MIT 6.045J - Automata, Computability, Complexity
Lecture 1 - Introduction

Lecture 1 - Introduction

What is Computer Science?

This course justifies why computer science deserves to be called a "science". It doesn't give you any skills that you can apply in your daily job, or your projects, but it introduces you to some Great Ideas in Computer Science.

It starts with a set of rules that we assume to be true, just as we assume axioms to be true in mathematics, and shows what we can and cannot do with this set of rules. It shows us the limits of what is possible and what is definitely impossible.

Factoring is Hard

The first "idea" we look at is that factoring a composite number into its prime factors is hard, at least with our current computational model. Factoring will be easy with a quantum computer. For now, the best we can hope to do is spend exponential time finding the factors, and the best algorithm we know takes $ O(2^{cN/3}) $ time to find the factors, where N is the number of digits in the number.

However, checking if a number is prime or composite is much easier - possible to do in polynomial time. This is called primality testing, and we have primality testing algorithms that run in around $ O(n^3) $ time.

These two ideas are the foundation of today's cryptographic systems, and if we are able to build a true quantum computer, that will basically be the end of all privacy.

Euclidean Geometry

Compass and straightedge geometry can be considered the first model of computation in theoretical computer science. The course explores this model as an example of what we will be doing further along with other, more modern models.

The compass and straightedge geometry model says that given any two points on a plane, we can perform the following operations on them -

1. Draw a line through them
2. Bisect the line
3. Extend the line arbitrarily
4. Draw a circle with the line segment as a diameter

Furthermore, you can mark the intersections of any two curves as a new point, and use that point for further constructions.

Using this model of computation, we can reduce it to the following model -

Starting with the numbers 1 and 0, perform the following operations on them -, +, x, ÷, and come up with new numbers. The limits of this model are then seen in the numbers (or types of numbers) that we can't come up with. For example, we can come up with 2 by using the pythagorean theorem in the plane. However, we can't come up with 32 using just 1s, 0s, and the above operations. The proof of this requires Galois theory, but it is a proven limitation.

This example is more mathematical than computational, but this illustrates the type of reasoning we perform in this course. We start with a set of well defined rules, and then extend them to find out exactly what we can and cannot do, as well as give reasons as to why we definitely cannot do something, regardless of how clever we are or how much computational power we have.