# Introduction

Ethan Arnold

March 2016

## 1 Introduction

### What is competitive programming?

Competitive programming is an activity in which competitors try to write efficient solutions to programming problems in a timed contest. A contest usually lasts between 2 and 5 hours, and there are usually between 5 and 10 problems to be solved. For each problem, contestants submit a program that reads input and produces output meeting the problem specification.

Some people praise competitive programming as a means of preparing for coding interviews. While it is true that good competitive programmers are almost always good interviewers, competitive programming has much more to offer than the relatively flat world of interviewing. For this reason, I will focus on the contest aspect.

### How to read these tutorials

These tutorials provide what I would consider an accelerated introduction to competitive programming. If you read every word and look over all the solutions, you will end up with no more knowledge than you have now. You must participate actively in the learning process.

Each tutorial starts with a motivating problem. I encourage you to read this problem and think about how you would solve it. After working on it for 10-15 minutes, you can read the discussion that follows. In the discussion portion, I attempt to explain the technique and apply it to the motivating problem. As you read this part of the tutorial, you should continue thinking about how you would implement each part of the solution as it is explained. After the full solution is explained, try to code it yourself before looking at my solution.

These motivating problems are often difficult. For example, the problem I use in the backtracking tutorial is not as straightforward as one might expect given its position in the series of tutorials. This is not meant to discourage you, but to challenge you. The topics I cover range from trivial to quite difficult, and if you are going to try to read all of these tutorials in one sitting, you will become hopelessly lost somewhere in the middle. If you actually want to improve, you need to take the chapters one at a time. For each chapter, do all the practice problems listed. Even if you think you already know a topic, do the practice problems anyway–if you know the topic that well, they shouldn't take more than a few minutes!

### How to practice

Reading is not enough to become a better competitive programmer. The practice problems listed in each chapter here are a good start: you should read the problem statement, come up with a solution, code the solution, and submit it. If you get wrong answer, don't look at the test case you failed! You won't get that luxury in a competition, and debugging with such limited information is an important skill to learn. Carefully reread your solution and think about edge cases. Write a few test cases and work them out by hand; if you're lucky, you'll find one that your solution fails.

If you can't determine the source of your problem after an hour of debugging, feel free to take a break. Come back in a day or two and see if you can find the bug.

On the other hand, if you read a problem and can't even figure out where to begin, don't panic. Keep thinking about it, and consider how you might apply the technique from the chapter. If you come up with an idea but you're unsure of its correctness, I encourage you to write an informal proof. Especially in greedy problems and at higher levels of competition, the solution is not obvious and a proof may be required to convince yourself that it is correct.

Now, solving all the linked problems is a good start. But if that's all the practice you do, you'll walk into a competition and have no idea where to start. You need to practice solving problems without the context of having just read a tutorial. For this, I recommend going on Codeforces and clicking on random problems (make sure you disable "show tags for unsolved problems" in your settings) and solving them. Other sources of practice problems include UVa online judge, Kattis, SPOJ, HackerRank, CodeChef, and Topcoder.

The last element of good practice is to actually compete. Codeforces holds regular competitions (although some of them are in the middle of the night) which generally have very good problems. HackerRank, CodeChef, and Topcoder are also popular contest platforms.

## Acknowledgments