Data Structures & Algorithms Study Guide

In this doc, I've compiled a list of the most important topics that you need to know for your interview as well as an outline of roughly what you should know about each topic.

This is by no means a comprehensive list, but as I've discussed, my goal is to give you the most important things to focus on. Use this list as a starting point and feel free to add to it and adapt it as you see fit.

I've also organized the topics roughly in order of importance, but I encourage you to focus on the topics that you find most difficult.

Arrays and Strings

I've written extensively about this here. Please refer to the article for additional detail.

- **Using a Length-256 Integer Array**
  Rather than storing characters into a hashmap, it is often easiest to just use a fixed-length array. This simplifies our code and standardizes the amount of space we're using.
  - Anagrams
  - Sorting the characters in a string
  - Longest substring without a repeating character
- **Use 2 pointers**
  Many times rather than using a single pointer to walk through the string or array, we may want to use multiple pointers.
  - Remove Duplicates
  - Is String a Palindrome
  - Reverse Words in a String
- **Array/String Math**
  We definitely want to be prepared to convert between strings/arrays and integers and do all those sorts of fun things.
  - Convert Strings to Binary
  - String to Integer
  - Compare Version Numbers
- **Sliding Windows**
  There are lots of good articles on this topic including this one
  - Find All Anagrams in a String
  - Minimum Window Substring
  - Substring with Concatenation of All Words
- **Comparison, Alignment, and Matching**
  These problems tend to have a lot of overlap with other topics, such as recursion.
  - Longest Common Substring
  - Edit Distance
  - Regex Matching

Trees/Graphs

- **Basic things you should implement**
  - Insert and remove node from graph or tree
  - Implement a graph as an adjacency list, adjacency matrix, and node classes (https://www.cs.rochester.edu/~nelson/courses/csc_173/graphs/implementation.html)
  - Implement a function to convert a graph between different implementations
- **Traversals**
  - Inorder Traversal
  - Preorder Traversal
  - Postorder Traversal
  - Level order Traversal
- **DFS and BFS**
  These are core to basically everything. You should know them like the back of your hand and understand when to use one versus the other.
  - Max Binary Tree Depth
  - Lowest Common Ancestor
  - Word Ladder
  - Sliding Puzzle
- **Divide and conquer**
  Most tree problems that aren't DFS or BFS end up being some variation of divide and conquer. Process each side of the tree separately and then combine the result somehow.
  - Merge Binary Trees
  - Balanced Binary Tree
  - Invert Binary Tree
  - Tree to List
- **Graph coloring/scheduling**
  This is a less common topic and you don't need to understand it in detail, but it is good to at least briefly review. Graphs can often be used to represent real world situations, so you want to understand how that works.
  - Course Schedule
  - Is Graph Bipartite
  - Build Order

Recursion

I've written extensively about this here. Please refer to the article for additional detail.

- **Iteration**
  Use recursion to iterate over a set of data.
  - Insert an item at the bottom of a stack
  - Print a linked list in reverse order
  - Find all substrings of a string
- **Subproblems**
  Break the problem down into a series of smaller subproblems. Then get the solution by combining the solutions to all of the individual subproblems.
  - Towers of Hanoi
  - Is a string a palindrome?
  - Stair steps
- **Selection**
  If you understand one pattern, focus on this one. The core pattern is to find all of the combinations of a given set of inputs and then filter the combinations to find the ones that match your desired result.
  - Find all combinations of a set of inputs
  - 0-1 Knapsack
  - Find all ways to interleave 2 strings
- **Ordering**
  This is similar to Selection except that you are going to find all of the permutations instead of all of the combinations.
  - Find all permutations of a set of inputs
  - Find all permutations when the input includes duplicates
  - Find all N-digit numbers whose digits sum up to a target
- **Divide and Conquer**
  These tend to be some of the trickiest recursion problems. In these problems, you are in some way splitting up the input and recombining the results.
  - Implement binary search
  - Implement mergesort
  - Find all unique binary search trees
- **Depth-first Search**
  Almost all recursive problems can be reframed as DFS. However, the most common applications are the ones that you are already aware of, such as trees and graphs.
  - Find all paths between two nodes in a graph
  - Knight probability
  - Greatest product path

Dynamic Programming

For dynamic programming, know the FAST Method. That's the most important thing. If you haven't already, you can download my free ebook here: www.byte-by-byte.com/dpbook

Bit Manipulation

Bit manipulation doesn't come up too frequently, but it is useful to have a good handle on the basics. Here are the core things you should be aware of:

- **Properties of different operations**
  - Bitwise AND, OR, NOT, and XOR
  - Hamming Distance
  - Missing Number
- **N & (N-1)**
  This common operation removes the lowest-order 1 bit from a binary number.
  - Counting bits
  - Number of Ones
- **Bit masking**
  Sometimes we want to extract a specific value or range of values from a binary number. We can do this by doing a bitwise AND with a mask.
  - Convert to Hexidecimal
- **Bit shifting**
  Bit shifting allows us to quickly divide by 2, create bitmasks, and do a variety of other binary operations
  - Reverse Bits
  - Rotate Bits
- **Properties of binary numbers**
  - Add binary numbers
  - Divide binary numbers
  - Powers of 2
  - Powers of 4

Searching and Sorting

- **Types of Sorts and Searches**
  For each common sort you should know how to implement it, the complexity, and any tradeoffs.
  - Quicksort (and Quickselect)
  - Mergesort (Top-down and bottom-up)
  - Insertion Sort
  - Heap Sort
  - Binary Search
- **Implementing in your language**
  It is super common that you'll want to use built-in sorting for your language. You should be prepared to do this.
  - Sort primitives
  - Custom searches/comparators
  - Custom iterators

Linked Lists

- **Linked List Basics**
  You should be able to implement all of the following for both singly- and doubly-linked lists
  - Iterate over the list
  - Insert into the list
  - Remove from the list
  - Reverse the list
  - Split the list
  - Find cycles in the list
- **Runner pointers**
  - Find the nth element in a list
  - Remove the nth-to-last element
  - Find the middle of the list