Limits of Computation (theory and practice)

- **Computability**
  - problems for which no program exists
  - what is a program? What is an algorithm?
  - does language matter?

- **Complexity**
  - efficient algorithms
  - how hard is a problem?

- How to prove no program/algorithm exists for a problem
- How to prove no efficient algorithm exists for a problem
- How to show “everyone” thinks no efficient algorithm exists
Non-computable problems

● **Solving a problem in general**
  ➤ Given a program P, and an input S, determine *if program P halts on input S* this is the halting problem
  ➤ we must write one program that works for any P and any S, not for a specific P
  ➤ proof by contradiction: assume we can write \textit{halt}, show this leads to a contradiction, therefore we cannot write halt

● **Proofs by contradiction are interesting**
  ➤ Theorem: consider whole numbers: 0, 1, 2, …; all whole numbers are interesting
  ➤ Proof: suppose not …..
Halting problem

bool halt(const string & p,
           const string & s);
// pre: p is a valid program/function
// post: returns true if p halts on s
//       returns false otherwise

void confuse()
{
    string s;
    cin >> s;
    if (halt(s,s))
        while (true)
        {}
    cout << “hello world” << endl;
}

What happens if we type the
text of confuse?[check halt pre/post]
Does confuse print “hello world”?
More on computability/complexity

- Program/algorithm cannot do the following:
  - determine if a program has an infinite loop
  - determine if two programs do the same thing
  - note: one program might work in a restricted domain, no program works for solving all problems

- Can’t do efficiently
  - Towers of hanoi
  - factor numbers (why do we care about this?)

- Can’t do efficiently so far [and most don’t think we can]
  - can we tour cities for less than $$$
  - determine of N trucks can be packed with given boxes
NP complete problems

● NP: non-polynomial
  ➤ O(n^2), O(n^3), even O(n^{20}) are polynomial, O(2^n) is not
  ➤ deals with decision problems: yes/no answer to the problem (but we can find a numeric solution in some cases with binary search)
  ➤ verifying that a particular solution-attempt works/doesn’t work can be done efficiently
  ➤ must try lots of potential solutions to find overall yes/no answer

● This branch of computer science was invented in the 70’s, due in large part to Stephen Cook [brother teaches at Duke!]
What can we do with impossible or hard problems?

- **Impossible in theory, solvable in practice**
  - what’s the value of n we’re interested in?
  - heuristic approaches can work

- **Don’t need absolutely the best answer, can deal with “close enough”**
  - with approximate solutions some intractable problems become tractable

- **Wait for machines to become faster**