# 📚 COMPLETE_SYLLABUS_v10_FINAL.md — 19-Week Mental-Model-First DSA Curriculum

**Version:** 10.0 (Reorganized 19-week structure, original v9.2 topics preserved)
**Status:** ☑ OFFICIAL FINAL SYLLABUS
**Philosophy:** Mental-model-first, systems-focused, pattern-centric, understanding before code.

This syllabus is structured to be:

- **Human-friendly:** clear weekly goals, rationale, and day-wise topics
- **AI-friendly:** consistent Markdown headings and bullet formatting

Each week includes:

- **Primary Goal** — what you should internalize
- **Why This Week Comes Here** — rationale in the larger sequence
- **Day-by-Day Topics** — specific concepts for each day

---

## 🗺 Phase Overview

- **Phase A — Foundations (Weeks 1–3)**
- **Phase B — Core Patterns & Strings I (Weeks 4–6)**
- **Phase C — Trees, Graphs & Advanced DS (Weeks 7–11)**
- **Phase D — Algorithm Paradigms (Weeks 12–13)**
- **Phase E — Pattern Integration & Extensions (Weeks 14–15)**
- **Phase F — Advanced Deep Dives (Weeks 16–18)**
- **Phase G — Mock Interviews & Final Review (Week 19)**

---

# ▧ PHASE A — FOUNDATIONS (Weeks 1–3)

---

## ▧ Week 1 — Foundations I: Computational Fundamentals

**Primary Goal:**
Understand the RAM model, memory layout, time/space complexity, and recursion mechanics.

**Why This Week Comes Here:**
Everything later—Big-O, data structures, patterns—rests on these mental models. Without them, complexity and performance feel arbitrary.

### Day 1: RAM Model & Pointers

- RAM model: memory as an array of cells; constant-time access assumption.
- Process address space: code, globals, heap, stack; what lives in each region.
- Pointers/references: variables that store addresses; dereferencing as “follow this arrow”.

- Virtual memory: pages, page tables, TLB; conceptual mapping to physical memory.
- Caches (L1/L2/L3): why contiguous access is faster than random access.

---

### Day 2: Asymptotic Analysis (Big-O, Big-Ω, Big-Θ)

- Motivation for asymptotics: scaling behavior vs constant factors.
- Definitions and intuition:
  - Big-O: upper bound
  - Big-Ω: lower bound
  - Big-Θ: tight bound
- Common complexity classes: O(1), O(log n), O(n), O(n log n), O($n^2$), O($2^n$), O(n!).
- Simple recurrence reasoning (e.g., binary search, merge sort at a high level).

---

### Day 3: Space Complexity & Memory Usage

- Total vs auxiliary space; why both matter.
- Stack vs heap usage:
  - Local variables vs dynamic allocations
  - Lifetime and scope
- Overheads: pointers, object headers, fragmentation (conceptually).
- Trade-offs: caching/precomputation vs memory footprint.

---

### Day 4: Recursion I — Call Stack & Basic Patterns

- Call stack: frames, parameters, locals, return addresses; visualizing call chains.
- Base case and recursive case; avoiding infinite recursion.
- Recursion tree visualization for simple problems (factorial, sum, Fibonacci).
- Mapping recursion to repeated subproblems and branching factor.

---

### Day 5: Recursion II — Advanced Patterns & Memoization Intro

- Tail recursion vs general recursion (conceptual, not compiler details).
- Mutual recursion and indirect recursion examples.
- Memoization: caching results of expensive recursive calls; concept bridge to DP.
- When recursion is natural vs when iteration or explicit stack is preferable.

---

## ▧ Week 2 — Foundations II: Linear Data Structures

**Primary Goal:**
Develop solid mental models for arrays, dynamic arrays, linked lists, stacks, and queues, and how they behave in memory.

**Why This Week Comes Here:**
These structures are the basis for many patterns and higher-level structures (trees, heaps, graphs).

### Day 1: Arrays

- Static arrays: contiguous memory, index→address mapping.
- Advantages: locality, fast sequential access, cheap random access.
- Disadvantages: fixed size, expensive insert/delete in middle.
- Multi-dimensional and jagged arrays: row-major vs column-major intuition.

