Preface

Graduating with a Computer science or engineering degree? Converting from physics, or math, or any unrelated field to computer science? Dreaming of getting a job as a software engineer in game-playing companies such as Google, Facebook, Amazon, Microsoft, Oracle, LinkedIn, and so on? Unfortunately, there are the most challenging "coding interview" guarding the door to these top-notch tech companies. The interview process can be intimidating, with the interviewer scrutinizing every punch of your typing or scribbling on the whiteboard. Meanwhile, you are required to express whatever is on your mind to walk your interviewer through the design and analysis process and end the interview with clean and supposedly functional code.

What kind of weapons or martial arts do we need to toughen ourselves up so that we can knock down the "watchdog" and kick it in? By weapons and martial arts, I mean books and resources. Naturally, you pull out your first or second year college textbook *Introduction to Algorithms* from bookshelf, dust it off, and are determined to read this 1000-plus-pages massive book to refresh your brain with data structures, divide and conquer, dynamic programming, greedy algorithm and so on. If you are bit more knowledgeable, you would be able to find another widely used book–*Cracking the Coding Interviews* and online coding websites–LeetCode and LintCode–to prepare. How much time do you think you need to put in? A month? Two months? Or three months? You would think after this, you are done with the interview, but for software engineers, it is not uncommon to switch companies frequently. Then you need to start the whole process again until you gain a free pass to "coding interviews" via becoming an experienced senior engineer or manager.

I was in the exact same shoes. My first war started in the fall of 2015, continued for two months and ended without a single victory. I gave up the whole interview thing until two years ago when my life (I mean finances)

situation demanded me to get an internship. This time, I got to know LeetCode and started to be more problem and practice driven from the beginning. 'Cause God knows how much I did not want to redo this process, I naturally started to dig, summarize or create, and document problem-patterns, from sources such as both English and Chinese blogs, class slides, competitive programming guideline and so on.

I found I was not content with just passing interviews. I wanted to seek the *source* of the wisdom of algorithmic problem solving–the principles. I wanted to reorganize my continuously growing knowledge in algorithms in a way that is as clear and concise as possible. I wanted to attach math that closely relates to the topic of algorithmic problem solving, which would ease my nerves when reading related books. But meanwhile I tried to avoid getting too deep and theoretical which may potentially deviate me from the topic and adds more stress. All in all, we are not majoring in math, which is not ought to be easy; we use it as a practical tool, a powerful one! When it comes to data structures, I wanted to connect the *abstract* structures to real Python objects and modules, so that when I'm using data structures in Python, I know the underlying data structures and their responding behaviors and efficiency. I felt more at ease seeing each particular algorithm explained with the source principle of algorithm design–*why* it is so, instead of treating each as a standalone case and telling me "what" it is.

Three or four months in midst of the journey of searching for answers to the above "wantes", the idea of writing a book on this topic appeared in my mind. I did not do any market research, and did not know anything about writing a book. I just embarked on the boat, drifted along, and as I was farther and deeper in the ocean of writing the book, I realized how much work it can be. If you are reading this sometime in the future, then I landed. The long process is more of an *agile* development in software engineering; knowledge, findings, and guidelines are added piece by piece, constantly going through revision. Yet, when I started to do research, I found that there are plenty of books out there focusing on either teaching algorithmic knowledge (*Introduction to Algorithms*, *Algorithmic Problem Solving*, etc) or introducing interview processes and solving interview problems(*Cracking the Coding Interview*, *Coding Interview Questions*, etc), but barely any that combines the two. This book naturally makes up this role in the categorization; learning the algorithmic problem solving by analyzing and practicing interview problems creates a reciprocal relationship–creating passion and confidence to make 1+1=4.

What's my expectation? First, your feeling of enjoyment when reading and practicing along with the book is of the upmost importance to me. Second, I really wish that you would be able to sleep well right the night before the interview which proves that your investment both financially and timewise was worthwhile.

In all, this is a book that unites the algorithmic problem solving, Coding

Interviews, and Python objects and modules. I tried hard to do a good job. This book differs from books focusing on extracting the exact formulation of problems from the fuzzy and obscure world. We focus on learning the principle of algorithm design and analysis and practicing it using well-defined classical problems. This knowledge will also help you define a problem more easily in your job.

Li Yin

Li Yin
`http://liyinscience.com`
8/30/2019

Acknowledgements