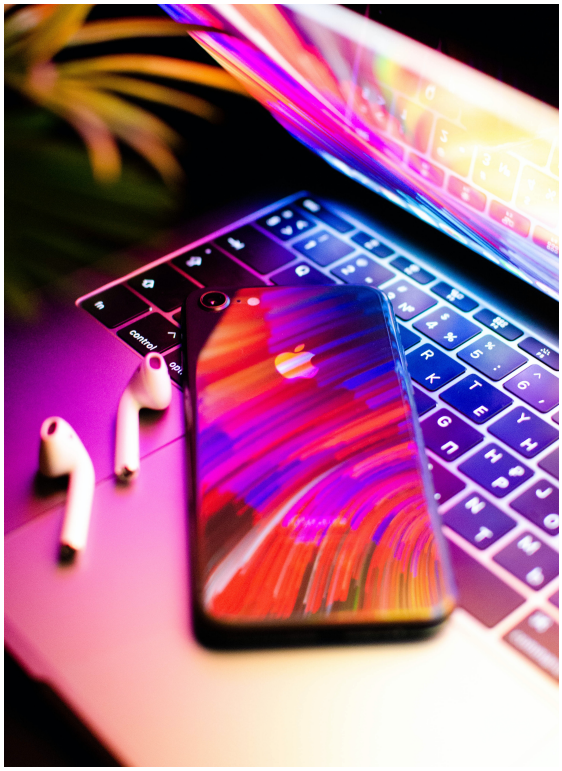
Andrew Codesmith

How to not suck at LeetCode (& DSA)

Chapters

About me



Andrew Codesmith

I learnt to code at the age of 32, whilst in another full time role. I got a law degree (which I never used), travelled the world, had jobs in bars, knocked on doors, in call centres and hospitals before I got into tech. I have since worked for large corporations, and start ups, whilst travelling the world working remotely. Before tech, I used to work as a recruiter, so coached and helped hundreds of candidates get jobs.
My content is about helping you get the role you want in tech, interview better and improve.

This is the guide I wish I had when learning LeetCode and Data Structures & Algorithms .

@andrewcodesmith

CHAPTER 01

Why...LeetCode

You can go your whole software career, and live a blissful stress free life without every having to learn or 'grind' LeetCode. There are many Software Development, AI/Machine Learning roles which don't ask for them (typically smaller companies). However, usually, for the very best paid roles in tech, you'll need them (the best paid roles are usually big tech/fintech, banks etc). So if you are avoiding it, you are limiting your career, and if you take it slow they aren't so bad (and can actually be fun). Secondly, they are the number one way to improve your programming skills in a short space of time, in my opinion. And you can solve pretty much any problem in the real world, if you can handle these. So let's get stuck in.

My LeetCode journey

When I first tried a LeetCode problem, it was a very humbling experience, as I realised I couldn't even do the most basic ones 💀

I'd taught myself to code, by watching tutorials. I could build things, but realised my problem solving wasn't great. Then LeetCode kept popping up again and again.

I then realised I was limiting myself by avoiding them, and that I'd be doing myself a disservice by not at least trying them.

I did some data structures and algorithms courses, started taking baby steps by solving problems on codewars.com (best free site to start you off on). Started off on the lowest level (8kyu), then whilst doing one to two of these a day, whilst learning the DSA knowledge, I started to feel more confident.

After a few months I went back to LeetCode I did FizzBuzz, targeted the array and string problems, and realised I could do them. I felt proud of myself, and haven't looked back. I then went through the topic roadmap (I'll share later), going through each topic in order, and took it step by step.

Then I added on mediums, and starting doing one each day at the start of each coding session. I then reached a stage where I could tackle a few hards. Whilst I'm not a LeetCode Jedi, I know enough and feel confident to navigate the job market with this knowledge.

And you can too. My page is all about helping you. When I started, I learnt everything online, and was really inspired by other tech creators sharing their knowledge. So that's the goal of my content, and resources like these (which I will continually share if you follow my account).
Anyway, lets get to it.

Start here

If you are a Computer Science student, you can skip this part. **Unless** you don't feel confident in your data structures and algorithms knowledge, then stay here.

To get good at LeetCode and get those $200k a year salaries you see those tech bros on social media talking about (they disclose their salary for 'transparency', but it's kinda crass in my opinion, but each to their own). You need to pass technical interviews, and to pass technical interviews you need to know data structures and algorithms. There is no way around this.

You could do array and string problems, but when you start getting to the more advanced data structures, you'll be screwed. But even just doing this, you'll really improve as a programmer.

