Powers

Textbook Reading:

- Section 1.1, pp. 9-14
- Section 3.7.1, pp. 122-123.

Compute x^p for a given positive integer exponent p.

- This problem has important applications in cryptography and security, and is usual used with large integers that can not be stored in a standard type such as int.
- Data structures for implementing large integers include strings, lists, linked lists and arrays.

Naïve Algorithm for powers

```
function NaivePowers(x,p) recursive
  Input: x (a real number), p (a positive integer)
  Output: $x^p$
  power ← x
  for counter ← 1 to p – 1 do
      power = power*x
  endfor
  return(power)
end NaivePowers
```

A key goal is not just to find an algorithm that solves the problem but to find the fastest algorithm

Input Size

- If we take the input size to be $n = p$ then *NaivePowers* performs $n - 1$ multiplications, which appears to be fairly efficient.
- However, this is deceptive. In applications such as computing a cryptographic key in cryptology, which we will see later in this lecture, we need to compute powers of x for large p, i.e., consisting of hundreds or even thousands of digits.
- For these applications, it is more meaningful to take the input size n to be the number of (binary) digits of p, in which case p will be approximately 2^n, i.e., we perform about 2^n multiplications.
- This is EXTREMELY inefficient for large n and would take zillions and zillions of years for thousand digit numbers.

USE SIMPLE KEY FACT:

$(x*x)^{p/2} = x^p$ **if** p **is even,**

$x*((x*x)^{(p-1)/2}) = x^p$ **if** p **is odd.**

```
function Powers(x,p) recursive
   Input: x (a real number), p (a positive integer)
   Output: $x^p$
     if $p = 1$ then return (x)
     if even(p) then
         return(Powers(x*x,p/2))
     else
         return(x*Powers(x*x,(p – 1)/2))
     endif
end Powers
```

Analysis

Algorithm *Powers* does at most about $2\log_2 p$ multiplications or about $2n$ multiplications where n is the number of digits of p in binary.

Problem Solving Notebook: Number of digits

Show that the number n of decimal digits of p is approximately $\log_{10}p$.

What about number of binary digits? Octal digits? hexadecimal digits?

What about using simpler formula: $(x*x)^{p/2} = x^p$ if p is even, $x*x^{p-1} = x^p$ if p is odd **Is it still efficient?**

```
function Powers(x,p) recursive
   Input: x (a real number), p (a positive integer)
   Output: $x^p$
  if $p = 1$ then return (x)
  if even(p) then
      return(Powers(x*x,p/2))
  else
      return(x*Powers(x,p – 1)
  endif
end Powers
```

- Yes, it is still efficient. Every other recursive call p is even and the next call cuts p in half.
- Therefore, the number of multiplications performed is about 2 $\log_2 p$ or about 2n, where n is the number of binary digits of p.
- This is the same performance as before, except that the stack of recursive calls will be larger.

Nonrecursive version for computing powers Left-to-Right Binary Method

```
1. Compute the binary representation of $p$
2. Initialize Pow to $x$
3. Scan from left-to-right starting with second position
    3.1 if 0 is encountered square Pow
        else  // if 1 is encountered
            square Pow and multiply by $x$
        endif
4. return Pow
```

Example of action of left-to-right Binary Method for powers (exponentiation)

Compute x^{114}

114 = 64 + 32 + 16 + 2

Binary representation is:

1110010

$x \rightarrow x^2 * x = x^3 \rightarrow (x^3)^2 * x = x^7 \rightarrow (x^7)^2 = x^{14} \rightarrow (x^{14})^2 = x^{28} \rightarrow (x^{28})^2 * x = x^{57} \rightarrow (x^{57})^2 = x^{114}$

Problem-Solving Notebook Changing to Binary

Give pseudocode for a recursive function *BinRep*(*n*) for converting an number *n* to its binary (base 2) representation stored in a string. Assume + performs the operation of adding a digit, i.e.,

“10001101” + 0 → “100011010”

One-way functions and security schemes on Internet

Most of the security schemes currently implemented on the internet, and on which the integrity of electronic commerce is based, depend on the notion of so-called "one-way" functions. A one-way function *f*(*n*) is one in which there is an efficient way to compute *f*(*n*) for even a very large n, but, there is no efficient way (or, at least no KNOWN efficient way) to compute its inverse. In other words, it is computationally infeasible to determine what n is given the value *f*(*n*).

