DSA Foundational Topics

Core knowledge required to master all 20 DSA patterns

---

1. Programming Fundamentals

1.1 Language Basics (Java/Python/C++)

- **Variables & Data Types:** int, long, float, double, char, boolean
- **Operators:** Arithmetic, Logical, Bitwise, Comparison
- **Control Flow:** if-else, switch-case, ternary operators
- **Loops:** for, while, do-while, nested loops
- **Functions/Methods:** Parameters, return types, recursion
- **Input/Output:** Reading from console, writing to console

1.2 Object-Oriented Programming (OOP)

- **Classes & Objects:** Constructors, methods, attributes
- **Encapsulation:** Access modifiers (public, private, protected)
- **Inheritance:** Parent-child relationships, method overriding
- **Polymorphism:** Method overloading, runtime polymorphism
- **Interfaces & Abstract Classes:** Contracts, implementation

1.3 Memory Management

- **Stack vs Heap:** Where variables are stored
- **Pass by Value vs Reference:** How parameters are passed
- **Pointers/References:** Memory addresses (C++), object references (Java)
- **Garbage Collection:** Automatic memory management

---

2. Complexity Analysis

2.1 Time Complexity

- **Big O Notation:** O(1), O(log n), O(n), O(n log n), O(n^2), O(2^n)
- **Best, Average, Worst Case:** Understanding different scenarios
- **Amortized Analysis:** Average cost over many operations
- **Common Rules:**
    - Nested loops → O(n^2)
    - Divide and conquer → O(log n)
    - Single loop → O(n)

2.2 Space Complexity

- **Auxiliary Space:** Extra space used by algorithm
- **In-place Algorithms:** O(1) space
- **Recursive Space:** Call stack depth

2.3 Master Theorem

- **Recurrence Relations:** T(n) = aT(n/b) + f(n)
- **Solving Recursive Complexity:** Used for divide-and-conquer algorithms

---

3. Core Data Structures

3.1 Arrays

- **Static Arrays:** Fixed size, contiguous memory
- **Dynamic Arrays:** Resizable (ArrayList, Vector)
- **Operations:** Access O(1), Insert O(n), Delete O(n)
- **2D Arrays:** Matrix representation
- **Techniques:** Prefix sum, suffix sum, sliding window

3.2 Strings

- **Immutability:** String vs StringBuilder/StringBuffer
- **Operations:** Concatenation, substring, split, join
- **Pattern Matching:** KMP, Rabin-Karp (advanced)
- **Character Operations:** ASCII values, case conversion

3.3 Linked Lists

- **Singly Linked List:** Node with data + next pointer
- **Doubly Linked List:** Node with data + prev + next
- **Circular Linked List:** Last node points to first
- **Operations:** Insert O(1), Delete O(1), Search O(n)
- **Applications:** LRU Cache, browser history

3.4 Stacks

- **LIFO (Last In First Out):** Push, pop, peek
- **Implementation:** Using arrays or linked lists
- **Applications:** Undo operations, function calls, expression evaluation
- **Problems:** Balanced parentheses, stock span

3.5 Queues

- **FIFO (First In First Out):** Enqueue, dequeue, front
- **Circular Queue:** Efficient use of space
- **Deque (Double-ended Queue):** Insert/delete from both ends
- **Priority Queue:** Elements with priority (uses heap)
- **Applications:** BFS, task scheduling

3.6 Hash Tables (HashMap/HashSet)

- **Hashing:** Key → Hash Function → Index

- **Collision Handling:** Chaining, open addressing
- **Load Factor:** When to resize
- **Operations:** Insert O(1), Search O(1), Delete O(1) average
- **Applications:** Frequency counting, caching, lookups