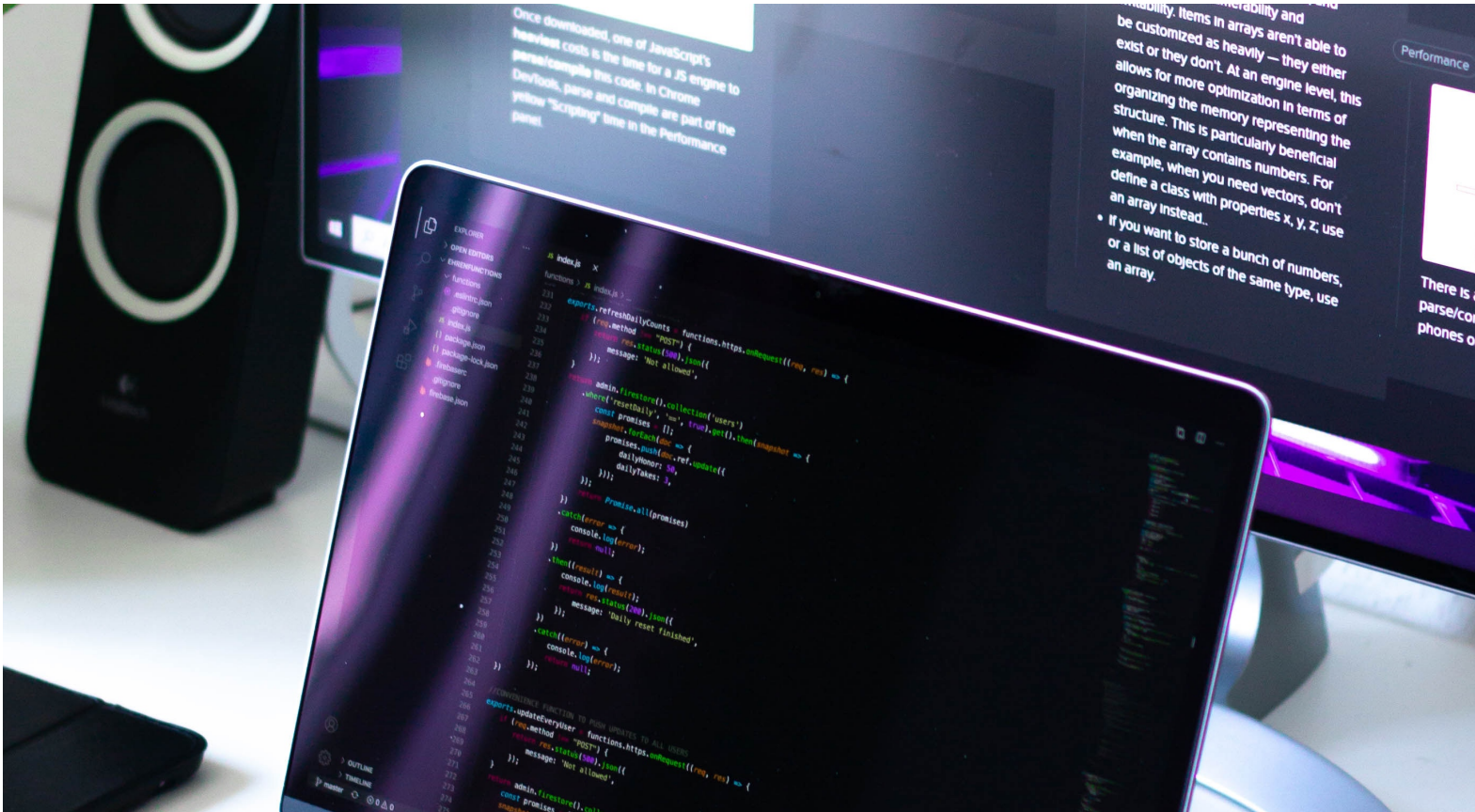
This guide is split into two seconds, Leetcode, and at the end Data Structures & Algorithms. This guide contains everything you need to get going and start preparing for technical interviews with some of the world's best tech companies.

You can still do problems whilst you learning DSA, but you'll struggle with the harder ones without the foundational DSA knowledge.

I will be making a dedicated 'How to not suck at LeetCode' video soon on my Youtube channel.

