How to Study for Data-Structures and Algorithms Interviews at FAANG

Esco Obong Follow
Jan 1 · 15 min read

Written By Esco Obong (@escobyte on Twitter), Senior Software Engineer @Airbnb, Founder of Algorythm study group on Facebook and Black Software Engineers Career Support Group on LinkedIn.

This is the story of how I got offers from **Google**, **Amazon**, **Uber** and more without a college degree.

This was me in 2015 ☠️. A startup I had joined as "founding employee" after we raised a

$500k seed round from a prototype was shut down six months later and we were looking for new roles. I secured an interview with Codecademy through a referral from one of the startup founders (also GIPHY but I wasn't interested for some reason. Since then they sold to Facebook 🤦‍♀️)

On the phone with Codecademy they said "don't worry we don't ask crazy algorithms questions or anything like that". I took that to mean I didn't need to study algorithms at all 🤣.

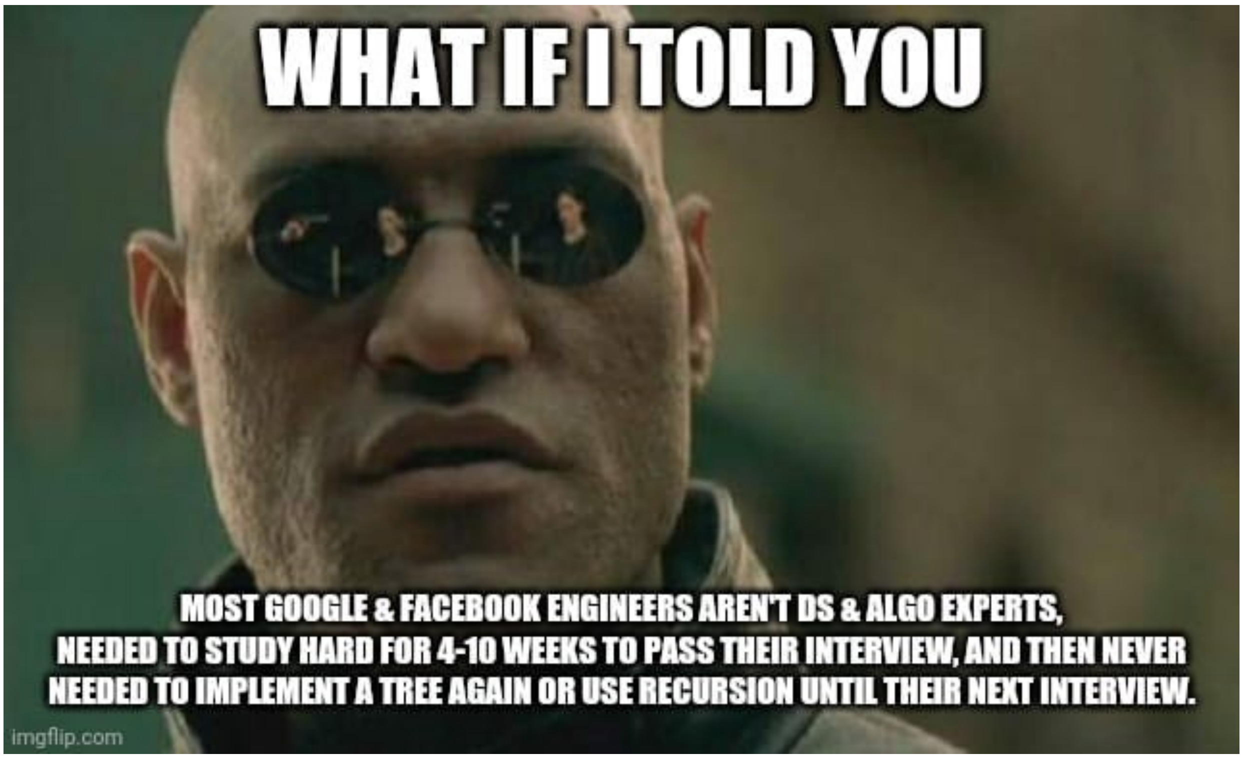
During the on-site interview, I got two rounds of algorithms questions that were extremely basic in hindsight. I remember one of them was asking how to traverse from point A to point B in a grid. I had no clue how to do that so I was just doing random shit. I landed on an infinite **while** loop… I actually wrote on the board:

> " while (true) {..."