---

### Day 2: Dynamic Arrays

- Dynamic arrays as resizable arrays; capacity vs size.
- Doubling strategy for amortized O(1) append.
- Reallocation: when and why it happens; effect on pointers.
- Trade-offs vs linked lists for growing/shrinking collections.

---

### Day 3: Linked Lists

- Singly and doubly linked list node structure.
- Pointer-based connections; visualizing lists as chains of nodes in heap.
- Operations: insert/delete at head, tail, arbitrary position; cost analysis.
- Pros: O(1) insert/delete at known position; cons: poor locality, no O(1) random access.

---

### Day 4: Stacks & Queues

- Stack (LIFO) concept:
  - Call stack model
  - Applications: parsing, undo/redo, DFS.
- Queue (FIFO) concept:
  - Buffering, BFS, scheduling tasks.
- Array-based vs list-based implementations.
- Circular buffer/ring queue; reducing wasteful movement.

---

### Day 5: Binary Search

- Binary search invariant: search range defined by low/high.
- Mechanics of halving search space; iterative vs recursive mental model.
- Edge cases: mid calculation, infinite loops, off-by-one errors.
- Variants: first occurrence, last occurrence, lower_bound, upper_bound.

---

## ▧ Week 3 — Foundations III: Sorting & Hashing

**Primary Goal:**
Understand how fundamental sorting and hashing techniques work internally and what trade-offs they involve.

**Why This Week Comes Here:**
Sorting and hashing are primitive operations used everywhere; they also illustrate algorithm design and complexity in practice.

### Day 1: Elementary Sorts — Bubble, Selection, Insertion

- Mechanics of each:
    - Bubble: repeated swaps of adjacent out-of-order pairs.
    - Selection: repeatedly selecting minimum/maximum, placing in correct position.
    - Insertion: insert each element into sorted prefix.
- Complexity: $O(n^2)$ worst/average; when they are acceptable (small n, nearly sorted).
- Stability and in-place nature; uses in hybrid algorithms (like Timsort).

---

### Day 2: Merge Sort & Quick Sort

- Merge sort:
    - Divide array into halves, sort, merge.
    - Stable, O(n log n) always; memory cost for merges.
- Quick sort:
    - Partitioning around a pivot; average O(n log n), worst $O(n^2)$.
    - Partition schemes (Lomuto, Hoare) and their behaviors.
- When to choose which in practice; hybrid strategies.

---

### Day 3: Heap Sort & Priority Queues

- Binary heap representation (implicit tree in array).
- Operations: heapify, insert, extract-min/max.
- Heap sort: build heap, repeatedly extract.
- Priority queues: scheduling tasks, event simulation, Dijkstra’s algorithm.

---

### Day 4: Hash Tables I — Separate Chaining

- Hash function intuition: mapping keys to indices.
- Buckets with chains (linked lists or small arrays).
- Load factor and resizing; effect on performance.
- Average vs worst-case complexities; mitigation via good hashing.

---

### Day 5: Hash Tables II — Open Addressing & Advanced Hashing

- Open addressing: linear, quadratic probing, double hashing.
- Clustering and strategies to reduce it.
- Cuckoo hashing and Robin Hood hashing (high-level idea).
- Perfect hashing and universal hashing concepts.

---

# ▨ PHASE B — CORE PATTERNS & STRINGS I (Weeks 4–6)

---

## ▨ Week 4 — Core Problem-Solving Patterns I

**Primary Goal:**
Acquire the foundational **array/sequence patterns** (two pointers, sliding window, divide & conquer, binary search as a pattern) that drastically simplify many problems.

**Why This Week Comes Here:**
You now know how arrays work; patterns are reusable “mental templates” to solve families of problems efficiently.

### Day 1: Two Pointers

- Same-direction two pointers:
  - Merging sorted arrays
  - Removing duplicates in-place.
- Opposite-direction pointers:
  - Container with most water
  - Two-sum in sorted arrays.
- When pointers move and why; maintaining invariants to avoid missing solutions.

---

### Day 2: Sliding Window (Fixed Size)

- Fixed-length window pattern:
  - Moving start/end by one each time.
- Typical tasks:
  - Average/sum of subarrays of size k
  - Maximum/minimum in sliding windows (with auxiliary DS).
- Handling edges: first window, last window.