Subscribe here

My YouTube

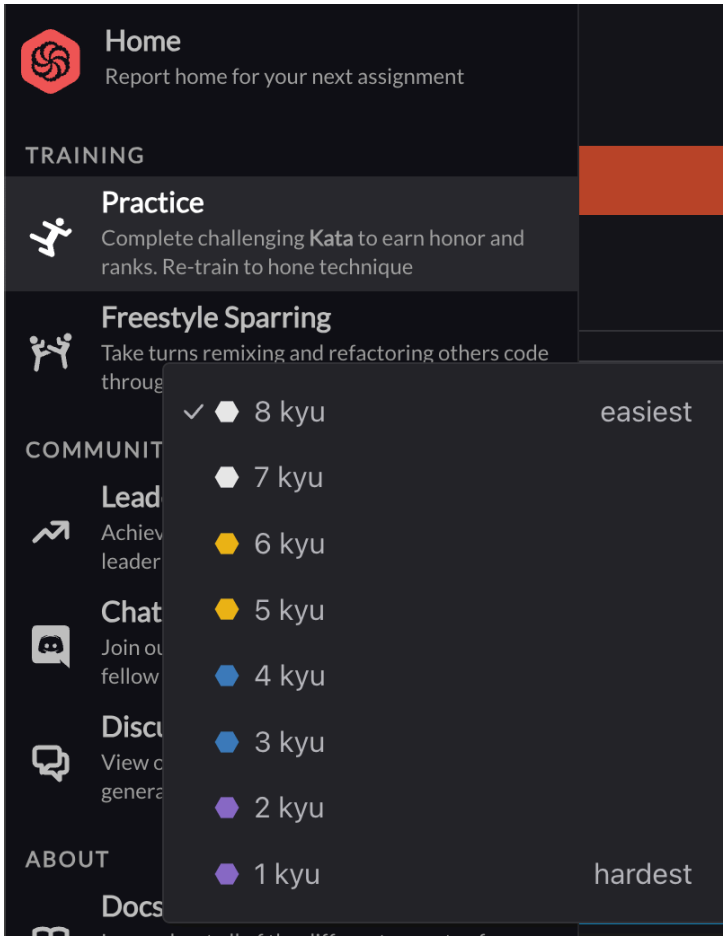


5 I can’t do easy’s!

If you can’t do easy leetcode problems. This is what I did to get past this, and you will, believe in yourself.

Start every programming session with a problem. Rather than coding along with tutorials, there is a time for that, but without the problem solving, you’re screwed.

I guarantee you will improve a lot faster by doing toy problems. They help you think like a programmer, and harness that problem solving ability in you. 🧠



When you are watching someone coding along, you are missing this.

1) **Google ‘Codewars.com’**
It’s like gymnastics for coders. Codewars is a gentle introduction to toy problems, later we’ll move onto LeetCode.

2) **Start with the easiest 8 kyu** 🎖️
Take it line by line. If you can’t solve a problem after 20mins. Look at the solution. Break it down, what is each line doing?

3) **Try again** 🏋️
Leave it till your next coding session and try it again. Until it clicks. Rinse and repeat. This process will slowly start to build something. In a few months you’ll do problems you simply couldn’t do before.

4) **Don’t feel bad looking at the solution** 😅
There comes a time when you are tying yourself up in knots because you have gone down the wrong path and have 3 nested loops. Take a break, look at the solution, and don’t be hard on yourself. You learn by learning the patterns. If you are trying these problems and failing, but still turning up and doing them again, you are a winner in my book.

4) **LeetCode** 🥇
Once you get to 6 kyu (ranking in Codewars) you can move onto LeetCode.

Short way

If you have a CS degree, and feel confident with DSA, then this is a good option. If you don't feel confident as a programmer, I recommend the slower approach. In particular if you are a self taught programmer. You have a long career ahead, play the slow game, and it's better for you mental health.

But here's the short way.

If you can afford it, and want to make one of the best investments of your life (getting good at leetcode, can earn you $ $$$$).

Buy a subscription to either

neetcode.io or
algoexpert.io

I've not taken either of these, and I'm not promoting these products for a fee. However, I have looked at the content, and it's very good, and concise. You can get good at leetcode without paying, but having the content in one place, with a clear path will be helpful for many.