In the loop, the point changed direction on each iteration depending on if it hit a wall and would eventually manually break the **while** loop if the target was found. The interviewer must've been like WTFFF, but he kept his cool and kept entertaining my different ideas.

That fiasco opened my eyes to the types of questions I needed to be able to answer. Two and a half months later I passed phone screens at Google, Uber, Shutterstock and Rent the Runway. I did the on-site interviews for Shutterstock, Rent the runway, and Uber and passed all of them. Google rescheduled my interview to be two weeks later than what I told them in the last minute. By that time I already had three offers and was pretty excited about Uber so I pulled the trigger and joined Uber.

I went from 0 → 100 in just a few months and I didn't do anything special aside from studying consistently. That's why I strongly believe any engineer can get good at these DS & Algo questions and get into F.A.A.N.G. or similar high paying roles.


WHAT IF I TOLD YOU
MOST GOOGLE & FACEBOOK ENGINEERS AREN'T DS & ALGO EXPERTS, NEEDED TO STUDY HARD FOR 4-10 WEEKS TO PASS THEIR INTERVIEW, AND THEN NEVER NEEDED TO IMPLEMENT A TREE AGAIN OR USE RECURSION UNTIL THEIR NEXT INTERVIEW.
imgflip.com

At first, I felt I wasn't really qualified because I had to study so hard to pass my interviews. In 2019, after having run my own consulting firm and startup for almost 2 years, I decided to go back into full-time work and found myself in the same position as I was in 2015.

This time, I had a larger network of friends at Uber and other top companies like Google and Facebook that gave me insight into the real deal. They *all* had to study just as hard. Even the ones that didn't drop out of college like I did.

> The ones who stayed and completed their Algorithms' courses and had gotten their Computer Science degrees, they all still needed to study hard.

I discarded this notion of the mythical engineer who can on a whim pass a tech interview and started to appreciate the reality of the situation, that tech interviews are like the SAT's they give in school. It doesn't matter that you spent four years learning all of the content in high school, you still need to prep if you want to ace the test. Just like with the SATs all of your past work and grades don't contribute to the score. Your success depends entirely on how well you perform on the test.

Once I realized the truth, that everybody needs to study, it was enough motivation for me to put in the hard work since I realized that's what my competition was doing. Through that motivation, I formed a process for studying that helped me pass every tech screen and on-site interview I did in 2019. I, a college dropout, passed technical screens for Stripe, Coinbase and Triplebyte. I passed screens and on-site interviews for Google, Amazon, Uber (again), Reddit, Squarespace and Braze. A 100% pass rate wasn't expected and isn't likely to continue to happen, but I believe focusing on the fundamentals helps approach that goal.

