**1. Data Structures**

* **Arrays**
* **Strings**
* **Linked Lists** (Singly, Doubly)
* **Stacks**
* **Queues** (Standard, Priority Queue, Deque)
* **Hashing** (Hash Maps, Hash Sets)
* **Heaps** (Min-Heap, Max-Heap)
* **Trees** (Binary Tree, Binary Search Tree, AVL Tree, Segment Tree, Fenwick Tree)
* **Graphs** (Adjacency List, Adjacency Matrix, BFS, DFS)
* **Disjoint Set Union (Union-Find)**
* **Tries**
* **Suffix Arrays and Suffix Trees**

**2. Algorithms**

* **Sorting Algorithms**
  + Quick Sort, Merge Sort, Bubble Sort, Selection Sort, Insertion Sort, Counting Sort, Radix Sort, etc.
* **Searching Algorithms**
  + Binary Search, Linear Search
* **Dynamic Programming**
  + Knapsack Problem, Longest Common Subsequence, Coin Change Problem, Matrix Chain Multiplication, etc.
* **Greedy Algorithms**
  + Activity Selection Problem, Huffman Coding, Kruskal’s and Prim’s Algorithms for Minimum Spanning Tree.
* **Backtracking**
  + N-Queens Problem, Subset Sum Problem, Permutations and Combinations.
* **Divide and Conquer**
  + Merge Sort, Quick Sort, Binary Search, Closest Pair of Points, etc.
* **Sliding Window Technique**
  + Longest Substring with K Distinct Characters, Maximum Subarray Sum.
* **Bit Manipulation**
  + XOR, AND, OR, bit shifting, counting set bits, and operations on bits.
* **String Matching Algorithms**
  + KMP Algorithm, Rabin-Karp Algorithm, Z-Algorithm, Aho-Corasick Algorithm.
* **Topological Sorting**
  + Used for Directed Acyclic Graphs (DAGs).
* **Matrix Exponentiation**
  + For solving recurrence relations efficiently.
* **Fenwick Tree (Binary Indexed Tree)**
  + For efficient querying and updating of prefix sums.

**3. Graph Algorithms**

* **Breadth-First Search (BFS)**
* **Depth-First Search (DFS)**
* **Dijkstra’s Algorithm** (Shortest Path)
* **Bellman-Ford Algorithm**
* **Floyd-Warshall Algorithm**
* **Topological Sorting** (for Directed Acyclic Graphs)
* **Minimum Spanning Tree**
  + Kruskal’s Algorithm, Prim’s Algorithm
* **Eulerian and Hamiltonian Paths**
* **Strongly Connected Components (Kosaraju’s or Tarjan’s Algorithm)**
* **Graph Coloring**
* **Network Flow Algorithms**
  + Ford-Fulkerson, Edmonds-Karp Algorithm

**4. Advanced Topics**

* **String Algorithms**
  + Suffix Arrays, Suffix Trees, LCP Array, Z-Algorithm
* **Number Theory**
  + Prime Numbers, Sieve of Eratosthenes, Modular Arithmetic, GCD, LCM, Euclidean Algorithm, Chinese Remainder Theorem
* **Combinatorics**
  + Permutations and Combinations, Pascal’s Triangle, Binomial Coefficients, Inclusion-Exclusion Principle, Pigeonhole Principle
* **Game Theory**
  + Minimax Algorithm, Nim Game, Grundy Numbers
* **Linear Programming**
  + Simplex Method
* **Geometry**
  + Convex Hull, Line Intersection, Polygon Area, Point-in-Polygon Test, Geometry in 2D/3D
* **Heavy-Light Decomposition**
  + Used for efficiently solving tree queries.

**5. Mathematical Algorithms**

* **Modular Arithmetic**
  + Modular Inverse, Fermat’s Little Theorem, Chinese Remainder Theorem
* **Matrix Multiplication**
  + Strassen’s Algorithm
* **Fast Exponentiation**
  + Exponentiation by Squaring, Modular Exponentiation
* **Probabilistic Algorithms**
  + Monte Carlo Methods, Randomized Algorithms

**6. Miscellaneous Topics**

* **Binary Search on Answer**
  + For optimization problems like finding the minimum/maximum value satisfying a condition.
* **Divide and Conquer**
  + Breaking down problems into smaller subproblems and solving them recursively.

**7. Problem-Solving Techniques**

* **Divide and Conquer**
* **Recursion and Backtracking**
* **Memoization and Dynamic Programming**
* **Iterative Techniques vs. Recursive Techniques**
* **Handling Large Inputs Efficiently** (Time and Space Complexity considerations)

**8. Practice Websites for Competitive Programming**

* **Codeforces**
* **AtCoder**
* **HackerRank**
* **LeetCode**
* **CodeChef**
* **TopCoder**
* **Google Code Jam**
* **Kick Start**
* **SPOJ (Sphere Online Judge)**
* **Project Euler**

**9. Time and Space Complexity**

* Learn to analyze the **time complexity** and **space complexity** of algorithms using **Big O notation**.
  + Common complexities: O(1), O(log n), O(n), O(n log n), O(n^2), O(2^n), etc.

**10. Interview Preparation**

* **Mock Interviews**: Practice real-world competitive programming problems often used in job interviews.
* **LeetCode Premium**: Offers interview-focused problem sets, including company-specific questions.

**Strategy for Learning**

* **Master Basic Topics First**: Arrays, Strings, Stacks, Queues, and Sorting algorithms.
* **Build a Strong Foundation in Recursion and Dynamic Programming**.
* **Start with Simple Problems** and gradually progress to harder ones as you get comfortable.
* **Practice Regularly**: Consistency is key to improving your problem-solving skills.

Let me know if you'd like detailed resources or links for any of the topics!