3.7 Trees

- **Binary Tree:** Each node has max 2 children
- **Binary Search Tree (BST):** Left < Parent < Right
- **Balanced Trees:** AVL, Red-Black (advanced)
- **Traversals:**
  - **In-order:** Left → Root → Right (gives sorted order in BST)
  - **Pre-order:** Root → Left → Right
  - **Post-order:** Left → Right → Root
  - **Level-order:** BFS using queue
- **Operations:** Search O(log n), Insert O(log n) in balanced trees

3.8 Heaps

- **Min-Heap:** Parent < Children
- **Max-Heap:** Parent > Children
- **Heap Property:** Complete binary tree
- **Operations:** Insert O(log n), Delete O(log n), Peek O(1)
- **Implementation:** Using arrays
- **Applications:** Priority queues, Kth largest/smallest

3.9 Graphs

- **Representation:**
  - **Adjacency Matrix:** 2D array, $O(V^2)$ space
  - **Adjacency List:** Array of lists, O(V+E) space
- **Types:**
  - **Directed vs Undirected:** Edges have direction or not
  - **Weighted vs Unweighted:** Edges have weights or not
  - **Cyclic vs Acyclic:** Contains cycles or not
- **Components:** Connected components, strongly connected

3.10 Trie (Prefix Tree)

- **Structure:** Tree where each node represents a character
- **Operations:** Insert O(L), Search O(L) where L = word length
- **Applications:** Autocomplete, spell checker, dictionary

3.11 Disjoint Set (Union-Find)

- **Structure:** Track disjoint sets
- **Operations:**
  - **Find:** Which set does element belong to
  - **Union:** Merge two sets
- **Optimizations:** Path compression, union by rank
- **Applications:** Kruskal's algorithm, connectivity

---

4. Core Algorithms

4.1 Sorting Algorithms

- **Bubble Sort:** $O(n^2)$, simple but slow
- **Selection Sort:** $O(n^2)$, finds min and swaps
- **Insertion Sort:** $O(n^2)$, good for small/nearly sorted arrays
- **Merge Sort:** O(n log n), divide and conquer, stable
- **Quick Sort:** O(n log n) average, $O(n^2)$ worst, in-place
- **Heap Sort:** O(n log n), uses heap
- **Counting Sort:** O(n+k), works for limited range
- **Radix Sort:** O(d*n), digit by digit sorting

4.2 Searching Algorithms

- **Linear Search:** O(n), check every element
- **Binary Search:** O(log n), requires sorted array
- **Binary Search Variants:**
  - Find first/last occurrence
  - Floor/Ceiling of a number
  - Search in rotated array
  - Search in 2D matrix

4.3 Recursion

- **Base Case:** Stopping condition
- **Recursive Case:** Problem broken into smaller subproblems
- **Call Stack:** How recursion uses memory
- **Tail Recursion:** Optimization technique
- **Common Problems:** Factorial, Fibonacci, Tower of Hanoi

4.4 Backtracking

- **Concept:** Try all possibilities, backtrack if invalid
- **Template:**

```
backtrack(state):
  if is_goal(state): save solution
  for each choice in choices:
    make choice
    backtrack(new_state)
    undo choice (backtrack)
```

- **Applications:** N-Queens, Sudoku, permutations, combinations

4.5 Divide and Conquer

- **Concept:** Break problem into smaller subproblems
- **Steps:** Divide → Conquer → Combine
- **Examples:** Merge Sort, Quick Sort, Binary Search

4.6 Greedy Algorithms

- **Concept:** Make locally optimal choice at each step
- **When to use:** When local optimum leads to global optimum
- **Examples:** Activity selection, Huffman coding, Dijkstra's

4.7 Dynamic Programming (DP)

- **Concept:** Break problem into overlapping subproblems
- **Memoization (Top-down):** Recursion + caching
- **Tabulation (Bottom-up):** Iterative, build table
- **Steps to Solve:**
    1. Define state (dp[i] meaning)
    2. Find recurrence relation
    3. Base cases
    4. Order of computation
- **Common Patterns:**
    - **1D DP:** Fibonacci, climbing stairs
    - **2D DP:** Knapsack, LCS, edit distance
    - **Grid DP:** Unique paths, min path sum