---

### Day 3: Sliding Window (Variable Size)

- Windows that grow and shrink based on constraints:
  - “At most K distinct”, “sum ≤ target”, etc.
- Recognizing when to shrink window vs expand window.
- Maintaining counts or frequency maps while sliding.

---

### Day 4: Divide and Conquer Pattern

- General structure: solve subproblems and combine results.
- Applied to:
  - Merge sort

    - Counting inversions
    - Majority element variants.
- Reasoning about complexity via recursion trees.

---

### Day 5: Binary Search as a Pattern

- Binary search on sorted arrays vs on abstract “answer space”.
- Using feasibility checks:
    - “Can we achieve X with these constraints?”
- Example patterns:
    - Minimizing maximum load, maximizing minimum distance.
- Designing checks and boundaries carefully.

---

## ▨ Week 5 — Tier 1 Critical Patterns

**Primary Goal:**
Master a set of **high-frequency patterns** (hash, monotonic stack, intervals, cyclic sort, Kadane, fast/slow) that cover a large part of interview problem space.

**Why This Week Comes Here:**
This builds directly on Week 4 patterns, adding powerful new tools for arrays, lists, and intervals.

### Day 1: Hash Map / Hash Set Patterns

- Hash-based two-sum/k-sum patterns.
- Frequency counting for:
    - Anagrams
    - Most/least frequent elements.
- Maps for complement tracking, pair matching, and membership tests.

---

### Day 2: Monotonic Stack

- Increasing/decreasing stacks for:
    - Next greater/smaller element
    - Stock span problems.
- Using stacks to “remember” promising candidates only.
- Applications to histograms, water trapping (conceptual).

---

### Day 3: Merge Operations & Interval Patterns

- Merging sorted arrays/lists using two pointers.
- Interval merging:
    - Sorting intervals first
    - Combining overlapping ranges.
- Insert interval, merging multiple intervals into disjoint set.

---

### Day 4: Partition & Cyclic Sort + Kadane's Algorithm

- Dutch National Flag: partition into three categories (0/1/2, negative/0/positive).
- Cyclic sort:
  - Arrays containing numbers 1...n; placing each at correct index.
- Kadane's algorithm:
  - Interpreting maximum subarray problem as best running sum.

---

### Day 5: Fast & Slow Pointers

- Floyd's cycle-finding algorithm:
  - Detect cycle existence
  - Find cycle start.
- Finding middle of a list; splitting list into halves.
- Applying to number problems (e.g., happy number).

---

## ▨ Week 6 — Tier 1.5 String Manipulation Patterns

**Primary Goal:**
Learn practical string patterns for palindromes, substrings, parentheses, and transformations.

**Why This Week Comes Here:**
Strings are just arrays of characters; now you adapt your earlier patterns to text problems.

### Day 1: Palindrome Patterns

- Palindrome check via mirrored pointers.
- Expand-around-center for longest palindromic substring.
- Palindrome partitioning concept; connection to DP.

---

### Day 2: Substring & Sliding Window on Strings

- Longest substring without repeating characters.
- Longest repeating character replacement within K changes.
- Finding permutations and anagrams as substrings.
- Minimum window substring (classic pattern).

---

### Day 3: Parentheses & Bracket Matching

- Using stack to validate parentheses and similar bracket sequences.
- Generating well-formed parentheses (backtracking preview).
- Longest valid parentheses:
  - Using stack
  - Alternative DP-based interpretation.

---

### Day 4: String Transformations & Building

- String-to-integer (atoi) with overflow and invalid input handling.
- Integer-to-Roman and Roman-to-integer mappings.
- Zigzag conversion: coordinate mapping of characters to rows.
- Run-length encoding and simple compression schemes.

---

# ▨ PHASE C — TREES, GRAPHS & ADVANCED DS (Weeks 7–11)

---

## ▨ Week 7 — Trees & Heaps

**Primary Goal:**
Understand tree structure, traversal, BSTs, and heap-based priority queues.

**Why This Week Comes Here:**
Trees generalize linked structures into hierarchies; heaps provide efficient access to extremes and underpin many algorithms.

