csce350 — Data Structures and Algorithms
Fall 2019 — Lecture Notes: Introduction

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with occasional supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 What is an algorithm?

1.1
2019-08-22

An *algorithm* is a sequence of unambiguous instructions for solving a problem, that is, for obtaining a required output for any legitimate input in a finite amount of time.

2 Who cares?

2019-08-27

Why is this course worth such a big investment of your time?

- **Theoretical importance**: A rigorous way to study computational processes.
- **Practical importance**:
  - faster programs
  - correct programs
  - a "toolbox" of algorithms for common problems
  - a "playbook" of strategies for new problems
  - an understanding of what happens "under the hood" when you use a library or API.

3 Example problem: Greatest common divisor

Input: Two non-negative integers m, n.

Output: The largest integer that evenly divides both m and n.

4 Example algorithm: Euclid's algorithm

We could prove **two identities**:

1. For any integers $m > 0$ and $n > 0$,

$$\gcd(m, n) = \gcd(n, m \bmod n).$$

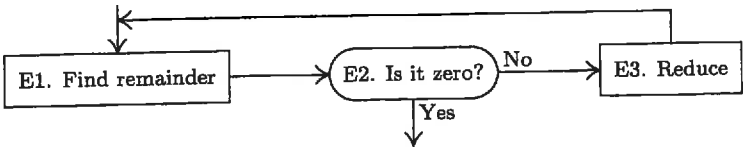
2. For any integers $m > 0$,

$$\gcd(m, 0) = m.$$

For example, we could use these identities to compute $\gcd(42, 12)$ *like this:*

$$\gcd(42, 12) = \gcd(12, 6) = \gcd(6, 0) = 6$$

5 Option 0: Flowchart



6 Option 1: "Structured" English

Algorithm E (*Euclid's algorithm*). Given two positive integers m and n, find their *greatest common divisor*, that is, the largest positive integer that evenly divides both m and n.

E1. [Find remainder.] Divide m by n and let r be the remainder. (We will have $0 \leq r < n$.)

E2. [Is it zero?] If $r = 0$, the algorithm terminates; n is the answer.

E3. [Reduce.] Set $m \leftarrow n$, $n \leftarrow r$, and go back to step E1. ∎

7 Option 2: Pseudocode

```
EUCLIDGCD(m, n)
  while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
  end while
  return m
```

8 Guidelines for good pseudocode

DO:

1. Use a descriptive title.

2. Show parameters clearly.
3. Use math notation.
4. Indent.
5. Show indentation with vertical lines.
6. Close loops and conditionals.
7. Use $\leftarrow$ for assignment.
8. Return a value.

DON'T:

1. Declare variables: `int`, `double`, etc.
2. Use goofy punctuation: `++ -- && || ; % { }`

> *Students sometimes ask for a detailed list of the "rules" for pseudocode. It would be impossible to produce this kind of list, for reasons that might be obvious after a bit of thought: If we had a precise grammar for what did or did not count as valid pseudocode, along with semantic descriptions of the meaning of each of those constructs, then what we would have is a* programming language, *rather than a form of pseudocode. Good pseudocode, which has some flexibility in its presentation, allows us to communicate the idea of an algorithm to* human *readers more clearly than using a true, strictly-defined programming language.*
>
> *The important thing is to remember that pseudocode is a form of writing. The goal is to communicate to steps of the algorithm, to a human reader, in a clear, consistent, and unambiguous way.*

9 The Algorithm Design Process

1.2

- Understand the problem.

  Input: What does an instance look like?
  Output: What is the correct answer?

- Decide on:

  Computational means (sequential, parallel, quantum, DNA, ...)
  Exact vs. approximate
  Data structures
  Design technique

- Design an algorithm.
- Prove correctness.
- Analyze the algorithm.
- Implement the algorithm.

10 Basic data structures: Array (review)

1.4

Advantage: Fast access to any element by index.
Disadvantage: Inserting or deleted (except at the end) is slow.

11 Basic data structures: Linked list (review)

Advantage: Fast inserts and deletes.
Disadvantage: Must be accessed sequentially.

12 Basic data structures: Stack (review)

Inserts (“push”) and deletes (”pop”) only at one end (“the top”).
LIFO

13 Basic data structures: Queue (review)

Inserts (“enqueue”) at one end (“rear”).
Deletes (“dequeue”) from the other end (“front”).
FIFO