Maurice's 12-Month Deep CS Learning Plan

Goal: Deep understanding of computer science fundamentals with career pivot readiness

Time Commitment: 5-10 hours per week (260-520 total hours)

Approach: 80% CS Fundamentals + 20% LeetCode Application

Philosophy: Learn deeply, practice immediately, build confidence

---

Your Profile & Strengths

Advantages to Leverage:

- ✅ Already can code (miniCycle proves this)
- ✅ Quality inspection background = systematic, detail-oriented thinking
- ✅ Blueprint reading = pattern recognition skills
- ✅ Associate's in Computer Engineering = technical foundation
- ✅ 12-month timeline = can learn correctly the first time
- ✅ Want deep understanding = will become a strong engineer

Weekly Time Breakdown (7.5 hrs average):

- 📚 **Concept Learning:** 3 hours (reading, videos, notes)
- 💻 **Implementation:** 2.5 hours (code concepts from scratch)
- 🧩 **Problem Practice:** 2 hours (LeetCode application)

Program Structure

```
Months 1-3:  Foundations (Data Structures + Basic Algorithms)
Months 4-6:  Intermediate (Trees, Graphs, Recursion)
Months 7-9:  Advanced (Dynamic Programming, System Design)
Months 10-12: Mastery (Hard Problems, Interview Prep, Specialization)
```

Phase 1: Foundations (Months 1-3)

Month 1: Arrays, Strings, Hash Tables & Complexity

Week 1-2: Arrays & Complexity Analysis

📚 **Learn:**

- How arrays work in memory (contiguous memory, indexing = O(1))
- Big O notation (time and space complexity)
- Best/average/worst case analysis
- Common array operations and their costs

💻 **Implement:**

- Dynamic array (ArrayList) from scratch
- Measure actual performance differences
- Visualize memory layout

🧩 **Practice (10 problems):**

- Two Sum
- Best Time to Buy/Sell Stock
- Contains Duplicate
- Product of Array Except Self
- Maximum Subarray
- Rotate Array
- *Focus: Understanding why solutions work, not memorizing*

Week 3-4: Hash Tables & Strings

📚 **Learn:**

- Hash function concepts
- Collision handling (chaining vs open addressing)
- Why O(1) lookup (amortized)
- String manipulation complexity

💻 **Implement:**

- Build a hash table from scratch

- Implement different collision strategies
- String matching algorithms (naive first)

🧩 **Practice (10 problems):**

- Valid Anagram
- Group Anagrams
- Longest Substring Without Repeating Characters
- Valid Parentheses
- Longest Palindromic Substring
- *Focus: When to use hash tables vs arrays*

📝 **Month 1 Checkpoint:**

□ Can explain Big O in your own words
□ Can build an array and hash table from scratch
□ Understand *why* hash tables are fast

Month 2: Linked Lists, Stacks, Queues

Week 1-2: Linked Lists

📚 **Learn:**

- Singly vs doubly linked lists
- Why linked lists exist (insertion/deletion advantages)
- Trade-offs vs arrays
- Pointer manipulation patterns

💻 **Implement:**

- Singly linked list (insert, delete, reverse)
- Doubly linked list
- Circular linked list
- *Debug pointer issues - this is crucial*

🧩 **Practice (12 problems):**

- Reverse Linked List
- Merge Two Sorted Lists
- Remove Nth Node From End
- Detect Cycle
- Add Two Numbers
- Copy List with Random Pointer
- *Focus: Drawing diagrams before coding*

Week 3-4: Stacks & Queues

📚 **Learn:**

- Stack: LIFO, when to use (undo, parsing, DFS)
- Queue: FIFO, when to use (BFS, scheduling)
- Deque (double-ended queue)
- Priority queues introduction

💻 **Implement:**

- Stack using array and linked list
- Queue using array and linked list
- Circular queue
- Min/Max stack (O(1) min operation)

🧩 **Practice (10 problems):**

- Valid Parentheses (stack application)
- Min Stack
- Evaluate Reverse Polish Notation
- Implement Queue using Stacks
- Sliding Window Maximum (deque)

📝 **Month 2 Checkpoint:**

□ Can code linked list operations without looking
□ Instinctively know when to use stack vs queue

□ Can draw pointer diagrams to debug

Month 3: Sorting, Searching & Two Pointers

Week 1-2: Sorting Algorithms

📚 **Learn:**

- Comparison sorts: Bubble, Selection, Insertion
- Efficient sorts: Merge Sort, Quick Sort
- Non-comparison: Counting Sort, Radix Sort
- When to use each, stability, in-place vs not

💻 **Implement:**

- Code each algorithm from scratch
- Visualize how they work (animate if possible)
- Compare actual performance on different inputs
- Understand the recursion in merge/quick sort

🧩 **Practice (8 problems):**

- Sort Colors (counting sort variation)
- Merge Intervals
- Kth Largest Element
- Sort List (linked list merge sort)

Week 3-4: Binary Search & Two Pointers

📚 **Learn:**

- Binary search template and variations
- Search space reduction concept
- Two pointers: opposite ends, same direction
- Sliding window technique

