Algorithms in Systems Engineering
ISE 172

Lecture 1

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - 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, $\gcd(m, n) = \gcd(n, r)$.

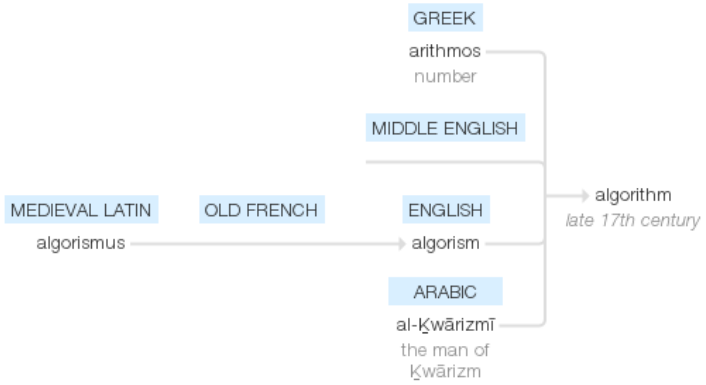
Google Etymology for Algorithm

al·go·rithm
/ˈalgəˌriT̲Həm/

noun
noun: **algorithm**; plural noun: **algorithms**

a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
"a basic **algorithm** for division"

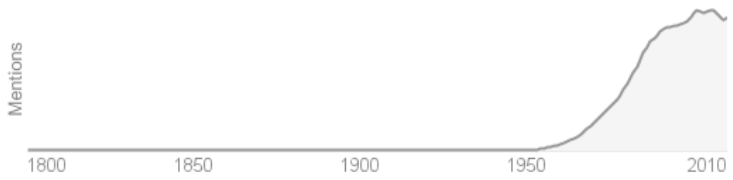
Origin



late 17th century: variant (influenced by Greek *arithmos* 'number') of Middle English *algorism*, via Old French from medieval Latin *algorismus* . The Arabic source, *al-Ḵwārizmī* 'the man of Ḵwārizm' (now Khiva), was a name given to the 9th-century mathematician Abū Ja'far Muhammad ibn Mūsa.

Translate algorithm to Choose language

Use over time for: algorithm



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

```
def pow(x, n):
    for i in range(log(n)):
        x *= x
```

- 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?

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:

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

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

Calculating Fibonacci Numbers

- Obvious Solution: Recursive implementation.

```
def fibonacci1(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fibonacci1(n-1) + fibonacci1(n-2)
```

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

Calculating Fibonacci Numbers: Second Implementation

- Second Try: Store and reuse intermediate results.

```
def fibonacci2(n):
    f = [0, 1, 1]
    for i in range(3, n+1):
        f.append(f[i-1] + f[i-2])
    return f[n]
```

- Are there any downsides of this implementation?

Calculating Fibonacci Numbers: Third Implementation

- Third Try: Only store the intermediate results that are needed.

```
def fibonacci3(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a+b
    return a
```

- Can we do any better?

Calculating Fibonacci Numbers: Fourth Implementation

- Fourth Try: Fast doubling

```
def fibonacci5(n):
    return _fibonacci5(n)[0]

# Returns the tuple (F(n), F(n+1)).
def _fibonacci5(n):
    if n == 0:
        return (0, 1)
    else:
        a, b = _fibonacci5(n // 2)
        c = a * (b * 2 - a)
        d = a * a + b * b
        if n % 2 == 0:
            return (c, d)
        else:
            return (d, c + d)
```

Calculating Fibonacci Numbers: Fifth Implementation

- Fifth Try: Calculate using direct formula.

```
def fibonacci(n):
    x = (1 + sqrt(5))/2
    return x**n - (1-x)**n)/(x - (1-x))
```

- This produces the answer in one step.
- What is a possible problem with this?

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.