Don't want to use those platforms? Just follow this roadmap -

https://neetcode.io/roadmap

Do blind 75

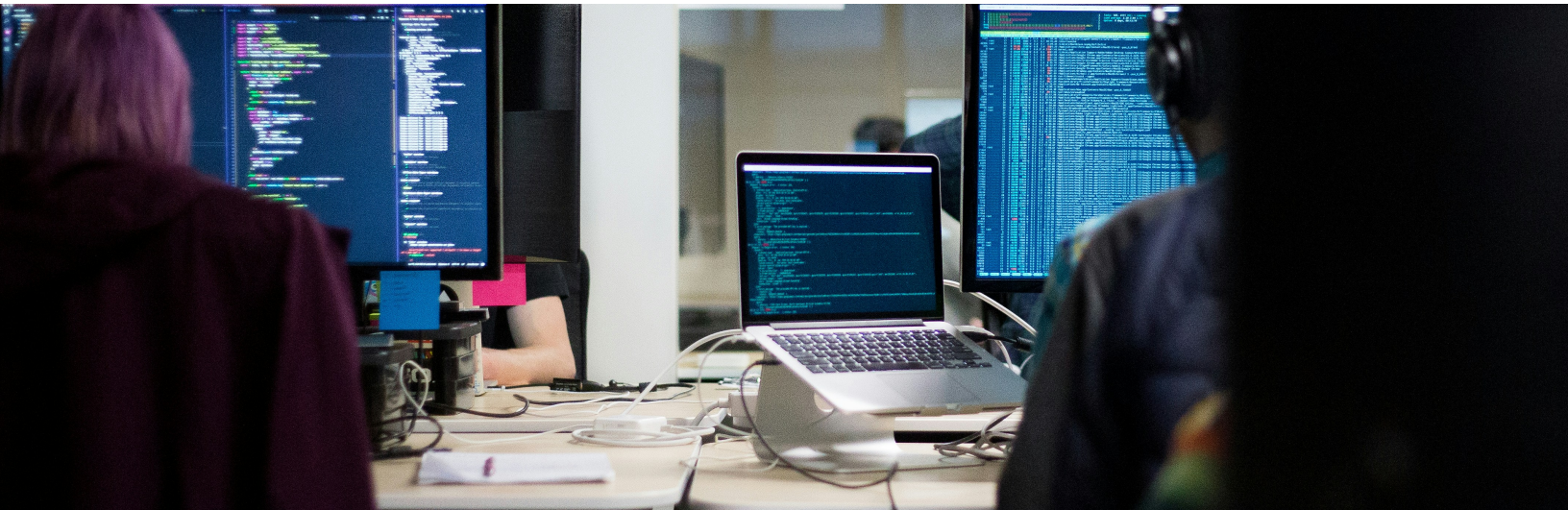
Do the blind 75, from top to bottom, and learn the content, data structures you need to at the time. You will be able to pass 95% of technical interviews, if you can pass all these

https://leetcode.com/discuss/general-discussion/460599/blind-75-leetcode-questions

Number 1 rule of leetcode - don't just do easy problems in a random order. You learn by discovering a pattern in the problem, which means if you can learn the pattern you can pass any problem, as it's the same problem disguised.

Only do medium/hard problems

Then do the "Leetcode top 150" list. The 75 questions will give you a nice practice for each topic while 150 will be a nice revisit (revision). For video solutions, check out NeetCode on YouTube. Lastly, you can move on to daily challenges or do company specific questions.



7 Long way (recommended)

Go through this topic list in order, do 15 - 20 problems on each one (do a few easy's then stick to mediums), only when you get a 'click', or feel confident with the underlying problem, do you move on.

Go to leetcode problems, and you can sort the problems by the topic.
1) String
2) Two Pointers
3) Sliding Window
4) Binary Search
5) Recursion
6) Binary Tree
7) Backtracking
8) DFS
9) BFS
10) Graph
11) Linked List
12) Sort
13) Trie
14) Stack / Queue
15) Priority Queue
16) Dynamic programming

Focus on top liked questions with the highest acceptance rate (a question is upvoted if its good, downvoted if it's confusing)

Again, the **Number 1 rule of leetcode** - don't just do easy problems in a random order. You learn by discovering a pattern in the problem, which means if you can learn the pattern you can pass any problem, as it's the same problem disguised.

You don't have to do 20, if things have 'clicked', you can move onto the next topic.

If you are still struggling, try different sources. Maybe try a book if videos aren't working, and constantly use your AI chatbot of choice, to explain things. It will gradually make sense. You will come to learn why linked lists can be better than arrays in some cases. Why hash tables are good to use, and this will come naturally with practice.

As I say all of this is learnable. If you are learning it for interviews, I will be doing more and more content on interviews. If you are learning it from your CS degree. The hard work now is going to pay off for you in the future.
This approach will take time, but you will be able to tackle any problem at any interview.

8 General Tips

1) Start by top 100 liked questions.

2) Focus on the questions by topic.

3) Order the questions from the highest to lowest acceptance rate. Focus on top liked questions with the highest acceptance rate (a question is upvoted if its good, downvoted if it's confusing).

4) Learn from the solutions - look at many, be curious and write them out yourself.

5) First ask about edge cases, write it out in pseudocode first, and talk to yourself - practice the real life scenario of interviewing by talking yourself through your own solution. 'I'm going to do this because….'. 'I think a linked list is a good idea because'. Write in comments what you want to do, even if you don't solve it, this looks good.

