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

Primality Testing

- RSA uses operations "modulo n" for n = p * q where p and q are large primes.
  - For large scale use of n, 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):
    - for all odd numbers m from 3 to $\lfloor\sqrt{N}\rfloor$
      - { 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 - EXERCISE

- (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 for testing whether an integer n is prime :
    - Running time: $O((\log n)^k)$ for $k \approx 7$

Primality Testing : Pragmatics

- Consider RSA – typical high security recommendation (circa 2010): 1024 bit keys
  - i.e. 1024 bit prime numbers have to be generated and $1024^7 = 2^{70} = 10^{21}$ operations (for AKS)
    - This will require a $10^{21} / 10^{12} = 10^9$ seconds on a system that can deliver 1 Tera operations per second
      - i.e. 10^9 / (3600*24*365) = about 30+ years!
- To put this in perspective:
  - In early 2014, typical multi-core systems with Intel i7 quad-core can deliver:
    - 50 to 60 Giga operations per second
  - And most security recommendations have switched from 1024 bit keys to 2048 bits key (or even 4096)

RSA – Keys

- Security of RSA depends on several aspects, but a key aspect is that of:
  - Finding large primes p and q
- 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?

Finding 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. $\lim_{x \to \infty} \pi(x) / (x / \ln(x)) = 1$

Finding Primes

- Then given this outline:

  for (next = oddLow; next < oddHigh; next +=2)

  { if prime(next) return next;}

- the expected running time would be

  $P(\log_2(\text{oddHigh}))*T(\text{oddHigh, oddLow})$ where

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

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