Google sends you swag when you pass the interview and match with a team

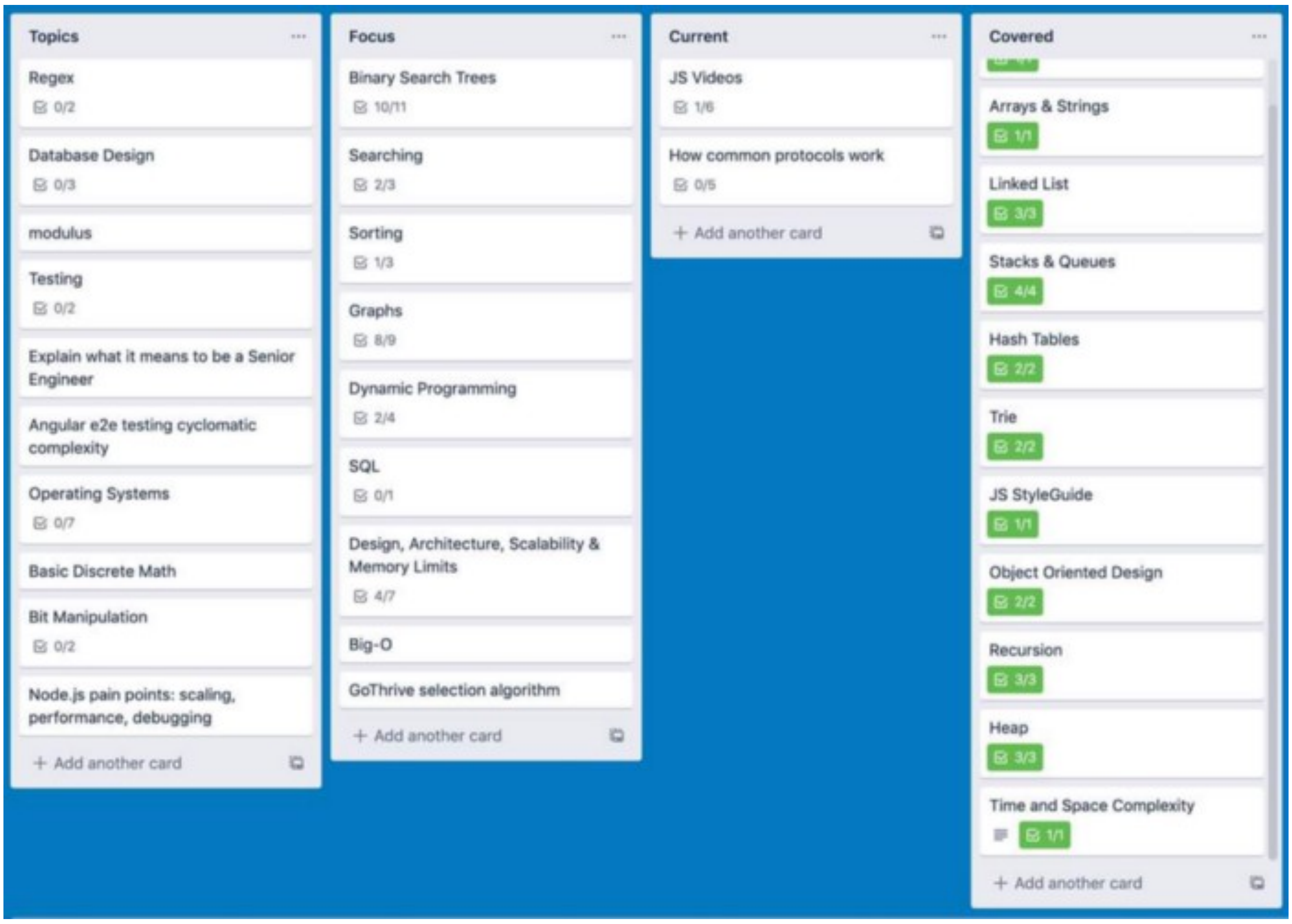
It wasn't a smooth transition. When I first started preparing for interviews, it took me over 2 hours to **ALMOST** solve what would now be called a "leetcode easy" and at the time it felt impossible! I truly believed that I just wasn't naturally smart enough and would have to study even harder to get to the same level where I could solve the problem at the same speed as a Google engineer.

I got one thing right, you do need to practice.

> What I didn't realize is that this level of difficulty and frustration in the beginning is to be expected by everyone just starting out, including those Google engineers.

I'm still the same person. The only difference between now and then was practice, practice, practice.

A common question people have is "how do I study"? It's hard to know where to start, what to do next, if you're making progress and even harder to stay on track. To solve a lot of these issues when I was studying, I created a Trello board with all the topics I wanted to cover. The board helped me focus on the most important topics I should be studying and manage my time to keep progress consistent.



Topics

- Regex 0/2
- Database Design 0/3
- modulus
- Testing 0/2
- Explain what it means to be a Senior Engineer
- Angular e2e testing cyclomatic complexity
- Operating Systems 0/7
- Basic Discrete Math
- Bit Manipulation 0/2
- Node.js pain points: scaling, performance, debugging
- + Add another card

Focus

- Binary Search Trees 10/11
- Searching 2/3
- Sorting 1/3
- Graphs 8/9
- Dynamic Programming 2/4
- SQL 0/1
- Design, Architecture, Scalability & Memory Limits 4/7
- Big-O
- GoThrive selection algorithm
- + Add another card