7) Mediums are the best bang for your buck. If you're doing too many easy's, you might not be getting the right questions which test your DSA knowledge. Whilst if you are doing hard's, you might be spending too long on them. Mediums are the best for learning.

8) Focus on getting the 'click'. If you solve lots of problems on the same topic, something will click and you can apply the same structure to a different problem.

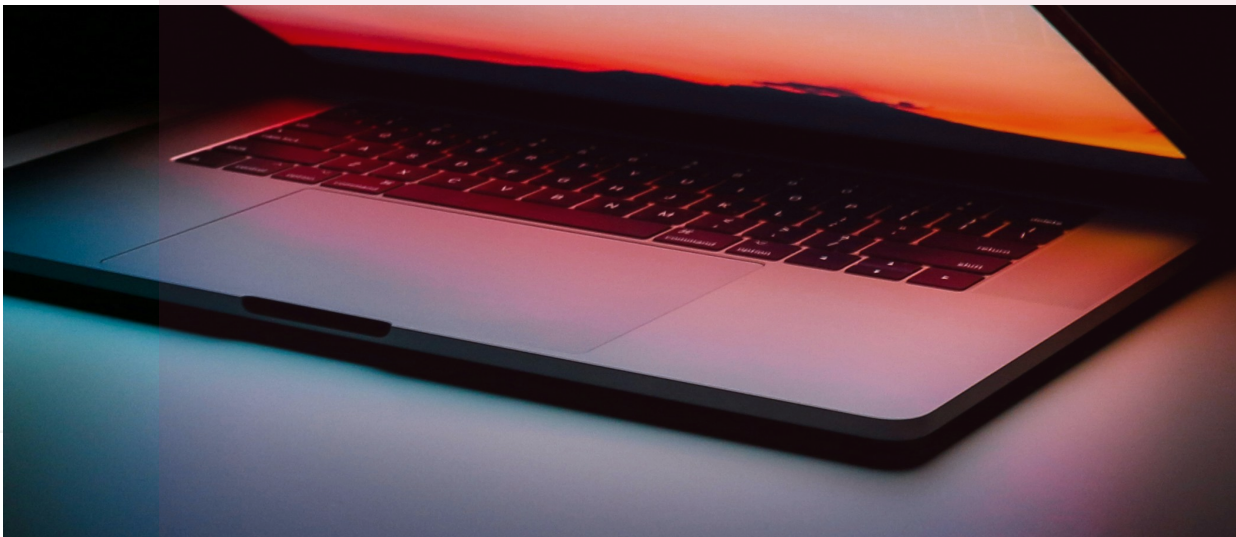
9) Don't just do random easy problems, you'll waste more time doing this, it'll take longer for the topic to click.

10) Revisit the problems (one or two weeks later until you are 100% confident to solve them).

11) For every problem you do, make sure you know the big O for it. If you aren't confident in big O yet, for every problem you finish, try to do it yourself, and if you struggle, put your code into our little friend chatGPT and ask if to explain the big O.

12) Try and do one at the start of every coding session you have. Within 6 months, your progress will be exponential.

Try and enjoy it, don't take it so seriously. Have fun, and laugh at how stupid you feel. Know that if you are trying these, you are pushing yourself, and growing.



Ok...Data Structures & Algorithms

You probably have a lot of insecurities running through your head right now. 'I'm not getting this'. 'I'm bad at maths'. 'Things just aren't clicking', 'This is too hard'.

Most people go through this and you have to just push through it. Whilst learning DSA looks super intimidating from the outside with terms like 'Big O', and 'time and space complexity'. Once you get a decent knowledge of these, trust me, these will become straight forward to you. But to get to that stage you have to go through a bit of discomfort and push yourself.

I think some people have more of a logical brain and will pick up DSA quicker. It makes sense to them intuitively. You don't? Welcome to the club. I'm completely self taught, and didn't code until my 30's. Worst case scenario is takes you a little longer, but it is all learnable. Let me repeat. It is ALL learnable.

3 years ago I started DSA, because I knew that if I could learn them. It would help me pass technical interviews and get opportunities working in the best paid roles in tech.

Now I feel confident in my knowledge of them, to pass on how I learnt them and what resources I used. What about the math? I'm average at math, and just picked up what I needed as I went but it wasn't much (just basic things like logarithms). Then I found myself reversing linked lists, and solving Leetcode problems. But I took it step by step, and used the following resources. I've done them in python and javascript.

