MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.081—Introduction to EECS I
Fall Semester 2006

Lecture Notes – September 12

Elements of Programming

Some simple Python procedures

```
def square(x):
    return x*x

def average(a,b):
    return (a + b) / 2.0

def meanSquare(a,b):
    return average(square(a), square(b))
```

A procedure for computing square roots:

```
def goodEnough(guess, x):
    return abs(x-square(guess)) < .00001

def improve(guess,x):
    return average(guess, x/guess)

def sqrtIter(guess,x):
    while not(goodEnough(guess,x)):
        guess=improve(guess,x)
    return guess

def sqrt(x):
    return sqrtIter(1.0,x)
```

Another version of the square root procedure, that uses block structure

```
def sqrt(x):
    def goodEnough(guess):
        return abs(x-square(guess)) < .00001
    def improve(guess):
        return average(guess, x/guess)
    def iter(guess):
        while not(goodEnough(guess)):
            guess=improve(guess)
        return guess
    return iter(1.0)
```

Computing powers, b^e

```
def expt(b,e):
    if e==0:
        return 1
    else:
        return b*expt(b,e-1)
```

This results in a **linear time process**

Fast exponentiation:

```
def fastexp(b,e):
    if e == 0:
        return 1
    elif e % 2 == 1:
        return b * fastexp(b,e-1)
    else:
        return square(fastexp(b,e/2))
```

This results in a **logarithmic time process**

Orders of growth:

For a process that uses resources $R(n)$ for a problem of size n, we say that $R(n)$ has *order of growth* $\Theta(f(n))$ if there are positive constants k_1 and k_2 independent of n such that

$$k_1 f(n) \leq R(n) \leq k_2 f(n)$$

Computing powers modulo m, $b^e \pmod{m}$

```
def expmod(b,e,m):
    if e == 0:
        return 1
    elif e % 2 == 1:
        return ( b * expmod(b,e-1,m)) % m
    else:
        return square(expmod(b,e/2)) % m
```

For the problem: Given a and p and x, find y such that $a^x = y \pmod{p}$ can be solved in time logarithmic in x.

But there is no known shortcut for the inverse *discrete logarithm problem:* Given a and p and y, find x such that $a^x = y \pmod{p}$. Solving this problem takes *exponentially more time* than computing powers modulo p. (If p is a prime number.)

This fact is the basis of Diffie-Hellman key agreement, which makes it possible to have secret communication on public channels.