INTRODUCTION

What are Computer Scientists?

Computer scientists are more than just programmers. For example, you must be able to:

Recognize patterns in a new problem

- Example: "Predicting stock market trends is like curve fitting and extrapolation."
- This requires abstraction, and familiarity with known patterns.

Identify options for solving a problem

- Example: "A sorted array might be a good data structure for storing a set of items."
- This requires creativity, and familiarity with known solutions.

Weigh options

- Example: "Keeping the array sorted allows faster search, but not keeping it sorted allows faster insertion."
- This requires the ability to analyze potential solutions for correctness and efficiency.

Make an appropriate choice for a given situation

- Example: "Because I expect to do more searching than insertion, a sorted array is better."

Notice that none of this is about programming.

The Software Lifecycle

You are probably most familiar with programming, but that is only a small part of the full life cycle of software, which includes:

- Specification: stating precisely *what* the program must do.
- Design: deciding *how* the program will do it — data structures, algorithms, and software "architecture".
- Implementation (programming): obvious.
- Testing: checking that the program meets the specifications.
- Release: distribution and installation of the software.
- Maintenance: fixing errors, handling new requirements, accommodating resource changes, etc.

(There are many models of how these stages relate and in what order they should happen.)

There is a tendency to think that programming is the central activity, and that most errors are introduced at that stage.

In fact, errors happen at all stages. The sooner they are found, the cheaper they are to fix*:

Stage when error is caught	Cost to fix it
design	1.0 units
just before testing	6.5
during testing	15
after release	60–100

Moral: Pay close attention to design and testing. "The sooner you start coding, the longer it will take to finish."

And most of the effort spent on a program is in its maintenance.

Moral: Design, document, and test your code so that it will be easy to maintain.

*Pressman, *Software Engineering: A Practitioner's Approach*, 1992, p. 559.

What This Course is About

This course is an introduction to the *science* of computing. It assumes that you already know some programming.

We'll work on the skills you need to be a computer scientist, able to contribute to all parts of the software lifecycle. This will include:

Specification:

- Writing precise specifications.

Design:

- Looking at a problem abstractly.
- Knowing standard abstractions that have proven useful in computer science, and how to use them.

- Some new data structures that offer alternative ways to implement an abstract idea.
- Analyzing the efficiency of an algorithm.
- Using proofs to establish facts about data structures and algorithms.

Implementation:

- Knowing the properties of a good program.
- Designing a program to have these properties.
- Writing code that uses a new programming technique: recursion

Testing:

- Choosing a systematic and thorough set of test cases.
- Documenting testing so that it will be convincing.