💻 **Implement:**

- Binary search (iterative and recursive)
- Search in rotated array
- Two pointer templates for different problems

🧩 **Practice (12 problems):**

- Binary Search
- Search in Rotated Sorted Array
- Container With Most Water
- 3Sum
- Minimum Window Substring
- *Focus: When binary search applies*

📝 **Quarter 1 Review (End of Month 3):**

□ Code a sorting algorithm from memory
□ Solve a medium problem using each data structure
□ Explain trade-offs between different approaches
□ **Mini-project:** Build something using what you learned

Phase 2: Intermediate Concepts (Months 4-6)

Month 4: Recursion & Backtracking

Week 1-2: Recursion Fundamentals

📚 **Learn:**

- Recursive thinking (base case + recursive case)
- Call stack visualization
- Tail recursion optimization
- When recursion vs iteration

💻 **Implement:**

- Classic problems: Fibonacci, factorial
- More complex: Tower of Hanoi

- String permutations
- Visualize call stack for each

🧩 **Practice (10 problems):**

- Climbing Stairs
- Pow(x,n)
- Generate Parentheses
- Letter Combinations
- Subsets
- Permutations

Week 3-4: Backtracking

📚 **Learn:**

- Backtracking template (choice, explore, unchoose)
- Pruning search space
- State management in backtracking

💻 **Implement:**

- N-Queens visualization
- Sudoku solver
- Word search

🧩 **Practice (10 problems):**

- Combination Sum
- Palindrome Partitioning
- Word Search
- N-Queens
- *Focus: Drawing decision trees*

Month 5: Trees (Binary Trees & BST)

Week 1-2: Binary Trees

📚 **Learn:**

- Tree terminology (root, leaf, height, depth)
- Tree traversals: inorder, preorder, postorder, level-order
- Recursive vs iterative traversals
- Properties: balanced, complete, perfect

💻 **Implement:**

- Binary tree from scratch
- All traversal methods (recursive + iterative)
- Tree visualization
- Common operations: height, count nodes, etc.

🧩 **Practice (12 problems):**

- Invert Binary Tree
- Maximum Depth
- Same Tree
- Symmetric Tree
- Binary Tree Level Order Traversal
- Validate Binary Search Tree

Week 3-4: Binary Search Trees

📚 **Learn:**

- BST property and why it matters
- BST operations: insert, delete, search
- BST vs hash table trade-offs
- Balanced BSTs introduction (AVL, Red-Black concepts)

💻 **Implement:**

- BST from scratch with all operations
- Understand BST deletion (hardest operation)
- Iterator for BST

🧩 **Practice (10 problems):**

- Kth Smallest Element in BST
- Lowest Common Ancestor
- Convert Sorted Array to BST
- Serialize and Deserialize BST

Month 6: Graphs (The Foundation)

Week 1-2: Graph Representations & BFS

📚 **Learn:**

- Graph terminology (vertices, edges, directed/undirected)
- Adjacency matrix vs adjacency list
- When to use graphs
- BFS algorithm and applications

💻 **Implement:**

- Graph class with both representations
- BFS from scratch
- Shortest path in unweighted graph
- Visualize BFS exploration

🧩 **Practice (10 problems):**

- Number of Islands
- Clone Graph
- Course Schedule
- Minimum Knight Moves

- Word Ladder
- Rotting Oranges

Week 3-4: DFS & Graph Applications

📚 **Learn:**

- DFS algorithm (recursive and iterative)
- Cycle detection
- Topological sort
- Connected components

💻 **Implement:**

- DFS from scratch
- Cycle detection in directed/undirected graphs
- Topological sort

🧩 **Practice (10 problems):**

- Course Schedule II
- Pacific Atlantic Water Flow
- Number of Connected Components
- Graph Valid Tree
- Redundant Connection

📝 **Quarter 2 Review:**

□ Solve tree/graph problems without hints
□ Explain BFS vs DFS trade-offs
□ Build a small project using trees or graphs

Phase 3: Advanced Topics (Months 7-9)

Month 7-8: Dynamic Programming

This is the big one - take your time here

Month 7: DP Foundations

📚 **Learn:**

- What is DP (overlapping subproblems + optimal substructure)
- Memoization vs tabulation
- How to identify DP problems
- 1D DP patterns

💻 **Implement:**

- Fibonacci with memoization and tabulation
- Visualize DP table construction
- Classic problems: coin change, house robber

🧩 **Practice (15 problems over 4 weeks):**

- Climbing Stairs
- House Robber
- Coin Change
- Longest Increasing Subsequence
- Word Break
- Decode Ways
- *Focus: Recognize the pattern*

Month 8: 2D DP & Advanced Patterns

📚 **Learn:**

- 2D DP problems
- Grid-based DP
- String DP patterns
- State machine DP

💻 **Implement:**

- Edit distance visualization

- Longest common subsequence
- Matrix chain multiplication

🧩 **Practice (15 problems):**

- Unique Paths
- Minimum Path Sum
- Edit Distance
- Longest Common Subsequence
- Best Time to Buy/Sell Stock (all variations)
- Regular Expression Matching