The best language? That is the language you are more confident with. Lots of it is language agnostic. If you are starting from scratch, python is a good start with them. You are learning a very valuable skill in the world, it will take time.

Why you should learn DSA

Do you need to learn Data Structures & Algorithms to get a job as a developer? 100% no. If you want to work as a Front end Web Developer, you could skip it and come back once you have 6 months experience.

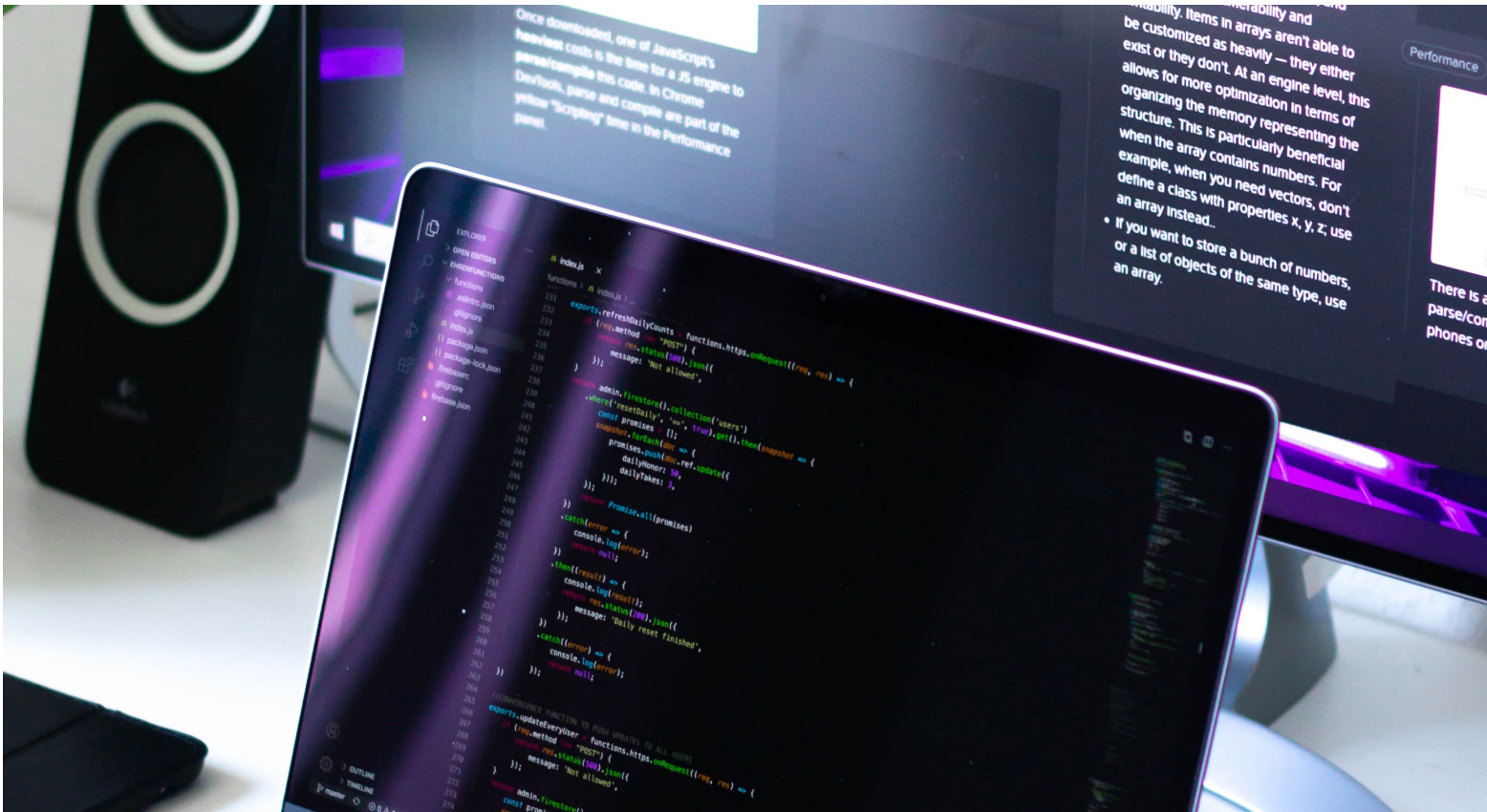
You can go your whole career without ever touching DSA. My first manager made web sites, and when I asked him about DSA, he grimaced…very successful, and never learnt them.

However, I think you should. Why? If you learn these, and get good at 'Leetcode', you will able solve any problem you are faced with. You will struggle, and then one day will feel like superman. They are the single best way to train that problem solving part of your brain.

