hash maps and sets (e.g., two-sum, anagram detection).

Week 3: Algorithms and Problem Solving

Day 15: Study dynamic programming (DP). Start with the Fibonacci sequence and a basic knapsack problem.
Day 16: Solve 3 easy DP problems (e.g., climbing stairs, house robber).
Day 17: Study graph theory: BFS and DFS. Implement both in Python or C++.
Day 18: Solve 3 graph traversal problems (e.g., connected components, cycle detection).
Day 19: Study greedy algorithms. Solve problems like the activity selection problem.
Day 20: Learn about backtracking. Implement N-Queens and solve 2 backtracking problems.
Day 21: Review the week, retry harder problems, and refine your solutions.

Week 4: Object-Oriented Programming (OOP)

Day 22: Study OOP principles: encapsulation, inheritance, and polymorphism. Write examples in C++ or Java.
Day 23: Implement a simple class hierarchy (e.g., vehicles or animals) and use inheritance.
Day 24: Learn about design patterns: implement the Singleton and Factory patterns.
Day 25: Learn about the Observer and Strategy patterns, and implement them.
Day 26: Solve a small project using OOP principles (e.g., a library system or a basic game).
Day 27: Study exceptions and error handling. Implement robust error handling in a small program.
Day 28: Review and improve your OOP code and project from this week.

Week 5: Databases and SQL

Day 29: Study relational databases and basic SQL. Learn how to create tables and insert data.
Day 30: Write SQL queries for SELECT, JOIN, and WHERE clauses using sample datasets.
Day 31: Study database normalization and design a normalized schema for a sample application.
Day 32: Learn about transactions and ACID properties. Write queries demonstrating transaction management.
Day 33: Implement a small project with a database (e.g., a todo app with a SQLite or PostgreSQL backend).
Day 34: Learn about indexing in databases. Practice optimizing SQL queries.
Day 35: Review and expand the project, adding more features or optimizing performance.

Week 6: Web Development and APIs

Day 36: Study the basics of web development: HTTP, REST, and how the web works.
Day 37: Learn about Flask or Express.js. Create a basic RESTful API.
Day 38: Implement routes for GET, POST, PUT, and DELETE requests in your API.
Day 39: Study authentication methods (e.g., JWT, OAuth). Implement basic authentication in your API.
Day 40: Add error handling and validation to your API.
Day 41: Deploy your API using a platform like Heroku or AWS.
Day 42: Review the week’s work and optimize or add new features to your API.

Week 7: Software Design and System Design

Day 43: Study software design principles: SOLID and DRY. Apply them in a small refactoring project.
Day 44: Learn about REST vs. GraphQL. Understand the pros and cons of both.
Day 45: Study microservices architecture. Understand the key concepts and benefits.
Day 46: Study caching mechanisms (e.g., Redis, memcached). Implement basic caching in an API.
Day 47: Study and implement rate limiting and throttling in a web service.
Day 48: Read about system design fundamentals (load balancing, scaling, fault tolerance).
Day 49: Review and solidify your knowledge with a small system design exercise (e.g., design a URL shortener).

Week 8: AI and Machine Learning Fundamentals

Day 50: Study the basics of machine learning: supervised vs. unsupervised learning.
Day 51: Learn about regression models. Implement a simple linear regression model in Python.
Day 52: Study classification algorithms (logistic regression, decision trees). Implement one in Python.
Day 53: Study neural networks. Implement a simple feedforward network using PyTorch or TensorFlow.
Day 54: Solve a small ML problem (e.g., predict house prices using a regression model).
Day 55: Learn about overfitting and regularization techniques. Apply them to your model.
Day 56: Train a neural network on a small dataset (e.g., MNIST) and improve accuracy.

Week 9: Advanced Topics and Final Projects

Day 57: Study concurrency. Implement a multithreaded application using C++ or Python.
Day 58: Review graph algorithms (shortest path, minimum spanning tree) and solve a few problems.
Day 59: Work on a final project combining multiple areas (e.g., a web app with a backend, database, and API).
Day 60: Finalize and polish your project. Ensure good code quality and documentation.

Tips for Success:

1. Code Daily: Even if the tasks seem simple, aim to code every day for consistent improvement.
2. Leverage Online Resources: Use platforms like LeetCode, Codeforces, HackerRank, and Project Euler for algorithm problems.
3. Collaborate: Discuss problems or pair-program with peers to deepen your understanding.
4. Refactor: Always revisit your code and try to improve it after you've completed it. Clean code is important.
5. Build Projects: Projects help you apply your knowledge in a real-world context and give you tangible results to showcase.