Month 9: Heaps, Tries & Advanced Structures

Week 1-2: Heaps/Priority Queues

📚 **Learn:**

- Heap property (min-heap, max-heap)
- Heapify operations
- When to use heaps
- Heap sort

💻 **Implement:**

- Min-heap from scratch
- Heap operations (insert, extract, heapify)
- Priority queue using heap

🧩 **Practice (8 problems):**

- Kth Largest Element
- Top K Frequent Elements
- Merge K Sorted Lists
- Find Median from Data Stream

Week 3-4: Tries & Specialized Structures

📚 **Learn:**

- Trie structure and applications
- Union-Find (Disjoint Set)
- Segment trees (introduction)

💻 **Implement:**

- Trie from scratch
- Union-Find with path compression

🧩 **Practice (8 problems):**

- Implement Trie
- Word Search II
- Number of Islands II (Union-Find)
- Range Sum Query (Segment Tree intro)

📝 **Quarter 3 Review:**

☐ Solve a hard DP problem
☐ Implement any advanced data structure from memory
☐ **Project:** Build something that uses DP or advanced structures

---

Phase 4: Mastery & Specialization (Months 10-12)

Month 10: System Design & Problem-Solving

📚 **Learn:**

- How to approach unknown problems
- Trade-off analysis
- Basic system design concepts
- Space/time optimization techniques

💻 **Build:**

- Design and implement a cache (LRU)
- Design a rate limiter
- Design a simple database index

🧩 **Practice:**

- Mix of all previous topics
- Focus on medium/hard problems
- Explain solutions out loud

Month 11: Interview Preparation

🎯 **Mock Interviews:**

- Timed problem-solving (45 min sessions)
- Explain thought process
- Handle follow-up questions

📊 **Pattern Recognition:**

- Review all major patterns
- Create your own pattern cheat sheet
- Identify which pattern applies quickly

🧩 **Practice:**

- 20-30 medium/hard problems
- Focus on explaining, not just solving
- Time yourself

Month 12: Specialization & Portfolio

Choose your path:

- **Path A:** Deep dive into algorithms you love (graphs, DP, etc.)
- **Path B:** Apply CS to a project (rebuild miniCycle with better algorithms)

- **Path C:** Competitive programming for fun
- **Path D:** Start interviewing for real

Resources

For Deep Understanding

📘 **Books:**

- "Grokking Algorithms" by Bhargava (visual, beginner-friendly)
- "Introduction to Algorithms" (CLRS) - reference, not cover-to-cover
- "Algorithm Design Manual" by Skiena (practical)

🎥 **Video:**

- MIT OpenCourseWare: Introduction to Algorithms
- Abdul Bari's Algorithm Playlist (YouTube) - fantastic explanations
- Back To Back SWE (YouTube) - problem walkthroughs

💻 **Practice:**

- LeetCode (obviously)
- Visualgo.net (algorithm visualizations)
- AlgoExpert (if you want structured curriculum)

Weekly Routine Template

Monday (2 hours): Concept Learning

- Watch video / read chapter
- Take notes in your own words
- Draw diagrams

Wednesday (2.5 hours): Implementation

- Code the concept from scratch
- No looking at solutions until you try

- Debug and understand every line

Friday (1.5 hours): Easy Problems

- Apply what you learned
- 2-3 problems maximum
- Focus on understanding, not speed

Saturday/Sunday (2 hours): Harder Problems

- Challenge yourself
- It's okay to look at hints
- Study solutions you don't get

Success Metrics

By Month 3:

- ✅ Solve 50+ easy problems
- ✅ Implement basic data structures from memory
- ✅ Explain Big O confidently

By Month 6:

- ✅ Solve 100+ easy, 30+ medium problems
- ✅ Comfortable with trees and graphs
- ✅ Understand when to use each structure

By Month 9:

- ✅ Solve 50+ medium, 10+ hard problems
- ✅ Can recognize DP patterns
- ✅ Build complex structures from scratch

By Month 12:

- ✅ 150+ problems total (100 medium, 20 hard)

- ✅ Pass mock interviews
- ✅ Deep CS understanding
- ✅ Ready to interview OR confident building anything

Leverage Your Background

Quality inspection → Algorithm correctness:

You're used to verifying things work exactly right

Blueprint reading → Drawing solutions:

Always diagram before coding

CMM/precision tools → Big O analysis:

You understand measurement precision

miniCycle builder → Real applications:

Connect every concept to how it could improve your app

First Steps

1. **Assess:** What CS concepts do you remember from your Associate's degree?
2. **Set up:** Create a GitHub repo for your learning journey - commit implementations and solutions
3. **Start:** Begin with Month 1, Week 1 - Arrays & Big O
4. **Track:** Keep a learning journal noting what clicks and what struggles

---

Remember: This plan is your roadmap, but we'll adjust based on how you learn. Some topics might take longer, others might go faster. The goal is deep understanding, not rushing through content.

Let's build your CS foundation! 🚀