But the biggest reason to learn them, is that once you reach a certain salary point, you will come across these in interview technical tests. If you can do them (which is a lot easier than you think, if you are patient, learn the concepts and give it a year). The world will be your oyster.

If you are actively avoiding jobs because they ask you these kinds of questions, you are inherently limiting yourself. So I'd just learn them, as to repeat, they aren't so hard, but look really intimidating from the outside (just like programming). Also just being able to answer simple questions and show a basic understanding of DSA will set you apart from other candidates.

Also if you are a CS student, you literally have to learn them for your degree



11 DSA resources

Before getting into the different data structures, you need to understand the foundational concepts. These are :-

Time Complexity, Space Complexity, Big O

Basically if you have an interview, and they give you a problem, which you then solve. These three terms allow you to describe how good your solution is.

Where to learn them

Here are some courses that will teach you all of DSA. Like the whole thing. I've highlighted which are paid and which are free.

Colt Steele - DSA Masterclass (paid)

The first course I took. He is a really beginner friendly instructor, and is aimed at self taught developers. It is in JS.

ZTM - Data Structures and Algorithms (paid)

This is another one in JavaScript. I'd say it's the more beginner friendly one. He explains data structures with really good metaphors.

Use the roadmap on this site to learn DSA

Geeks for Geeks (free)

Great written free resource. This has everything you need and goes through every DS. If you don't want to spend money, I would use this. Then if there's something I don't understand, I'd use the YouTube videos I mention later.

Books

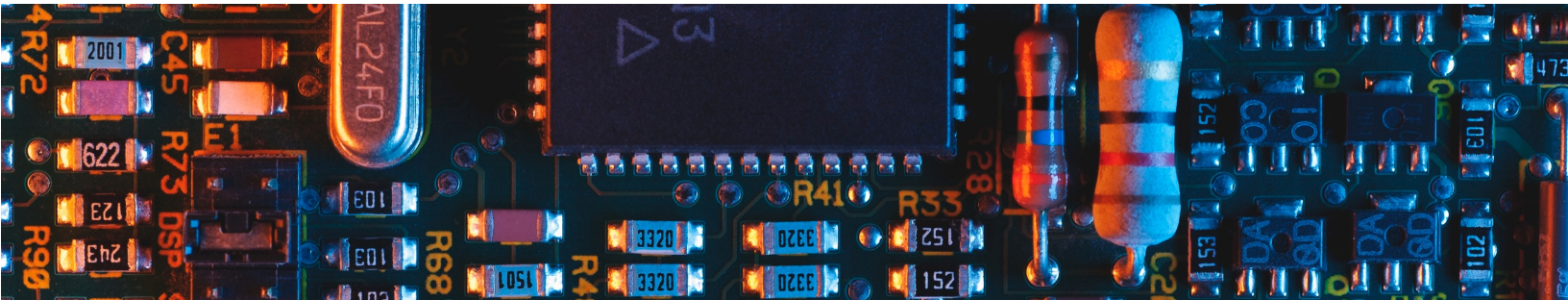
If you prefer books to video tutorials, I recommend this for complete beginners.

Grokking Algorithms - as a self taught web developer, looking to make the switch into Software engineering/development, I picked this up a few months into my first developer job.

Whilst I'm sure data structures and algorithms come naturally to some people. If you are a visual learner this will give you a foundational start in DS&A's

If you're struggling with DS&A's, this is your day one. Then move onto books like :_

Algortihms



More resources

DSA in Python (free)

Another good free resource with written and video content.

Best free intro to DSA video I found YouTube

Explains concepts really well, its in python, but is language agnostic. But it is visual, and will get you started with all the foundational concepts, big O and the starting data structures.

Big O cheatsheet (free)

Explains it very simply with examples of all the different types of complexity you'll come across

Still struggling with concepts?

Watch this guys videos, short videos on each concept explained simply

Still not getting it?

Bro code is another no nonsense channel with lots of concepts explained simply.