### Day 1: Binary Tree Anatomy

- Tree terminology: root, parent, child, leaf, height, depth.
- Structural types: full, complete, perfect, balanced.
- Representations:
  - Pointer-based node structures
  - Array-based (for heaps).

---

### Day 2: Tree Traversals

- Depth-first traversals:
  - In-order, pre-order, post-order.
- Breadth-first traversal: level-order with queues.
- Recursive vs iterative traversal methods.
- Use-cases: expression trees, tree serialization.

---

### Day 3: Binary Search Trees (BSTs)

- BST invariant: left subtree < root < right subtree (for some ordering).
- Searching, inserting, deleting nodes; structural transformations.
- Degenerate (linked-list-like) vs balanced BSTs.
- Real-world use: ordered maps/sets, indexes in DBs.

---

### Day 4: Heaps & Top-K Elements

- Min-heap and max-heap mental models.
- Heap operations: push, pop, heapify.

- Using heaps for Top-K problems and streaming scenarios.
- Real-world uses: schedulers, priority queues, job dispatching.

---

### Day 5: Balanced Trees (Conceptual)

- AVL trees: rotations, height balance property.
- Red-Black trees: coloring rules, balancing intuition.
- Complexity guarantees: O(log n) for search/insert/delete.
- Where they show up: language libraries (TreeMap, TreeSet), DB/FS internals.

---

## ▨ Week 8 — Tier 2 Strategic Patterns & Transformations

**Primary Goal:**
Learn strategic techniques for range updates, matrix manipulation, and advanced string patterns.

**Why This Week Comes Here:**
These patterns are "bridge concepts" to heavy-duty structures like segment trees/BITs and to more advanced string algorithms.

### Day 1: Difference Array & Range Tricks

- Prefix sums vs difference arrays:
  - Understanding why difference arrays can update ranges in O(1).
- 1D range updates, then final prefix pass to compute actual values.
- 2D difference arrays for matrices.
- Applications: event processing, range update queries.

---

### Day 2: In-Place Array & Matrix Transformations

- In-place rotation of matrix (e.g., 90° rotation).
- Transpose in-place; relationships between indices (i,j) and (j,i).
- Spiral order traversal: visiting a matrix layer by layer.
- In-place array rearrangement patterns:
  - Move zeros, partition by parity/sign, etc.

---

### Day 3: Advanced String Patterns

- Manacher's algorithm for palindromic substrings in linear time.
- Z-algorithm for prefix-based pattern matching and string analysis.
- KMP's failure function variants and deep intuition.
- String hashing (Rabin-Karp recap and more robust design).

---

## ▨ Week 9 — Graphs I: Foundations

**Primary Goal:**
Model problems as graphs and use BFS/DFS to explore and solve them.

**Why This Week Comes Here:**
You now have strong DS and pattern background; graphs add a powerful modeling dimension.

### Day 1: Graph Representations & Modeling

- Adjacency matrix vs adjacency list vs edge list.
- Memory usage and performance trade-offs.
- Implicit graphs: grids, puzzles, state spaces.
- Translating real problems into graphs (nodes and edges).

---

### Day 2: Breadth-First Search (BFS)

- BFS algorithm and queue-based frontier tracking.
- Shortest paths in unweighted graphs.
- Connected components and bipartite checks (conceptual).
- Applications: social networks, shortest route when edges all equal.

---

### Day 3: Depth-First Search (DFS)

- DFS algorithm via recursion or explicit stack.
- Use in exploring connected components, path existence, simple cycle detection.
- Differences vs BFS in typical tasks.
- Basis for many advanced algorithms (topo sort, SCC, etc.).

---

### Day 4: Graph Cycles & Connectivity

- Detecting cycles in undirected vs directed graphs.
- Connected components and articulation points (high-level).
- Union-Find/Disjoint Set for offline connectivity queries.
- Network connectivity examples: reliability of network, connectivity in grids.

---

### Day 5: Shortest Path I (Dijkstra)

- Dijkstra's algorithm: mental model of "expanding frontier of known shortest paths".
- Use of priority queue (heap) and visited set.
- Preconditions: non-negative weights.
- Practical uses: GPS routing, shortest latency path.