Current

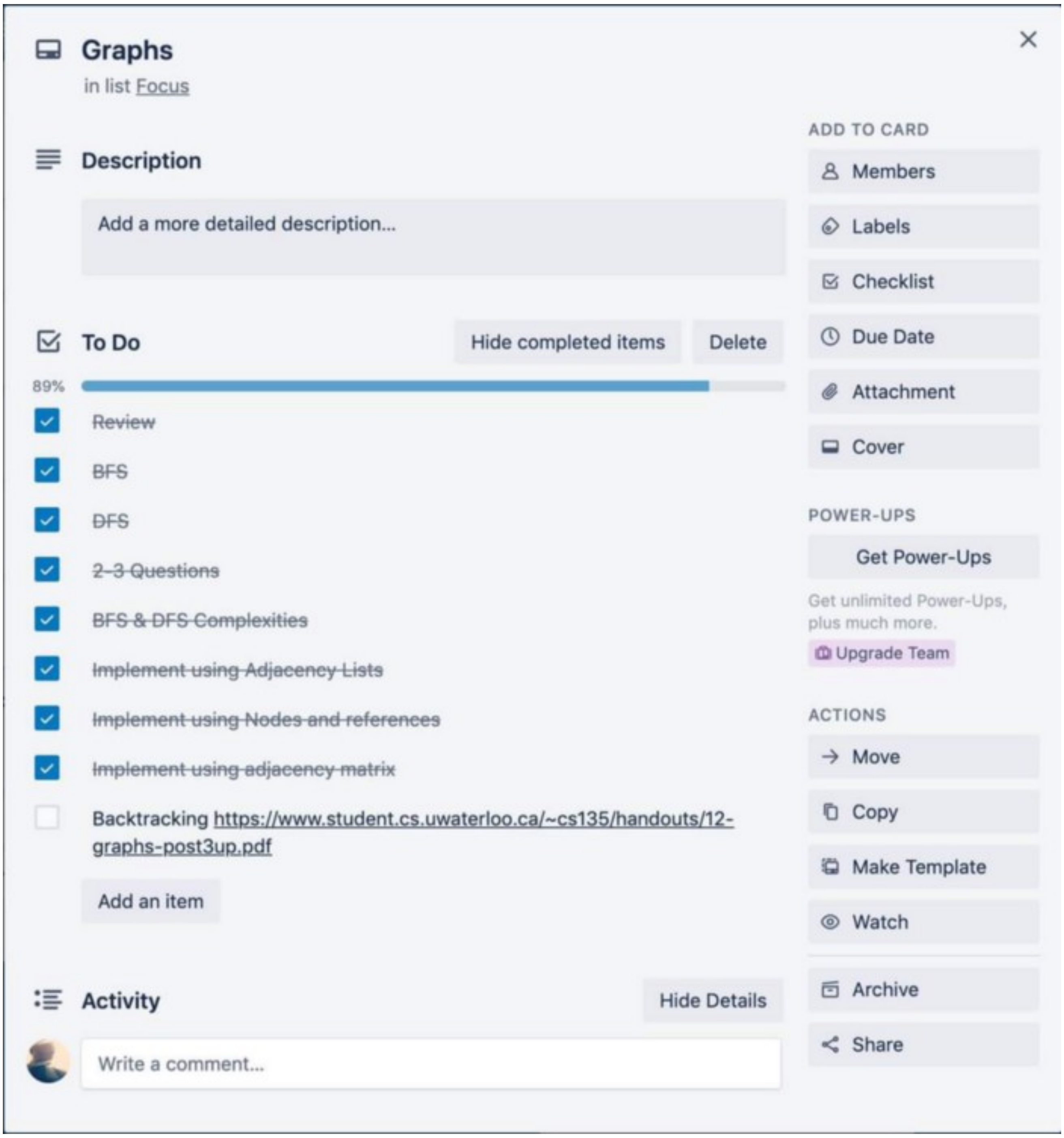
- JS Videos 1/6
- How common protocols work 0/5
- + Add another card

Covered

- Arrays & Strings 1/1
- Linked List 3/3
- Stacks & Queues 4/4
- Hash Tables 2/2
- Trie 2/2
- JS StyleGuide 1/1
- Object Oriented Design 2/2
- Recursion 3/3
- Heap 3/3
- Time and Space Complexity 1/1
- + Add another card

I made a post about the Trello board on LinkedIn and it went viral. A Trello PM contacted me about creating an official Trello template for it which can be found here: Interview Study Tracker.

The tracker provides a template for how to study. It requires identifying a list of topics and searching for two or more resources that can teach you about the topic. Then assigning yourself 2–3 practice questions to solidify your understanding. It can be from any source, the most important thing is the content itself. I found most of my content with simple google searches using the topics as the keyword.



Graphs

in list Focus

Description

Add a more detailed description...

