Open in app ↗ Sign up Sign in

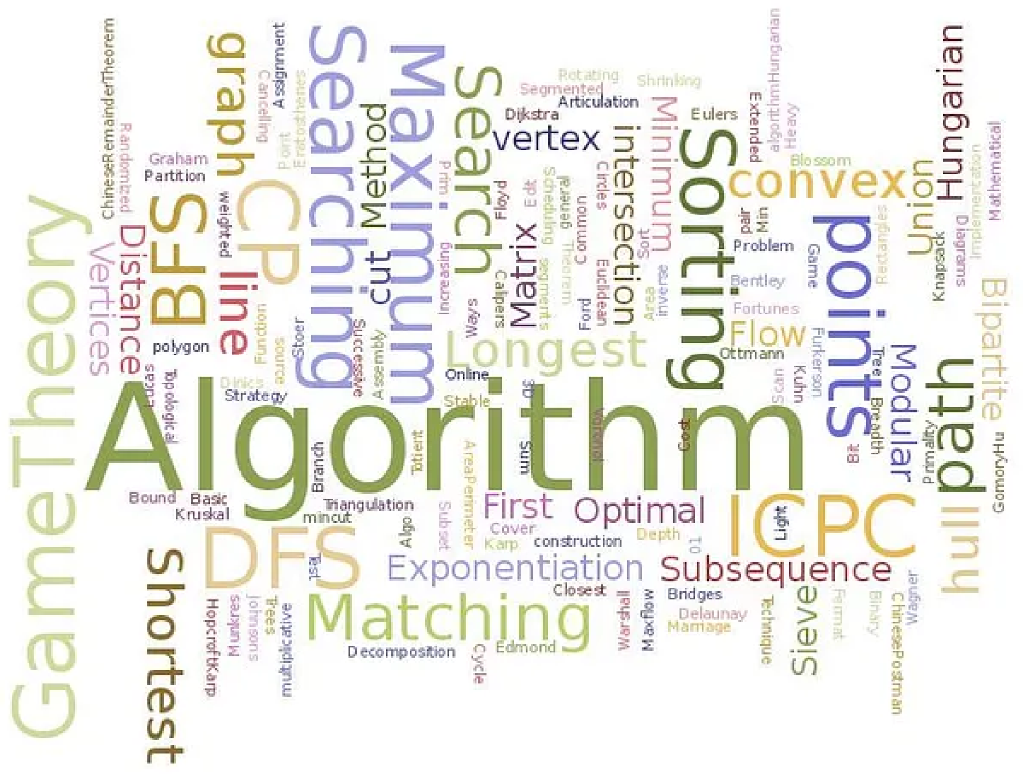
Medium Search

An Algorithmic Approach to Solving Algorithms

Avoid the panic of having no clue where to begin and approach every problem brimming with confidence

Matt Werner · Follow
Published in The Startup
7 min read · Oct 22, 2019

Listen Share



The way I see it, solving algorithms is a lot like golf. Golf, as they say, is 90% mental. You still need to practice your physical skills a whole heck of a lot, but having a solid game plan, executing it as best you can, and responding well in the face of adversity is how you truly succeed. I believe solving algorithms is much the same, especially when it comes to doing so in an interview.

Imagine this. You're in an interview, and the interviewer asks you to solve a problem. You've practiced so much for this, but you look at this particular problem and have absolutely no clue where to begin. Panic hits. *Shoot, what do I do*. You stare blankly at the board or screen. A massive mental block makes you forget everything you thought you knew. *Do I need to write a function?* Somehow you have forgotten how to even write a function. *Wait, do I need the word function? Do I need parentheses?* Imposter syndrome sets in, and you start to question why you're even there in the first place.

My guess is that pretty much every aspiring programmer fears this scenario at one point or another, myself included. But my belief is that **with a good game plan, this situation can be avoided altogether**. Through practice, I have developed my own game plan that I want to share. I am not here to say it's the best approach out there, but it helps me persevere when I feel like I don't know what to do. I hope it can help others as well.

So without further ado, here is one humble programmer's algorithmic approach to solving algorithms.

Overview

Here is my step-by-step process. Each step is explained in detail below.

1. Make no assumptions
2. Determine the full scope of the problem
3. Think of a few strategies that *might* work
4. Find reasons why your strategies *will not* work
5. Try your strategy for specific cases, then generalize
6. Repeat steps 3–5 until success
7. Internalize the solution

Step 1: Make no assumptions

When presented with a problem, **never assume you have done it before.** Approach it at first like it is completely new to you, even if it seems extremely similar to something you have solved before. Sometimes tasks can seem very similar but require very different solutions. Assuming you know what to do because you think this problem is the same as another can set you up for failure before you even begin. (Eventually we will compare the problem to similar ones we have solved, but not yet.)

Step 2: Determine the full scope of the problem

When I think about the “scope” of the problem, I think about three things:

1. **What are the inputs?** Am I working with an integer? A string? An array of arrays? Something else?