---

## ▨ Week 10 — Graphs II: Advanced

**Primary Goal:**
Understand advanced graph algorithms for shortest paths, MST, topological order, and basic flows.

**Why This Week Comes Here:**
These algorithms solve foundational problems in networking, scheduling, and resource allocation.

### Day 1: Shortest Path II — Bellman-Ford & Floyd-Warshall

- Bellman-Ford:
  - DP over edges; handling negative weights.
  - Detecting negative weight cycles.
- Floyd-Warshall:
  - All-pairs shortest path, triple nested loops DP view.
- Use cases and trade-offs vs Dijkstra.

---

### Day 2: Minimum Spanning Trees (MST)

- Definition: tree connecting all nodes with minimum total edge weight.
- Kruskal's algorithm:
  - Sorting edges, DSU-based merging.
- Prim's algorithm:
  - Growing MST from a starting node with a priority queue.
- Applications: network design, clustering.

---

### Day 3: Topological Sort

- DAGs: directed graphs with no cycles.
- DFS-based topological sort:
  - Post-order finishing times.
- Kahn's algorithm:
  - In-degree zero nodes with BFS.
- Uses: task scheduling, prerequisite resolution, build systems.

---

### Day 4: Network Flow I — Max Flow Basics

- Flow networks: capacities, flows, source/sink.
- Ford-Fulkerson method:
  - Augmenting paths, residual graphs.
- Max-flow min-cut theorem (intuition).
- Base applications: assignments, routing.

---

### Day 5: Network Flow II — Refinements & Matching

- Edmonds-Karp algorithm (BFS-based Ford-Fulkerson).
- Complexity and guarantees.
- Bipartite matching via max flow.
- Flow decomposition and real-world interpretations.

---

## ▨ Week 11 — Specialized Data Structures

**Primary Goal:**
Learn powerful data structures (tries, segment trees, BIT, DSU, suffix structures) for efficient queries and string processing.

**Why This Week Comes Here:**
By now, arrays/trees/graphs are familiar; you're ready for advanced DS that plug into range query and text processing tasks.

### Day 1: Tries (Prefix Trees)

- Trie structure: nodes and child pointers for characters.
- Insert, search, delete — mechanics and complexity.
- Applications: autocomplete, dictionary checks, IP routing.
- Trade-off vs hash maps: prefix queries vs memory usage.

---

### Day 2: Segment Trees

- Segment tree structure: representing intervals in a tree.
- Building a segment tree over an array.
- Range queries (sum, min, max) and point updates.
- Lazy propagation for range updates.
- Relation to difference arrays and prefix sums.

---

### Day 3: Fenwick Tree / Binary Indexed Tree (BIT)

- BIT structure: partial sums encoded in indices via bit operations.
- Point updates and prefix sum queries in O(log n).
- Comparison vs segment tree (simplicity vs flexibility).
- Use cases: prefix sums, offline queries.

---

### Day 4: Union-Find / Disjoint Set Union (DSU)

- Operations: find with path compression, union by rank/size.
- Amortized near-constant time complexity.
- Applications: connectivity, Kruskal's MST, grouping problems.

---

### Day 5: Suffix Structures

- Suffix arrays and suffix trees (conceptual).
- How suffix arrays support efficient substring queries.
- Basic string tasks: frequency of substrings, lexicographic ordering.
- High-level applications: text indexing, bioinformatics.

---

# ▨ PHASE D — ALGORITHM PARADIGMS (Weeks 12–13)

---

## ▨ Week 12 — Strings & Math Mastery

**Primary Goal:**
Master KMP, Rabin-Karp, bit tricks, number theory basics, modular arithmetic, and geometry essentials.

**Why This Week Comes Here:**
With strong DS and pattern background, these become powerful specialized tools rather than isolated tricks.

### Day 1: KMP (Knuth-Morris-Pratt) String Matching

- Failure/LPS array: “how much of the previous prefix do we keep?”
- Linear-time pattern matching; avoiding re-checking characters.
- Comparison to naive O(nm) algorithm.
- Application: substring detection in large text bodies.