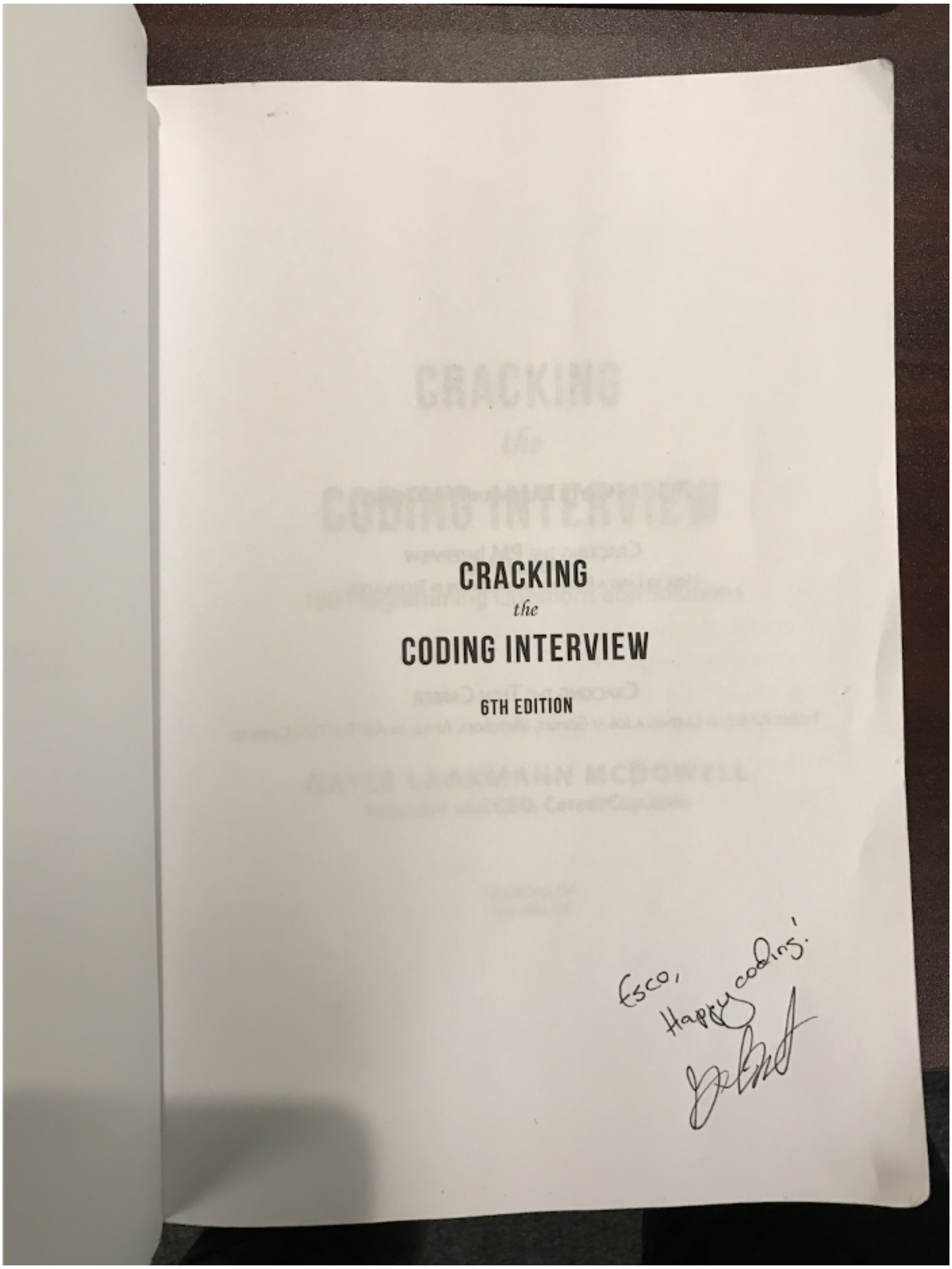
To Do

89%

- [x] ~~Review~~
- [x] ~~BFS~~
- [x] ~~DFS~~
- [x] ~~2-3 Questions~~
- [x] ~~BFS & DFS Complexities~~
- [x] ~~Implement using Adjacency Lists~~
- [x] ~~Implement using Nodes and references~~
- [x] ~~Implement using adjacency matrix~~
- [ ] Backtracking https://www.student.cs.uwaterloo.ca/~cs135/handouts/12-graphs-post3up.pdf

Get unlimited Power-Ups, plus much more.

