SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2025

Topic Notes: Introduction and Overview

Welcome to Design and Analysis of Algorithms!

What is an Algorithm?

We explore the idea of an algoritm and important properties in class with a separate handout.

Why Study Algorithms?

The study of algorithms has both *theoretical* and *practical* importance.

Computer science is about problem solving and these problems are solved by applying algorithmic solutions.

Theory gives us tools to understand the efficiency and correctness of these solutions.

Practically, a study of algorithms provides an arsenal of techniques and approaches to apply to the problems you will encounter. And you will gain experience designing and analyzing algorithms for cases when known algorithms do not quite apply.

We will consider both the *design* and *analysis* of algorithms, and will implement and execute some of the algorithms we study.

We said earlier that both time and space efficiency of algorithms are important, but it is also important to know if there are other possible algorithms that might be better. We would like to establish theoretical *lower bounds* on the time and space needed by any algorithm to solve a problem, and to be able to prove that a given algorithm is *optimal*. We would also like to be able to prove that some things are *impossible*!

Some Course Topics

Some of the problems whose algorithmic solutions we will consider include:

- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Primality testing
- Traveling salesman problem

- Knapsack problem
- Chess
- Towers of Hanoi
- Sorting
- Program termination

Some of the approaches we'll consider:

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Greedy approach
- Dynamic programming
- Backtracking and Branch and bound
- Space and time tradeoffs

The study of algorithms often extends to the study of advanced data structures. Most should be familiar; others might be new to you:

- lists (arrays, linked, strings)
- stacks/queues
- priority queues
- graph structures
- tree structures
- sets and dictionaries

Finally, the course will often require you to write formal analysis including some proofs.

Pseudocode

We will spend a lot of time looking at algorithms expressed as *pseudocode*.

Unlike a real programming language, there is no formal definition or standard "dialect" of "pseudocode". In fact, any given textbook is likely to have its own style for pseudocode.

Our text has a specific pseudocode style. I will aim to approximate the book's style, but sometimes my own style might drift to look more like Java or C code. Please try to do the same when you write pseudocode. It doesn't have to match the text exactly, but should be close.

The book's dialect:

- omits variable declarations
- indentation shows scope of `for`, `if`, and `while` statements (no curly braces!)
- arrow ← used for assignment
- single = for equality comparison
- `//` used for comments
- no semicolons!

A big advantage of using pseudocode is that we do not need to define types of all variables or specify complex structures.