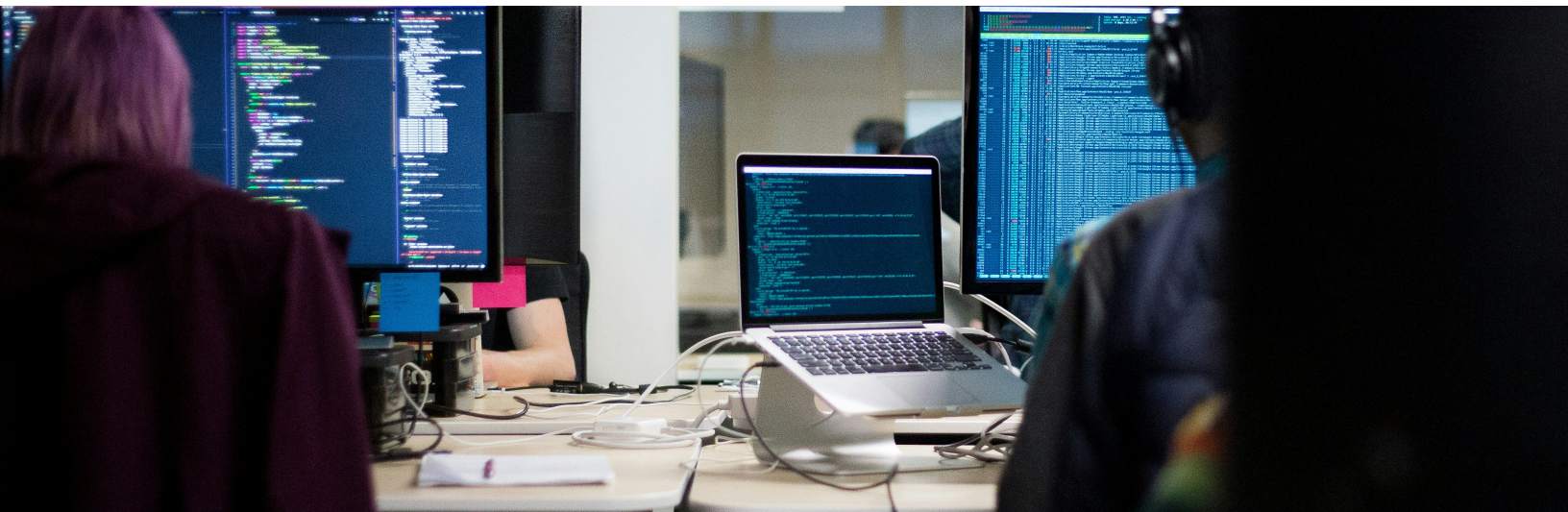
I understand the theory, but I need visual examples to understand it.

These three sites are the best sites for visualising what is going on. That's your next step if it's not clicking, slowly writing a solution out, and going through what is happening with each iteration (recursion I'd say is hard to understand, until you write it out yourself, constantly write things out and be curious as to what's going on. That curiosity, rather than being frustrated is a great mindset shift for top programmers)

1) **Visualgo** the best way to learn. Go through a linked list, stop it. What's going on with the pointers? Play it again, and watch the visualisation. Incredible resource

2) **cs.usfa** next best one.

3) **bost.ocks.org/mike/algorithms/** - this one has some of the most beautiful, almost artistic visualisations of DSA. It will be a nice break from you, and has good visuals on sorting algorithms.



I'm struggling with DSA

If you're still not getting it

This is the **best** GitHub DSA resource I've found. It has cheatsheets, online books, and even more courses etc

Start with the easiest data structures and don't move on until you are comfortable with it. Write it out again and again. It will click. You just have to sit with it, and be patient. Linked lists/pointers took me a while, and one day, they just made sense.

I understand stuff, but wanna go deeper (free)

Then you to one of the Indian programming gods of Youtube. Abdul Bari, and his famous Algorithms series on YouTube.

Not taken this one, but it's free and looks pretty professional

CS50 (free)

It wouldn't be a complete without this. If you are starting out with coding, this is arguable the best free course out there.

It is the starting point for millions of people, and it goes into DSA and why it is important. Explained with visual examples and lectures. Also it's from Harvard.

If you are still struggling, try different sources. Maybe try a book if videos aren't working, and constantly use your AI chatbot of choice, to explain things. It will gradually make sense. You will come to learn why linked lists can be better than arrays in some cases. Why hash tables are good to use, and this will come naturally with practice.

As I say all of this is learnable. If you are learning it for interviews, I will be doing more and more content on interviews. If you are learning it from your CS degree. The hard work now is going to pay off for you in the future. Tech is changing a lot, and whilst the roles are likely to evolve over the next few years. AI and tech are going to become a bigger and bigger part of our world, which means more opportunities for anyone who works in the software space.

That's a wrap

I hope this helped and thanks for supporting my page.

Also if you liked this, I have a guide on 'How to get your dream tech job' I put a lot of time in. Basically everything you need to know to get a job in tech. Buy it if you'd like, **here.** My goal is to help people like you, so I'll continue to make more free guides like this, so follow for more.

Also **I will be making a dedicated YouTube video** on this exact topic very soon.

Subscribe here My YouTube

I may never meet you. But I'm rooting for you.

https://beacons.ai/andrewcodesmith

A