Learning the same thing from different perspectives helps you understand it better. For example, I would read an article on geeksforgeeks.org about binary search trees and also read chapters in Cracking the Coding Interview about them. Then I would do 2–3 practice questions or enough to feel comfortable before moving on.


CRACKING
the
CODING INTERVIEW
6TH EDITION

Esco,
Happy coding!
[illegible]

My copy of “Cracking The Coding Interview 6th Edition” signed by Gayle Laakmann McDowell

List of Study Topics

Below is a list of nearly all of the data-structures and algorithms that you can come across during a tech interview along with a very brief description and some resources to get started building your study plan. The goal is to provide a starting point for you to build your plan and not to explain in detail what each topic is. Part of your plan is finding resources to learn more about each topic from different perspectives.

If you haven't heard of some data-structures in the list or haven't learned much about them before, they might seem complex but as you progress, you'll soon experience the same feelings you had before and after you understood Arrays.

> Think about how complex the concept of an Array is to a non-programmer and how basic it is to even the most junior developers.

These data-structures are just the basic building blocks that help you create algorithms… they actually make things simpler once you take the time to understand them. Just as an Array handles a lot of the work required to maintain an ordered list of elements, these data-structures have functions and containers for efficiently working with data.

Basic Data-Structures

These are considered basic. Everyone should know how they work, when to use them, **how to implement them** and the reasoning behind their tradeoffs:

- Array
- Set
- Hashmap
- Linked List
- Stack
- Queue
- Tree
- Graph

If this list seems long, don't worry. A lot of these data-structures are closely related and even build upon each other so you'll be able to learn them all quicker than you might think.

Hashmaps build upon Arrays. Stacks build upon LinkedLists or Arrays. Queues build upon LinkedLists. Trees share concepts of "nodes" and "links" from LinkedList. Graphs also share the same concepts. They also have a lot of similarities to trees. All trees are technically graphs too so you can apply a lot of the same techniques to them.

Advanced Data-Structures

After mastering the basic data-structures, knowing these more advanced ones will give you a higher chance of success. Some come up very often during interviews like Heaps and Binary Search Trees. LRU Caches and Tries come up less frequently but are becoming more common. Disjoint Sets and Skip Lists rarely come up, but even if not explicitly asked about them, they can be powerful tools to help you come up with quick and performant solutions.

- Heap (a.k.a Priority Queue)
- LRU Cache
- Binary Search Tree (AVL, Redblack)
- Disjoint Set
- Trie
- Skip List

As we go into more advanced data-structures you'll see they are usually made using the basic ones as building blocks. Getting a good understanding of the basic data-structures will make learning these advanced ones much easier. You'll think of them in terms of the smaller structures you already know rather than big new complicated structures.

Heaps, also known as Priority Queues are a type of tree that behave like a queue. Binary Search Trees (BSTs) are another type of tree that make it faster to search for nodes. AVL and Redblack are two popular specific types of balanced BSTs. An LRU Cache is a memory efficient cache built by combining a Hashmap with a LinkedList.

A Trie is another type of tree that makes prefix/substring searches fast. Disjoint Sets are a special type of set that separate its members into non-overlapping subsets, useful for the Union Find algorithm. A Skip List is an optimized version of a LinkedList that reduces the time it takes to find certain nodes.

Basic Searching/Traversal Algorithms

All data structures are meant to hold information. Some structures need specials ways to access this information efficiently that are more involved than simply accessing an array index or hashmap key. Data-structures that hold collections of data such as Trees and Graphs use a few basic algorithms to access the information within.

- Breadth First Search (BFS)
- Depth First Search (DFS)
- Binary Search

Advanced Searching/Traversal Algorithms

There is a ton of research in this field, but for the purposes of a tech interview, the most advanced searching algorithms you'll likely come across in order of frequency are:

- Quick Select
- Dijkstra
- Bellman-Ford
- A-star (rare)

Sorting Algorithms

Sorting is a common tool used to increase the performance of a solution. There are many sorting algorithms, but the most popular for interviews are listed below. Know how to implement all of these without looking anything up.