Example of a One-Way Function Modular Exponentiation x^p mod k

- It has been estimated that there are less than 10^{83} atoms in the known universe. While this number is large indeed, it still has less than 280 binary digits in its base two representation, and therefore is easily stored in a computer using an array of size 280.

- Using an Powers or the Binary Method $x^{(10^{83})}$ can be computed using about 2×280 = 560 multiplications.
- The catch is that the output is too large and can't be stored in any computer. For example for x = 10 the output would be $10^{(10^{83})}$, which would require more than the number of atoms in the known universe to store, so it certainly cannot be stored in any computer.

What to do?

To get around this problem in practice, i.e., in cryptography applications, we carry out all of our calculations **modulo some integer *k***, so that the numbers we calculate in the repeated squaring process never exceed $(k - 1)^2$. This latter fact follows from the relation

$$xy \bmod k = (x \bmod k)(y \bmod k) \bmod k$$

so, in particular,

$$x^2 \bmod k = (x \bmod k)(x \bmod k) \bmod k.$$

Modular Exponentiation

```
function $ModularExponentiation(x,p,k)$ recursive
  Input: $x, p, k$ (positive integers)
  Output: $x^p$ $(mod\ k)$
    if $p = 1$ then return $(x \bmod k)$
    if $(p \bmod 2 = 0)$ then
        return $(ModularExponentiation(x*x \bmod k, p/2))$  mod $k$ )
    else
        return ( $x*ModularExponentiation(x*x \bmod k, (p-1)/2))$ mod $k$ )
   endif
end Powers
```

Converse is HARD problem

- Given x, p and x^n mod p, there is no known efficient algorithm to compute n.
- This fact can be used to share secrets in a secure way.

Seminal Example from Cryptography cont'd

- Alice and Bob wish to communicate confidential information over the ether using insecure medium (Internet, cellphone, email). We assume that Eve is listening in on everything that is sent between Alice and Bob, and that Alice and Bob have not previously agreed on a secret key to use in some standard encryption scheme.
- The question is, can Alice and Bob exchange any piece of information, i.e., a cryptographic key, that Eve can not discern even though she intercepts every message sent between Alice and Bob?
- The answer is YES (and electronic commerce depends on this!). Here's the idea.

Seminal Example from Cryptography cont'd

- First Alice and Bob communicate with each other to decide on the value of a integer base *b* and a large integer *p*.
- Now Alice, Bob, and Eve all know *b* and *p*.
- Then Alice chooses a large integer *n*, (which ONLY she knows), and sends Bob the value b^n mod *p*.
- Then Bob chooses a large integer *m* (which ONLY he knows), and sends Alice b^m mod *p*.
- Now they ALL know values *b*, *p*, b^n mod *p*, and b^m mod *p*.

Who knows what

Alice	Bob	Eve
b	b	b
p	p	p
n	m	
$b^n \bmod p$	$b^n \bmod p$	$b^n \bmod p$
$b^m \bmod p$	$b^m \bmod p$	$b^m \bmod p$

Computing cryptographic key *K*

PSN. How about computing key as follows

$$K = (b^m \bmod p)(b^n \bmod p) = b^{m+n} \bmod p$$

Trick that works

- Alice computes $(b^m \bmod p)^n \bmod p$, and Bob computes $(b^n \bmod p)^m \bmod p$. Note that they have computed the SAME number $K = b^{mn} \bmod p$.
- Now if Eve could compute the value of n from b, p and $b^n \bmod p$, then she could also compute this common value by computing the value $(b^m \bmod p)^n \bmod p$. However, no one knows how to do this efficiently for large values of *n.*
- This problem is known as the **Discrete Logarithm Problem.**
- Some of the greatest minds in mathematics and computer science have tried to crack this problem for years now, without success.

Bagel Challenge

If you solve the famous Discrete Logarithm problem, you will get an A in the course.

Further, I will buy you a baker’s dozen bagels

and write you are check for

$$\$\,e^{i\pi}+1$$

There are 10 kinds of people in the world.

Those who know binary and those who don't.

10 people