4.8 Graph Algorithms

4.8.1 Graph Traversal

- **BFS (Breadth First Search):**
    - Uses Queue
    - Level by level
    - Shortest path in unweighted graph
- **DFS (Depth First Search):**
    - Uses Stack (or recursion)
    - Go deep, then backtrack
    - Cycle detection, topological sort

4.8.2 Shortest Path

- **Dijkstra's Algorithm:** Weighted graph, non-negative weights, O((V+E) log V)
- **Bellman-Ford:** Handles negative weights, O(VE)
- **Floyd-Warshall:** All pairs shortest path, $O(V^3)$

4.8.3 Minimum Spanning Tree

- **Kruskal's Algorithm:** Sort edges, union-find
- **Prim's Algorithm:** Greedy, similar to Dijkstra's

4.8.4 Topological Sort

- **Kahn's Algorithm:** BFS-based, uses in-degree
- **DFS-based:** Post-order traversal

---

5. Advanced Techniques & Patterns

5.1 Two Pointers

- **Same Direction:** Both move left to right
- **Opposite Direction:** One from start, one from end
- **Fast & Slow:** Different speeds (cycle detection)

5.2 Sliding Window

- **Fixed Size Window:** Window size K
- **Variable Size Window:** Expand/shrink based on condition

5.3 Bit Manipulation

- **Basic Operations:** AND, OR, XOR, NOT, shifts
- **Common Tricks:**
    - `x & (x-1)`: Remove rightmost set bit
    - `x ^ x = 0`: XOR with itself is 0
    - `x ^ 0 = x`: XOR with 0 is itself
- **Applications:** Subsets, unique numbers

5.4 Intervals

- **Merge Intervals:** Sort by start, merge overlaps
- **Insert Interval:** Find position, merge if needed

5.5 Monotonic Stack/Queue

- **Monotonic Stack:** Stack maintains increasing/decreasing order
- **Applications:** Next greater element, histogram problems

---

6. Mathematics for DSA

6.1 Number Theory

- **Prime Numbers:** Sieve of Eratosthenes

- **GCD/LCM:** Euclidean algorithm
- **Modulo Arithmetic:** (a+b) % m, (a*b) % m

6.2 Combinatorics

- **Permutations:** n!/(n-r)!
- **Combinations:** n!/(r!(n-r)!)
- **Pascal's Triangle:** Binomial coefficients

6.3 Logarithms

- **Properties:** log(ab) = log(a) + log(b)
- **Change of Base:** log_b(a) = log(a)/log(b)
- **Binary operations:** O(log n)

Learning Order (Prerequisites)

```
1. Programming Fundamentals
   ↓
2. Complexity Analysis
   ↓
3. Arrays & Strings
   ↓
4. Recursion Basics
   ↓
5. Sorting & Searching
   ↓
6. Linked Lists, Stacks, Queues
   ↓
7. Hash Tables
   ↓
8. Trees (BST, Traversals)
   ↓
9. Heaps
   ↓
10. Graphs (BFS, DFS)
   ↓
11. Advanced Graphs (Dijkstra, Topological Sort)
   ↓
12. Dynamic Programming
   ↓
13. Advanced Patterns (Backtracking, Tries, Union-Find)
```

Time Allocation (for 6-month plan)

Topic	Weeks	Why
Programming + Complexity	2	Foundation
Arrays + Strings	2	Most common
Linked Lists, Stacks, Queues	2	Linear structures
Sorting + Searching	2	Core algorithms
Hash Tables	1	Essential tool
Trees	3	Complex but crucial
Heaps	1	Priority problems
Graphs	3	Real-world modeling
Dynamic Programming	3	Optimization
Advanced (Trie, Union-Find, etc.)	2	Specialized
Practice & Revision	3	Mastery

Total: ~24 weeks (6 months)