- Quick Sort
- Merge Sort
- Topological Sort
- Counting Sort

Important Topics

Recursion is an extremely important topic that doesn't come up as much in day to day software engineering as it does during interviews. Bit manipulation may seem scary at first, but once you take time to understand the binary number format and operations you'll realize how simple it is.

- Recursion
- Greedy Algorithms
- Dynamic Programming
- Bit Manipulation (AND, NOT, OR, XOR)

Common Patterns

These patterns can be used to solve many similar algorithms questions

- Backtracking
- Two Pointers
- Sliding Window
- Divide & Conquer
- Reservoir Sampling

Math based problems

- Permutations
- Combinations
- Factorial
- Power Set

Other Common Problems

- String to Integer
- Integer to String
- Adding huge numbers (that can't fit into memory)
- Addition/Subtraction/Multiplication/Division without using operators

Complexity

As weird as it sounds, it's true that it's not enough to just provide a working solution to problems during the interview. The code has to run at a certain level of performance described by "time complexity" & "space complexity". The complexity is described in terms of **Big-O notation**.

This may seem complex at first (especially given the name) but the reason most people have trouble with it is because they don't take time to actually learn and just go by intuition after seeing a few examples. Rather than rely on intuition, read up on Big-O notation and how to determine the complexity of an algorithm. Oftentimes your interviewer will ask you to tell them what the time and space complexity of your solution is.

Practice

Self Practice

Studying means nothing without practice. Every time you learn a new topic from your study plan, you should put it to use. That'll help you remember it longer and also give a deeper understanding. Do a few practice questions for every topic you finish. Sites like leetcode.com let you search for practice questions using tags like "trees" or "graphs".

Practice should happen in two phases. The first is when you learn the topic for the first time. During this phase you should focus on really understanding whats involved and why your solution works. The second phase is after you're comfortable with the concepts. At this point you should time yourself and aim for faster and faster solution times.

During an interview you'll be expected to solve a problem within 15–45 minutes depending on the difficulty of the question. When you start out, it'll likely take hours for you to do the first one. Over the first few weeks that time should start to come down. After a month or two of consistent practice (not just time passing), you should start being able to solve a decent amount of problems on time.

Sometimes an entire interview question will only involve implementing a single data-structure such as an LRU Cache or a Trie. Other times it'll only involve implementing a single operation on a data-structure such as the delete operation on a Binary Search Tree.

Practice implementing all of the data-structures mentioned earlier in this article until you don't need to look anything up and you will have a very easy time with these questions during the real interview. Focus on understanding why its implemented the way it is rather than trying to remember the exact code. Implement from understanding of what needs to happen next and not from memory.

> *You probably won't be asked to implement an Array or balanced BSTs like AVL or Redblack trees but you should know how they work.*

Assisted Practice (Mock Interviews)

Practicing topics and questions alone isn't enough. During the interview you'll be interacting with a real person who is judging your skills in real-time. The situation is more stressful than when you study at home. To get around the awkwardness and anxiety of a real interview, you should do many practice interviews beforehand. This will help you be comfortable so that you can focus on the interview question rather than being distracted by non-technical things.

These types of practice interviews are called Mock Interviews. You can have a friend help you or find a service online that'll match you with an interviewer.

Knowing when you're ready

You'll never feel "ready" no matter how long you study. Theres always more topics to dive deeper into or weak areas that could use more practice.

There is definitely an element of luck in the whole process. It's very possible that you get a question on a topic you haven't prepared for at all or are really weak on. It's also possible that you get all questions on topics you're really strong in since two months ago and didn't need to spend that extra time studying to pass. You can't prepare for every possible outcome, but your chances of success get higher than everyone else's with more practice.

Keeping track of your progress and getting feedback is very important in knowing when you're truly ready to perform on the real interview without relying on just emotion.

Keep track of your completion times when you do practice questions and aim for at most 40 minutes to complete most medium level questions and 1 hour to complete hard level questions on sites like leetcode.com or hackerrank.com.

Realize that you must pass every edge case for it to count on those sites but luckily this isn't the case for real interviews! Most interview code is evaluated on a small set of edge cases and small errors get passed or considered to be irrelevant all the time. The point of evaluation isn't to be so strict as to expect perfection, but they need to see enough "signals" to be convinced.

Do at least three mock interviews before you do the real one. Only move forward when you can consistently perform well during practice.