---

### Day 2: Rabin-Karp & String Hashing

- Rolling hash: window-based hashing for substrings.
- Handling collisions with robust design (large mod, double hashing).
- Multi-pattern matching and approximate search (high-level).

---

### Day 3: Number Theory & Bit Manipulation

- Euclid’s algorithm for GCD; using it for LCM.
- Primality testing (basic) and prime-related operations.
- Bit operations:
  - Masks, shifts, AND/OR/XOR.
  - Common patterns like isolating lowest set bit, checking parity, subset generation.

---

### Day 4: Modular Arithmetic & Probability

- Modular addition/multiplication; avoiding overflow.
- Modular exponentiation (fast power) concept.
- Classic probability basics in algorithmic contexts (expected value).
- Reservoir sampling: uniform random selection from a stream.

---

### Day 5: Computational Geometry Basics

- Representation of points and vectors.
- Orientation tests using cross-product sign.
- Convex hull (Graham scan) idea.

- Applications: geographic systems, collision detection.

---

## ▨ Week 13 — Greedy & Backtracking

**Primary Goal:**
Understand when greedy is safe, and how backtracking systematically explores solution spaces with pruning.

**Why This Week Comes Here:**
You now know DS and fundamental algorithms; this week focuses on **choice strategies**.

### Day 1: Greedy Algorithms

- Greedy choice and optimal substructure properties.
- Classic examples:
  - Activity selection
  - Interval scheduling
  - Huffman coding.
- Counterexamples where greedy fails; comparing to DP.

---

### Day 2: Backtracking I — Permutations & Combinations

- Backtracking as DFS over decision trees.
- Generating permutations, combinations, subsets.
- Power set generation; bitmask vs recursive views.
- Pruning early when constraints are violated.

---

### Day 3: Meet-in-the-Middle (Optional / Advanced)

- Divide search space into two halves; combine partial results.
- Example: subset sum with n ~ 40 (half-split exponent).
- When this technique is effective vs overkill.

---

### Day 4: Backtracking II — Constraint Satisfaction

- Sudoku solver; constraint propagation idea.
- N-queens problem; representing states and constraints.
- Graph coloring small graphs.
- Role of heuristics: variable ordering, value ordering.

---

### Day 5: Backtracking III — Word & Path Problems

- Word ladder, Boggle, and typical board string search problems.
- Combining BFS/DFS with backtracking for search in large spaces.
- Thoughts on parallelizing search or pruning aggressively.

---

# ▧ PHASE E — PATTERN INTEGRATION & EXTENSIONS (Weeks 14–15)

---

---

## ▧ Week 14 — Dynamic Programming Mastery

**Primary Goal:**
Become fluent in identifying DP opportunities, designing states, and implementing transitions and optimizations.

**Why This Week Comes Here:**
DP builds on recursion, state modeling, and pattern recognition—skills you've developed over previous weeks.

### Day 1: DP Fundamentals

- Optimal substructure and overlapping subproblems.
- Top-down (memoization) vs bottom-up (tabulation).
- State definition: what each DP entry means, and which dimensions matter.

---

### Day 2: 1D DP & Classic Problems

- Climbing stairs variants (min cost, with constraints).
- House robber / non-adjacent sum.
- Longest increasing subsequence (LIS) via DP (and O(n log n) idea preview).
- Coin change variants (min coins, number of ways).

---

### Day 3: 2D / Sequence DP

- Longest common subsequence (LCS).
- Edit distance (Levenshtein distance); interpretation as matrix of transformations.
- Matrix chain multiplication: parentheses placement.

---

### Day 4: Advanced DP Techniques

- Digit DP: counting numbers with certain properties.
- Bitmask DP: subset-based state (e.g., traveling salesman, subset cover).
- DP on trees: subtree DP, combining children states.

---

### Day 5: DP Optimizations

- Space optimization: rolling arrays, dropping dimensions.
- Convex Hull Trick (CHT) concept for optimizing certain DP recurrences.
- Knuth optimization; divide-&-conquer optimization (high-level).

---

## ▧ Week 15 — Interview Pattern Integration

**Primary Goal:**
Reinforce and integrate patterns after mastering data structures, graphs, and DP.

**Why This Week Comes Here:**
This is your **post-strategic pattern practice** week: you revisit and integrate patterns in realistic problem settings.

