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
    - *factoring 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:
    - 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
    - at least **1392 bits**
      - estimated using a theoretical calculation by Lenstra
    - at least **2048 bits**
      - recommended by NIST (circa 2013)
    - at least **3072 bits**
      - recommended 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 *large number of large pairs* of primes have to be found:
  - There are *no algebraic (or other) techniques to generate all primes!*
    - i.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 ⌊√N⌋ {
      if (m | N) return false;
    }
    return true;
 }
```

- What is the time complexity of this algorithm,
  - assuming ***uniform cost model***?
  - assuming ***logarithmic cost model***?

- Is it a polynomial time algorithm?

Primality Testing – Cost – (Naive) Sieving

- (Naive) Primality Testing – Algorithm 2

sieveEratosthenes(N):

1. create a list L[1..N] of Boolean
2. for j = 1 to N { L[j] = true; }
3. L[1] = false; nxtPrm = 2;
4. while (nxtPrm < sqrt(N)) {
    1. mult = nxtPrm;
    2. while (mult < N) { mult += nxtPrm; L[mult] = false; }
    3. while (!L[nxtPrm]) nxtPrim += 1 ;
5. }
6. create a set Primes and initialize it to the empty set;
7. for j = 2 to N { if (L[j]) { add L[j] to Primes; } }
8. return Primes;

Exercise

a) What is the time complexity?
b) Does it depend on the cost model?
c) What is the space complexity?
d) What is the amortized cost per prime number?

Primality Testing

- Primality Testing:
  - Proven to be a problem that can be computed in polynomial time by:
    - Agarwal, Kayal, and Saxena. *PRIMES is in $\mathcal{P}$*, 2002.
  - AKS is the best known deterministic algorithm – *without assumptions* – for testing whether an integer n is prime :
    - Running time: $O((\log n)^k)$ 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 $\mathbf{1024^{7.5} = 2^{75} > 10^{22}}$ operations (for AKS)
    - This will require a $\mathbf{10^{22} / 10^{12} = 10^{10}}$ seconds on a system that can deliver **1 Tera operations per second**
      - i.e. $\mathbf{10^{10} / (3600*24*365)}$ **= about 300+ years**!

- To put this in perspective:
  - Circa 2015, typical multi-core systems with Intel i7 octa-core can deliver:
    - ~350 Giga operations per second

Finding Large primes

- 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 $\pi(x)$ 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;}

- the expected running time would be

  $P(\log_2(hi))$ * 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).**
    - **T(x,y) = 1/D(x,y)** where **D(x,y)** is the probability of finding a prime between x and y.
    - i.e. **$T(x,y) = |x-y| / |\pi(x) - \pi(y)|$**

      $= |x-y| / |(x/\ln(x)) - (y/\ln(y))|$