2. **What is the output?** What will a correct result look like? **Be as specific about this as possible.** Just saying *I need to make an n x n matrix of integers from 1 to n*^{2} is not going to be very helpful. From my own experience*, I might say something like, *I need to make an n x n matrix of integers from 1 to n*^{2}*. This means an array with n elements, where each element itself is an array of n integers. And I need to populate each individual element in the matrix with one of the numbers from 1 to n*^{2} *based on that element’s position in the matrix, with the top-left element starting at 1, and then each element increasing by one as I go clockwise around the edge of the matrix, spiraling inward until the last integer n*^{2} *is placed in the middle of the matrix.* Do not take for granted that you know what the output is supposed to look like just because you heard or read the problem one time.

3. **What tools are at my disposal?** This is maybe less crucial than the first two, but especially in an interview setting, you might get limited or told not to use certain methods. For example, maybe you need to reverse an array, but the interviewer specifically said not to use `.reverse()`. Make sure up front that you know what you can and can't use so you don't back yourself into a corner later on.

Solving an algorithm is kind of like solving an encoded message. With an encoded message, you need the code’s key. Otherwise, you are stuck looking at nonsense. When solving an algorithm, if you don’t know where to begin, it can feel like you have the encoded message without the key and are looking at nonsense. But this step is your key. And **it might seem like an obvious step, but do not take it for granted.** Even if you have no clue how to solve the actual problem, you can at least know the inputs and outputs. (If you are not sure, clarify with your interviewer!) Getting this information is a small accomplishment. It’s *progress*. And that gives you some momentum. Instead of thinking, “Oh no, I have no idea what to do!” you will think, “Okay, I know x, y, and z. I am off to a good start. What are the next steps?”

Step 3: Think of a few strategies that might work

At this point, you still may not be sure how to turn the input into the output you need, and that's okay. No need to panic because you have done your practice. And unlike in step 1, this is the time to think about how this problem might be similar to others you've solved before. Given the information you have from step 2, think about a few coding strategies that have worked for similar problems. Then take these strategies, and for each one, try to tell yourself a story about why that strategy might work. If you can come up with a story, perfect. Keep that strategy handy. If you can't, drop that strategy. At this point, maybe you will see exactly what you need to do, but if not, no problem! Bring your viable strategies to the next step.

Step 4: Find reasons why your strategies will not work

Look at your possible strategies and try to come up with a reason why each of those strategies *won't* work for this specific problem. If you have practiced enough, you will be pretty good at spying future roadblocks. If you can't find a reason against a particular strategy, take that strategy to the next step and try to implement it. And don't worry if you aren't 100% sure about it. **It's okay if that strategy ends up being wrong.** You don't have to come up with the perfect solution on your first try. A wrong strategy is a great opportunity to show off some of your character, as how you handle a failed attempt will say a lot more about you than if you just have the solution memorized. And you can always come back to this step if your strategy ends up not working out. So take your strategy and move on with confidence to step 5.

Step 5: Try your strategy for specific cases, then generalize

Okay, you have a strategy. Now try to apply it to some specific cases. If the input for the problem is a number n, maybe try solving for 3 first. Then try 4 and 5. As you do this, one of two things will happen. Either you will realize your strategy isn't viable, or you will start to see a pattern emerge that will work in all cases. If you find the pattern and can generalize it into a full solution, congrats! If not, don't worry. Stay confident. You got this. You just need to try a new strategy.

Step 6: Repeat steps 3–5

It feels a bit cliché for one step to just be repeating other steps, but solving an algorithm is not always a straight line. There is a decent chance you hit a roadblock on your first attempt, especially if the specific problem is unlike any you have seen before. If this happens, don't panic. **Embrace the roadblock**. As I mentioned before, how you handle this will say a lot about you both as a person and as a coder.

Acknowledge that the current strategy is not going to work, regroup, look at your other strategies from step 3 that you thought *might* work, and continue from there. And as always, stay confident. Every wrong strategy brings you one step closer to finding the right one. And in the end, you will find that right one.

Step 7: Internalize the solution

You did it! You truly are an algorithm master. But don't get too cocky. There are more problems out there to solve. Look at what you did here, soak in what did or didn't work, analyze why that was the case, and turn this experience into another tool in your problem-solving tool belt.

Summary

As I mentioned at the top, in no way do I believe this strategy is better than anyone else's. But it does work for me, so maybe it will work for you as well. Just don't use this strategy (or any other) as an excuse to not practice. Preparation is key. So work hard, put your time in, solidify your game plan, and get to solving!

**This example is a reference to the matrix spiral problem in* The Coding Interview Bootcamp: Algorithms + Data Structures, *a course on* Udemy *created by Stephen Grider. I highly, highly recommend this course to anyone and everyone. Mr. Grider does a spectacular job of explaining concepts and walking through solutions (often more than one) for each exercise. (Thanks Mr. Grider!)*

Programming Algorithms Strategy Interview Problem Solving

Follow

Written by Matt Werner

35 Followers · Writer for The Startup