📘 Module A1: Mathematics for DSA (Advanced)

- Recurrence Relations
- Master Theorem
- Backtracking Trees
- Bit Manipulation
- Combinatorics for Algorithm Design
- Modular arithmetic (useful for hashing/cryptography)

📘 Module A2: Advanced Strings & Pattern Matching

- KMP Algorithm
- Rabin-Karp Algorithm
- Z Algorithm
- Boyer-Moore Algorithm
- Trie
- Suffix Array
- Longest Prefix Suffix (LPS array)

📘 Module A3: Advanced Linked List Problems

- Floyd Cycle Detection
- Remove loops
- Merge K sorted lists
- Deep copy linked lists
- Linked list segment reversal

📘 Module A4: Advanced Graph Algorithms

- Topological Sorting
- Strongly Connected Components (Kosaraju / Tarjan)
- Bridges & Articulation Points
- Cycle Detection (Directed/Undirected)
- Bipartite Graph
- Minimum Spanning Tree (Kruskal / Prim)
- DSU in Graphs
- Multi-source BFS

📘 Module A5: Dynamic Programming

(Most important topic for placements)

- Memoization & Tabulation
- DP on sequences:
  - Longest Increasing Subsequence (LIS)
  - Longest Common Subsequence (LCS)
- DP on grids
- Knapsack (0/1, Unbounded)
- DP on strings
- Palindromic Subsequence/Substring
- Coin change problems
- DP on trees
- Bitmask DP (intro level)

📘 Module A6: Greedy Algorithms (Interview Essentials)

- Activity selection
- Fractional knapsack
- Job sequencing
- Gas station problem
- Interval scheduling/merging
- Huffman coding (detailed)

📘 Module A7: Advanced Tree Data Structures

- AVL Tree (Deep dive)
- Red-Black Tree (Detailed)
- Splay Trees
- Segment Tree (Range Query, Lazy Propagation)
- Fenwick Tree (Binary Indexed Tree)
- Trie (Full Implementation)

📘 Module A8: Advanced Hashing

- Perfect hashing
- Double hashing deeply
- Bloom Filters (Modern industry topic)
- Rolling Hash (for string matching)

📘 Module A9: Heaps & Priority Queues (Advanced)

- Merge K sorted arrays
- K largest/smallest elements
- Sliding window maximum using heap
- Min-cost problems
- Implementing custom comparator heaps

📘 Module A10: Advanced Problem-Solving Techniques

- Sliding window advanced
- Two-pointer advanced
- Binary search on answer
- Subarray/subsequence optimizations
- Prefix-sum advanced
- Recursion tree analysis

📘 **Module A11: Competitive Programming Essentials**

- Efficient I/O
- Debugging techniques
- Time limit optimization
- Memory-efficient coding
- Common templates

📘 **Module A12: LeetCode / FAANG Interview Patterns**

- 12 Major problem patterns:
  - Two pointers
  - Fast & slow pointer
  - Sliding window
  - Greedy
  - Backtracking
  - Graph BFS/DFS
  - DP on arrays
  - DP on strings
  - Binary search patterns
  - Linked list patterns
  - Trees/Tries patterns
  - HashMap/Set patterns

📘 **Module A13: Real-World Advanced Applications**

- Real-time route planning (graphs)
- Caching systems (LRU cache)
- Auto-complete (Trie)
- Load balancing (hashing)
- Search engines (suffix arrays, tries)
- Compilers & interpreters (stack + tree)

🔧 **Implementation Languages for Advanced DSA**

For the **Advanced Data Structures & Algorithms** course, we focus on **problem-solving and interviews**, not just syntax. Students can choose **one primary language** from the options below:

🥇 **Option 1: C++ (Recommended for Advanced DSA)**

Why C++?

- ✅ Fastest performance – ideal for complex algorithms & large inputs
- ✅ Powerful STL (Standard Template Library): vector, set, map, priority_queue, etc.
- ✅ Standard language for **competitive programming** (Codeforces, LeetCode, ICPC)
- ✅ Preferred in **product-based companies** & FAANG for DSA rounds

Best For:

- Students targeting **Google, Amazon, Microsoft, etc.**
- Students interested in **competitive programming**
- Those who want to build **strong low-level fundamentals**

🥈 **Option 2: Java (Job-Oriented + Backend-Friendly)**

Why Java?

- ✅ Strong Collections Framework (ArrayList, HashMap, TreeMap, PriorityQueue, etc.)
- ✅ Widely used in **enterprise/backend development** (Spring, Microservices)
- ✅ Many companies conduct DSA rounds in **Java**

Best For:

- Students planning for **Java + Spring Boot** backend careers

---

🥉 **Option 3: Python (Simple & Interview-Friendly)**

Why Python?

- ✅ Very simple syntax – students can focus on logic, not boilerplate
- ✅ Great for **AI/ML/Data Science + DSA** combination
- ✅ Excellent built-in structures: list, dict, set, heapq, etc.

Best For:

- Students interested in **Data Science, Machine Learning**