### Day 1: Merge Intervals — Advanced

- Complex scheduling problems with varied constraints.
- Interval trees / segment tree-based interval reasoning (conceptual).
- Combining intervals with greedy and DP where necessary.

---

### Day 2: Monotonic Stack — Advanced Variants

- Next greater/smaller element in circular arrays.
- Trapping rain water problems; multi-dimensional variations (conceptual).
- More complex range visibility and skyline problems.

---

### Day 3: Cyclic Sort Pattern

- Advanced missing/duplicate number variants.
- First missing positive problem and in-place rearrangement.
- Designing cyclic sort patterns for customized constraints.

---

### Day 4: Matrix Problems — Advanced

- 2D search problems (matrix with sorted rows/cols).
- Set matrix zeroes and other transformation tasks.
- Combining prefix sums, difference arrays, and BFS/DFS on grids.

---

### Day 5: System Integration & Strategy

- Multi-concept problems:
  - E.g., combining BFS + DP + heap, or DSU + path queries.
- Interview strategy:
  - Reading problem
  - Uncovering invariants
  - Selecting patterns
  - Communicating trade-offs.

---

# ▨ PHASE F — ADVANCED DEEP DIVES (Weeks 16–18, Optional Track)

---

## ▨ Week 16 — Tier 3: Advanced Extensions

**Primary Goal:**
Extend familiar patterns (fast/slow pointers, two pointers, matrices, DSU, encoding) to complex variants.

**Why This Week Comes Here:**
For advanced learners, this week dramatically deepens skill beyond typical interview expectations.

### Day 1: Fast & Slow Pointers — Extended Problems

- Partitioning linked lists around pivots.
- Reordering lists (e.g., reorder list variants).
- Handling complex linked list structures with multiple pointers.

---

### Day 2: Reverse & Two Pointers — Extended

- In-place string/array segment reversals.
- Reversing words in a string with minimal extra space.
- Using deques and two pointers for hybrid patterns.

---

### Day 3: Matrix Traversal — Advanced Patterns

- Diagonal traversals (zigzag diagonals).
- Boundary and nested spiral traversals with tricky corner cases.
- Applying advanced traversal to problems like diagonal sums, anti-diagonals.

---

### Day 4: Matrix Exponentiation & Linear Recurrences

- Representing linear recurrences (e.g., Fibonacci) as matrix multiplication.
- Fast exponentiation of matrices.
- Applications to sequences and DP speedups.

---

### Day 5: Union-Find Advanced

- Weighted union-find, additional metadata per component.
- DSU on trees or more complex structures (conceptually).
- Combining DSU with other algorithms (e.g., Kruskal + connectivity via queries).

---

### Day 6: Conversion & Encoding

- Advanced run-length encoding patterns.

- Codec style problems (encode/decode lists of strings).
- Base conversions and simple domain-specific encodings.

---

### Day 7: Advanced Optimization Patterns

- Ternary search for unimodal functions.
- High-level view of approximate/heuristic algorithms.
- Parallelization opportunities in search and DP (conceptual).

---

## ▨ Week 17 — Advanced Mastery: Deep Dives Part 1

**Primary Goal:**
Study advanced DS and algorithms used in competitions and specialized systems.

**Why This Week Comes Here:**
This week is for those who want to push beyond typical SWE interviews into algorithmic engineering.

### Day 1: Segment Trees — Advanced

- Custom operators and function composition.
- Persistent segment trees (versioned data structures).
- 2D segment trees for range queries across dimensions (conceptual).

---

### Day 2: Heavy-Light Decomposition (HLD)

- Decomposing trees into heavy and light chains.
- Mapping tree paths to contiguous segments in arrays.
- Using segment trees/BIT over these segments for path queries.

---

### Day 3: Advanced Graph Algorithms

- Tarjan's algorithm for strongly connected components (SCC).
- 2-SAT via SCC component condensation.
- Biconnected components and articulation points; use in resilience analysis.

---

### Day 4: Advanced DP Optimizations

- Detailed look at CHT and monotone queue optimization.
- More examples of divide-&-conquer optimization.
- Criteria for when to apply these optimizations in practice.

