A Complete Data Structures & Algorithms Roadmap

Embarking on a data structures and algorithms (DSA) learning journey is a cornerstone for anyone delving into the world of computer science and software development. This roadmap outlines a structured path to mastering DSA, breaking down complex concepts into manageable segments, and suggesting a sequence for learning that builds upon each topic progressively. Let's dive into the essentials of each category and what you should focus on.

~ TIME & SPACE COMPLEXITY ~

- Algorithm Analysis: Understand the basics of algorithm analysis, focusing on how to evaluate the efficiency of an algorithm. This sets the foundation for all future learning in DSA. It is crucial you master Big O notation and time complexity analysis before moving forward.
- Space Complexity: Learn to assess the amount of memory an algorithm uses during its execution. Alongside time complexity, this gives a complete view of an algorithm's efficiency.

~ BASIC DATA STRUCTURES ~

- Arrays and Linked Lists (Single and Double): Start with these linear data structures, as they are the building blocks for more complex structures. Arrays offer fast access but fixed size, while linked lists provide dynamic size at the cost of slower access.
- Queue and Stack: Grasp these fundamental data structures for managing objects in a particular order. Queues follow a First In, First Out (FIFO) rule, while stacks use Last In, First Out (LIFO).
- Binary Trees and Binary Search Trees: Understand these hierarchical structures for efficient data organization and retrieval. Learn about traversals, creation, insertion, deletion and searching.
- Heaps: Learn about heaps for priority queue implementation and efficient sorting. Understand both the min and max heaps and how they are created and how elements are added and removed.
- Graphs: Dive into graphs to represent and work with networks of nodes and edges. Specifically, look at graph representations like adjacency lists, adjacency matrices and edge lists. You should also familiarize yourself with graph terminology.
- Hashing: Explore hashing for fast data retrieval through key-value pairs. This is to ensure you understand how hash maps and various other key-value paired data structures operate.

~ ALGORITHMS ~

- Recursion: Understand recursion for problems where a solution requires solving smaller instances of the same problem.
- Searching and Sorting Algorithms: Learn basic algorithms like binary search, insertion sort, bubble sort, selection sort, heap sort, quicksort, and mergesort for fundamental data manipulation tasks.
- Path Finding Algorithms: Study algorithms like Dijkstra and A* for finding shortest paths in graphs.
- Graph Algorithms: Delve into algorithms for traversing, searching, and analyzing graphs (e.g., BFS, DFS). Also look at minimum spanning tree algorithms like Kruskal and Prims Algorithm.
- Dynamic Programming: Master dynamic programming for optimizing recursive problems by storing results of subproblems.
- Greedy Algorithms: Understand greedy algorithms for solving optimization problems by making the locally optimal choice at each step.
- Divide and Conquer Algorithms: Learn how to break down problems into subproblems, solve them independently, and combine their solutions.
- Backtracking Algorithms: Explore backtracking for finding all (or some) solutions to problems by exploring possible candidate solutions.

~ ADVANCED DATA STRUCTURES (Optional) ~

The list below contains much more advanced data structures which are good to know but not commonly asked about in interviews or assumed to be common knowledge.

- Tries, B Trees, AVL Trees, and Red-Black Trees: These advanced tree structures provide efficient searching and balanced tree properties for different use cases.
- Skip Lists, Segment Trees, Fenwick Trees: Learn about these specialized data structures for specific scenarios like fast search within a list or supporting range queries and updates.
- Disjoint Set: Understand disjoint set (or union-find) for keeping track of a set of elements partitioned into non-overlapping subsets.

~ ADVANCED MATH (Very Optional) ~

If you really want to punish yourself and understand the math behind advanced algorithms and famous computer science proofs learn this.

- Combinatorics, Probability, Discrete Math, and Discrete Structures: These mathematical foundations are crucial for understanding the theoretical underpinnings of algorithms and data structures, including graph theory, probability, and more.

~ HOW TO LEARN THIS ~

The truth is there are a million different ways you can learn DSA, however, I will share with you what I did personally.

1. MASTER Time Complexity
   - Before diving into anything else ensure you have a solid understanding of time complexity analysis and can easily identify the run-time of any code you see.
2. Follow a course for the theory
   - Learning DSA is hard enough, don't make it harder by learning each algorithm/data structure from a different source. Spend some time, and potentially some money to find a great curriculum taught by a reputable instructor that takes you through all of the non-optional content listed above.
3. Quiz Yourself
   - A good course will have exercises and quizzes built-in but if not make sure you quiz yourself on the theory and truly understand the various data structures. Try your best not to memorize but to really deeply understand. It's been 4 years since I formally learned DSA and I can still answer any question you ask me, not because I memorized concepts, but because I really understand them.
4. Practice Famous Algorithms
   - Everyone should have experience writing algorithms like bubble sort and merge sort at least once. For the list of extremely famous and common algorithms actually write and implement them in code. This will really expose if you actually know what you're doing.
5. Do Coding Interview Style Questions
   - This is where you step outside of pure theory and into the real world. Try answering famous DSA problems asked by top tech companies like google. Here you need to take all your knowledge and apply it to problems you haven't been taught the solution to. This is HARD and will take a LOT of practice to get good at.
   -

Now I know what you're thinking, damn..., that's a lot. You're right, it is, but the truth is that to really understand this stuff it takes time and a ton of practice. With that said, most of you will be able to get by just being "decent" in this topic. It's really only the top tech companies that will require you to pass coding interview style questions where this knowledge is critical, if you're not aiming for those types of positions there's no need to grind this for months.