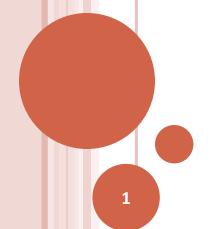
# CS F364 Design & Analysis of Algorithms

# PROBLEM DOMAIN - NUMBER THEORY

#### **Testing for Primes:**

- Motivation
- Cost of Deterministic algorithm(s)
- Expected Cost of Finding a Prime Number



## CHOICE OF MODULUS IN RSA

- RSA performs operations "modulo n" for some n = p \* q, where
   p and q are large primes
  - and its security is contingent on the condition that
     ofactoring cannot be done in time polynomial in length of n
     i.e. time polynomial in log(n)
- For the security condition to be meaningful in practice,
  - n should be large enough
    i.e. p and q should be large enough.
  - Today 1024-bit modulus is not strong enough:
    - o i.e. a 1024-bit **n** can be factored in "reasonable" time

### LENGTH OF MODULUS IN RSA

- For factoring to be computationally expensive:
  - current recommendations for n are
    - o at least 1392 bits
      - estimated using a theoretical calculation by Lenstra
    - oat least 2048 bits
      - recommended by NIST (circa 2013)
    - oat least 3072 bits
      - orecommended by NSA (circa 2015)
- Thus we need two prime numbers

in the range of ~700 bits to ~1500 bits

for each pair of communicators!

### GENERATION OF PRIMES FOR RSA

- For large scale use of RSA, a <u>large number of large pairs of</u> <u>primes</u> have to be found:
  - There are <u>no algebraic</u> (or other) techniques to generate all <u>primes!</u>
    - oi.e. primes have to be found by
      - sampling numbers and testing them for primality.

#### **PRIMALITY TESTING - COST**

```
• (Naive) Primality Testing – Algorithm 1:
testPrime(N) { // large N
for all odd numbers m from 3 to L√N | {
if (m | N) return false;
}
return true;
}
```

- What is the time complexity of this algorithm,
  - assuming uniform cost model?
  - assuming logarithmic cost model?
- o Is it a polynomial time algorithm?

# PRIMALITY TESTING - COST - (NAIVE) SIEVING

```
(Naive) Primality Testing – Algorithm 2
sieveEratosthenes(N):
                                                                      Exercise
     create a list L[1..N] of Boolean
1.
                                               What is the time complexity?
      for j = 1 to N { L[j] = true; }
                                          Does it depend on the cost model?
                                     b)
                                              What is the space complexity?
      L[1] = false; nxtPrm = 2;
                                              What is the amortized cost per
     while (nxtPrm < sqrt(N)) {
                                                              prime number?
         mult = nxtPrm;
         while (mult < N) { mult += nxtPrm; L[mult] = false; }
        while (!L[nxtPrm]) nxtPrim += 1;
    3.
5.
     create a set Primes and initialize it to the empty set;
6.
     for j = 2 to N { if (L[j]) { add L[j] to Primes; } }
7.
     return Primes;
8.
```

### **PRIMALITY TESTING**

- Primality Testing:
  - Proven to be a problem that can be computed in polynomial time by:
    - Agarwal, Kayal, and Saxena. PRIMES is in P, 2002.
  - AKS is the best known deterministic algorithm without assumptions – for testing whether an integer n is prime :
    - Running time:  $O((logn)^k)$  for  $k \approx 7.5$

## PRIMALITY TESTING: PRAGMATICS

- Consider typical high security recommendation for RSA (circa 2015): 1024 bit keys
  - i.e. 1024 bit prime numbers have to be generated and  $1024^{7.5} = 2^{75} > 10^{22}$  operations (for AKS)
    - oThis will require a  $10^{22}$  /  $10^{12}$  =  $10^{10}$  seconds on a system that can deliver 1 Tera operations per second
      - oi.e.  $10^{10}$  / (3600\*24\*365) = about 300+ years!
- To put this in perspective:
  - Circa 2015, typical multi-core systems with Intel i7 octacore can deliver:
    - ~350 Giga operations per second

## FINDING LARGE PRIMES

- O How do you find large primes?
  - This would be the outline:

```
for (next = oddLow; next < oddHigh; next += 2) {
  if prime(next) return next;
```

- What is the expected running time of this algorithm?
  - What should be oddLow and oddHigh?

## **DISTRIBUTION OF PRIMES**

- Basic guarantees:
  - Elementary Prime Distribution Theorem:
    - There exists a prime number between n and 2\*n for any n>1
  - Exercise: Prove this.
- Definition  $\pi(x)$ 
  - Let x be a positive real number > 1. Then π(x) is defined as the number of primes less than or equal to x.
- Prime Number Theorem:
  - $\pi(x)$  is asymptotic to  $x / \ln(x)$ • i.e.  $\lim_{x \to \infty} \pi(x) / (x / \ln(x)) = 1$

## **COST OF FINDING PRIMES**

• Then given this outline:

```
for (next = lo; next < hi; next +=2 )
    { if prime(next) return next;}</pre>
```

the expected running time would be

P(log<sub>2</sub>(hi)) \* T(hi,lo) where

- P(M) is the time taken for testing a number sized M and
- **T(hi, lo)** is the expected number of trials to find a prime in the interval (**lo,hi**).
  - oT(x,y) = 1/D(x,y) where D(x,y) is the probability of finding a prime between x and y.
  - oi.e.  $T(x,y) = |x-y| / |\pi(x) \pi(y)|$ = |x-y| / |(x/ln(x)) - (y/ln(y))|