

Algorithms in Systems Engineering IE170

Lecture 2

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
 - CLRS [Chapter 1](#)
- References
 - D.E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (Third Edition), 1997.
 - On-line dictionary www.onelook.com, and a few other on-line dictionaries.

What is an Algorithm?

An Interesting Quote

“Here is your book, the one your thousands of letter have asked us to publish. It has taken us years to do, checking and rechecking countless recipes to bring you only the best, only the interesting, only the perfect. Now we can say, without a shadow of a doubt, that every single one of them, if you follow the directions to the letter, will work for you exactly as it did for us, even if you have never cooked before.”

–McCall's Cookbook (1963)

A Brief History of Algorithms

- According to the Oxford English Dictionary, the word algorithm is a combination of the Middle English word *algorism* with *arithmetic*.
- This word probably did not enter common usage in the English language until sometime last century.
- The word *algorism* derives from the name of an Arabic mathematician circa A.D. 825, whose surname was Al-Khwarizmi.
- Al-Khwarizmi wrote a book on solving equations from whose title the word *algebra* derives.
- It is commonly believed that the first algorithm was **Euclid's Algorithm** for finding the greatest common divisor of two integers, m and n ($m \geq n$).
 - Divide m by n and let r be the remainder.
 - If $r = 0$, then $\gcd(m, n) = n$.
 - Otherwise, $\gcd(m, n) = \gcd(n, r)$.

Show Me the Algorithms

- Algorithms are literally everywhere you look.
- What are some common applications of algorithms?
- Why is it important that algorithms execute quickly?

What is a Problem?

- Roughly, a *problem* specifies what set of outputs is desired for each given set of inputs.
 - A *problem instance* is just a specific set of inputs.
 - **Example:** The Sorting Problem
-
- *Solving* a problem instance consists of specifying a procedure for converting the inputs to an output of the desired form (called a *solution*).
 - An algorithm that is guaranteed to result in a solution for every instance is said to be *correct*.
 - Note that a given instance may have either no solutions or more than one solution.

Simple Example

- Consider the following simple problem:

Input: $x \in \mathbb{R}$ and $n \in \mathbb{Z}$.

Output: x^n .

- What is the simplest algorithm for solving this problem?

- Is there a *better* algorithm?

A More Efficient Algorithm

- Let's assume that $n = 2^m$ for $m \in \mathbb{Z}$.
- Repeated squaring

```
for (y = 1, i = 0; i < m; i++)  
    y *= y;
```

- Is this algorithm correct?
- How much more efficient is it?
- What do we mean by *efficiency*?
- Why don't we call it *speed*?
- How do we modify it for the general case?

Aside: Pseudo-code

- We will use *pseudo-code* to specify algorithms.
- The syntax will be similar to C/C++, but with departures for clarity.
- We will use descriptions in English to specify certain operations.
- The pseudo-code used in lecture may differ from that in the book.
- Check the book for a detailed list of its pseudo-code conventions.

Importance of Algorithms

- We have just seen two different algorithms for solving the same problem.
- The second algorithm was much more *efficient* than the first.
- Would you rather have a faster computer or a better algorithm?

Another Example: Fibonacci Numbers

Thanks to David Eppstein

- Another simple example of the importance of efficient algorithms arises in the calculation of *Fibonacci numbers*.
- Fibonacci numbers arise in population genetics, as well as a host of other applications.
- The n^{th} Fibonacci number is defined recursively as follows:

$$F(1) = 1, \tag{1}$$

$$F(2) = 1, \tag{2}$$

$$F(n) = F(n-1) + F(n-2) \quad \forall n \in \mathbb{N}, n > 2. \tag{3}$$

- How do we calculate $F(n)$ for a given $n \in \mathbb{N}$?

Calculating Fibonacci Numbers

- **Obvious Solution:** Recursive function `Fibonacci()`.

```
fibonacci(int n)
{
    if (n == 1 || n == 2){
        return 1;
    }else{
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

- How efficient is this?
- Is there a more efficient algorithm?

What is a Data Structure?

- Computers operate on tables of numbers (the *data*).
- Within the context of solving a given problem, this data has *structure*.
- *Data structures* are schemes for *storing and manipulating data* that allow us to more easily see the structure of the data.
- Data structures allow us to perform certain operations on the data more easily.
- The data structure that is most appropriate depends on how the algorithm needs to manipulate the data.

Example

- Consider the two implementations of the list class from last lecture.
- An *array* is a simple data structure that allows us to store a sequence of numbers.
- A *linked list* does the same thing.
- What is the difference?

Comparing List Data Structures

- To compare the two data structures, we must analyze the running time of each operation.
- This table compares the running times of the operations.

	Array	Linked List
getNumItems		
getValue		
setValue		
addItem		
delItem		

Importance of Data Structures

- Specifying an algorithm completely includes specifying the data structures to be used (sometimes this is the hardest part).
- It is possible for the same basic algorithm to have several different implementations with different data structures.
- Which data structure is best depends on what operations have to be performed on the data.

Summing Up

- **Algorithms** that are both efficient and correct are a technology that must be developed.
- **Data structures** allow us to represent relationships between various data and allow us to manipulate them effectively during an algorithm.
- Efficient algorithms enable us to solve important problems more quickly, which is critical for many applications.
- This is the focus of this class.