COMP 200: Elements of Computer Science
Fall 2004
Lecture 1: August 23, 2004

On the Board

Reading: Chapter 1 in Algorithmics

Homework: none yet

Administrative Details

1. Web site is http://www.owlnet.rice.edu/~comp200; other information flows from the web site. See, in particular, general information page.
2. Professor is Keith Cooper, DH 3131, x6013, keith@rice.edu; give some background information.
3. Laboratory Assistants are Nathan Froyd and Gabriel Marin; office hours and contact information to be posted on the web site
4. Go over the general information document.
5. Lecture style: some will be (slick) PowerPoint presentations. Some will be taught at the marker board. Some will be discussions. Some will (I hope) be guest lectures.

What is Computer Science?

"Computer science" is a strange term. Why is a "science" located in the School of Engineering? (CAAM used to be "Mathematical Sciences".) If "computer science" is an academic discipline, what about television science, toaster science, or motorcycle science? In one sense, "computer science" is a misnomer. Better terms might be "computation" or "digital-systems engineering. However, we're stuck with "computer science", so we should get used to it.

Teachers at elementary and secondary schools who use the term "computer class" drive me nuts. Most of the time, they are talking about a course in typing or in using specific applications, such as Microsoft Office.

Computer Science is hard to define because so many of the main ideas are abstract.

My definition: *Computer Science is the academic discipline that encompasses the design of computers and the software that runs on them.*

This definition is usually followed by a list of subareas in Computer Science. The list is constructed so that no colleague is ignored or neglected. The result is an excessively long list that provides almost no motivation or excitement.

Gibbs & Tucker propose the following definition (somewhat algorithm-centric): (paraphrased)

The study of algorithms including aspects of linguistic implementation, hardware implementation, mathematical properties, and applications.

Again, I find this definition to be unsatisfying. The point of the course is to convey to you the breadth and depth of the field that we call Computer Science, ***and*** to introduce you to the fundamental concepts that make it a field worth studying. The book, and the professor, both take the position that the core of Computer Science lies in the notion of an algorithm.

What is an algorithm?

Ask the class to define algorithm.

Definitions that I saw in classes that I took:

1. Turing machine that always halts
2. An effective procedure for solving a problem

Actually, these two are equivalent, but the second is more comprehensible.

On the board, numbered

- An algorithm must produce an answer — that is what we mean by *effective*. The answer may be imprecise or approximate (a Google search); but the answer must appear.
- An algorithm must halt. The set of steps that it executes must be finite. Running through a series of steps forever without producing an answer may be a good way to burn power or waste money, but it is not an algorithm. ***Lather. Rinse. Repeat.***
- An algorithm must be precisely specified ⇒ finite set of well-defined steps. The steps can be described at a high-level of abstraction (look up the phone number in the white pages), but each step must be concrete and doable in a finite number of steps.
- An algorithm must be complete ⇒ it must handle any situation that can arise during its execution. Windows XP is not an algorithm.
- An algorithm must be executable ⇒ otherwise, we cannot tell if it halts and we cannot use it for effective computation.

Example — using a phone book

Consider the problem that arises when you want to order pizza. (After all the details are worked out…) Eventually, you need the phone number where you can place an order. Since there is only one pizza place in Houston (Star Pizza on Norfolk), you grab the business white pages and lookup the number. How do you lookup a number in the phonebook?

⇒ Start at the first entry and work forward, one at a time. Stop when you find the entry for Star Pizza on Norfolk or when you run out of entries?

⇒ Is that an algorithm? Effective? Halts? Precisely Specified? Complete? Executable? Yes to all five.

⇒ Divide the book in half and examine the middle entry. If the entry is Star Pizza on Norfolk, halt. Otherwise, repeat the process on half of the remaining entries, choosing the half in which Star will fall. (If "Star Pizza" is earlier in lexicographic order, focus on the first half. If "Star Pizza is later in lexicographic order, focus on the second half.) If the remaining part of the book is a single entry, then halt and report failure.

⇒ Is that an algorithm? Effective? Halts? Precisely Specified? Complete? Executable? Yes to all five.

⇒ Search the section for entries beginning with S in the same way. (Why do dictionaries have those inset tabs, anyway?)

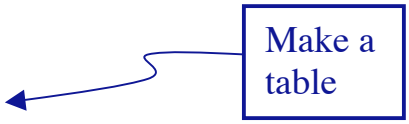
⇒ This algorithm is an optimization of the previous algorithm that considers a smaller problem (a smaller set of entries). We should expect its properties to be the same, except that it probably runs faster.

This method uses the previous method as a subtask — it invokes the functionality of the previous method as if that method were both well understood and easily available. (Using a previously defined algorithm as a step in a new algorithm is, essentially, *functional abstraction*.)

⇒ Check your own memory, where you find the number 713 523 0800.

⇒ Is this an algorithm? Effective? – yes; number is there or it is not. Other questions? Sure.

Each of these methods has different properties.



First idea (linear search) is simple to explain and to execute. We expect it to take, on average, time proportional to the size of the phonebook.

Second idea (binary search) is much harder to describe. (For example, critical details depend on whether the phonebook has an even or an odd number of entries.) It halves the problem size at each step. On average, it should take roughly $\log_2$(number of entries) time.

Third idea (binary search over single letter) has properties of two but reduces problem size ⇒ it should run faster. (Of course, we could use linear search on a single letter and improve its performance, too.) This idea is an algorithm, but it is not distinct from binary search. Instead, it is a trick to decrease the number of steps that binary search requires. We call such a trick an "optimization" (even though it has nothing to do with "optimality".)

Fourth idea is easy to describe—imagine a mathematical function

$\mathcal{F}$: name—> integer

that takes each name into a unique integer. If we associate each phone number with $\mathcal{F}$(name), then the algorithm consists of evaluating $\mathcal{F}$ to obtain the integer and then obtaining its associated phone number.

(Simplest example considers the name as a base 26 number of admittedly huge size. Trick to making this notion efficient is capitalizing on the sparsity of the space of words and, in particular, the sparsity of the set of names that you call on the phone.)