---

### Day 5: Advanced String Algorithms

- Aho-Corasick automaton for matching multiple patterns.
- Suffix array construction methods (e.g., prefix doubling) at a high level.

- Revisiting Manacher's algorithm in context of entire string toolkit.

---

## ▨ Week 18 — Advanced Mastery: Deep Dives Part 2

**Primary Goal:**
Explore probabilistic data structures, advanced flows/geometry, and how algorithms manifest in system design.

**Why This Week Comes Here:**
This week bridges algorithms and large-scale systems, targeting advanced roles and system-level thinking.

### Day 1: Advanced Hash Structures

- Bloom filters: probabilistic membership with false positives but no false negatives.
- Count-Min Sketch: approximate frequency counting.
- HyperLogLog: approximate distinct counting.
- Trade-offs: accuracy vs memory vs speed.

---

### Day 2: Advanced Graph Coloring

- Graph coloring basics and NP-hardness context.
- k-coloring frameworks (for small graphs).
- Chromatic polynomial (high-level).

---

### Day 3: Advanced Network Flow

- Min-cost max-flow concepts.
- Circulation with demands.
- Applications: transportation, assignment with costs, scheduling.

---

### Day 4: Advanced Geometry

- Delaunay triangulation (intuitive notion of "nicest" triangulation).
- Voronoi diagrams: partitioning plane by closest site.
- Applications: load balancing, nearest-neighbor queries, spatial data.

---

### Day 5: System Design Patterns with Algorithms

- Where DSA sits in large systems:
  - Indexing
  - Caching
  - Ranking and search.
- Scalability considerations:
  - Sharding, replication, caching, load balancing.
- Mapping algorithm choices to system constraints.

---

# ▨ PHASE G — MOCK INTERVIEWS & FINAL REVIEW (Week 19)

---

## ▨ Week 19 — Mock Interviews & Final Mastery

**Primary Goal:**
Translate the entire curriculum into **interview-ready skill**: solving problems under time pressure, communicating well, and choosing patterns confidently.

**Why This Week Comes Here:**
You've built knowledge and intuition; now you practice applying them as in real interviews.

### Day 1: Mock Interview Session 1

- Run a full mock interview:
  - 2–3 problems of varying difficulty.
- Focus on:
  - Clarifying requirements, identifying underlying model (array, graph, DP, etc.).
  - Selecting appropriate patterns and data structures.
  - Articulating approach, constraints, and trade-offs.

---

### Day 2: Mock Interview Session 2

- New problem set targeting different areas:
  - E.g., one DP-heavy, one graph-heavy, one string/pattern-heavy.
- Emphasis on:
  - Adapting to problem types
  - Handling unexpected twists
  - Keeping calm under pressure.

---

### Day 3: Weak Points Diagnosis & Targeted Practice

- Analyze performance from Day 1–2:
  - Which topics felt shaky?
  - Which patterns did you struggle to recall?
- Design a mini-study plan:
  - Revisit specific weeks
  - Do targeted practice on those concepts.

---

### Day 4: System Integration Problems

- Solve 1–2 complex problems that combine multiple areas:
  - Graph + DP
  - DSU + MST + queries

    - Matrix + BFS + DP.
- Emphasis:
    - Decomposing complex problems
    - Layering data structures and patterns
    - Justifying each choice.

---

## Day 5: Final Preparation & Strategy

- Rapid review:
    - Core patterns and their “trigger phrases”
    - Time/space trade-offs for main structures
    - Common implementation pitfalls and how to avoid them.
- Create a personal ”interview checklist”:
    - How you start problems
    - How you explore ideas
    - How you handle dead-ends and pivot gracefully.

---

This completes the **19-week DSA Master Curriculum (v10)**:

- It is **sequenced** for mental-model-first learning.
- It **preserves all topics** from v9.2.
- It **clearly separates** standard interview coverage from advanced/optional deep dives.

You can now use this syllabus with:

- `Template_v10.md` (Instructional & Support file template)
- `SYSTEM_CONFIG_v10_FINAL.md` (global standards)
- `MASTER_PROMPT_v10_FINAL.md` (generation workflow)

to generate all instructional and support files in a coherent, pattern-based, systems-aware way.