Big tech companies hire non-stop so they're extremely flexible on scheduling or rescheduling your interview date. I rescheduled my interviews with Google and Uber by 2 and 1 months respectively because I wasn't ready. I've done even longer push backs in the past.

You don't need any special excuse to reschedule an interview. I simply told them "I need more time to prepare". These companies dedicate a lot of money for your interview. 5-6 full-time employees get paid to interview you for 4-5 hours and then evaluate the results after. Factor in the salaries of recruiters who took time to find you and process only a small % of applications for people they interact with and that's at least a few thousand dollars per interview.

They want you to come prepared. They will wait rather than waste time, money and the opportunity cost of rejecting an engineer who could've passed if they just had waited an extra month or two. So push back your interview if you're not ready. Never go in without being ready!

If you follow the above, knowing when you're ready is easy. You might still feel you aren't but you'll know for sure because you're hitting your completion times and have passed many mock interviews. Being ready means you've proven undoubtedly through practice that you can pass interviews that ask the same types of difficult questions under the same time limits as the real one.

This article only focuses on data-structures and algorithms questions. System Design and Behavioral questions also play a large role in the hire decision. I'll leave those topics for another article.

Resources

Articles

For more tips and perspectives, check out these articles from other members of the *Algorythm study group on Facebook*:

Three Lessons from my journey from a 2.8 GPA to a Senior Engineer at Netflix

How intentional academic mediocrity became one of my greatest strengths.

medium.com

Podcast

For more tips check out this podcast interview I did about getting your resume noticed by employers, prepping for technical interviews and negotiating offers.

Mastering The Programming Interview With Uber Engineer Esco Obong | newline

Today we're talking with Esco Obong who is an engineer at Uber and co-author of the our book JavaScript Algorithms In...

podcast.newline.co

Books

Cracking the Coding Interview 6th Edition

Elements of Programming Interviews

JavaScript Algorithms

Websites

GeeksforGeeks | A computer science portal for geeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and...

www.geeksforgeeks.org

Educative: Interactive Courses for Software Developers

We use cookies to ensure you get the best experience on our website. Please review our Privacy Policy to learn more...

www.educative.io

Learn Data Structures on Brilliant

The way we store and manipulate data with computers is a core part of computer science. In Data Structures, you'll...

brilliant.org

AlgoExpert | Ace the Coding Interviews

The leading platform to prepare for coding interviews. Master essential algorithms and data structures, and land your...

www.algoexpert.io

Know Thy Complexities!

Hi there! This webpage covers the space and time Big-O complexities of common algorithms used in Computer Science. When...

www.bigocheatsheet.com

Data Structures Algorithms Tech Interview Faang Silicon Valley

Bootcamps

The Best Technical Interview Prep Courses | Interview Kickstart

Technical interview preparation Bootcamp. Get trained by industry experts to land your next dream job.

www.interviewkickstart.com

Outco Interview Prep | Career Accelerator for Software Engineers

The software engineer career accelerator. The 4-week course develops core CS fundamentals and communication skills so...

www.outco.io

Practice

Self

LeetCode — The World's Leading Online Programming Learning Platform

At LeetCode, our mission is to help you improve yourself and land your dream job. We have a sizable repository of...

leetcode.com

HackerRank

HackerRank is the market-leading technical assessment and remote interview solution for hiring developers. Learn how to...

www.hackerrank.com

Assisted

Free and Anonymous Mock Interviews

Practice. Get Confident. Get hired

www.interviewbit.com

Practice Live Job Interviews — For Free

Pramp definitely played a role in my performance. Nothing beats mock coding interviews. I still remember my first...

www.pramp.com

All of the topics you need to know along with resources explaining them are freely available online. Google searching any of the topics mentioned in this article, even the obscure ones li "two pointers" will yield a wealth of information about them. You don't need a special colleg education, lucky genes or a long track record of success to get a job at a top tech company. T only thing you need is to dedicate time to consistent study and practice.

This article was inspired by a few related questions asked in the Facebook study group for black software engineers:

Facebook Groups

Data Structures, Algorithms & Architecture — Black Software Engineers has 1,549 members. Ther
is a huge racial gap in...

www.facebook.com

At the time of writing, we have 1,500+ black software engineers studying and helping each other with data-structures and algorithms. If you are an underrepresented person of color ir tech who needs help studying or is in a position to help others study, please join us! Otherwi please share the group with someone who has the time or needs the help.