Recognition of Prime Numbers in Polynomial Time

Richard Borie
University of Alabama
July 4, 2005

The problem

- Input: Positive integer n
- Size of input: \( b = \lg n \) (# bits)
- Question: Is \( n \) prime or composite?
- Note: Pseudo-polynomial-time algorithms are easy to find
  - Polynomial in \( n \)
  - Exponential in \( b \)

Some brief history

- Listed as one of 12 important problems having unknown complexity
- Complexity status is resolved
  - "PRIMES is in P", Agrawal/Kayal/Saxena, 2002
  - Algorithm runs in \( O^{*}(b^{12}) = O(b^{12+o(1)}) \) time

High-level view of algorithm (part 1)

- Find the smallest prime \( r \) such that
  - \( n \) is not divisible by any of 2, 3, ..., \( r \)
  - \( r-1 \) has a prime factor \( q \geq 2 \sqrt{r \lg n + 2} \)
    - \( r^{p-1}\) \(\not= 1 \mod r \)
- If no such \( r \) exists then \( n \) is composite
- Otherwise proceed to next step

High-level view of algorithm (part 2)

- For each \( a \) in \( 1 \leq a \leq 2 \sqrt{r \lg n + 1} \)
  - Verify that \( (x+a)^p = (x^p+a) \mod n \) \mod \( x^{r-1} \)
  - Note: these are polynomials over variable \( x \)
- If any condition fails then \( n \) is composite
- Otherwise proceed to next step

High-level view of algorithm (part 3)

- For each \( p \) in \( 2 \leq p \leq \lg n \)
  - Compute \( m = \text{floor}(n^{1/p}) \)
  - Verify that \( m^p \neq n \)
- If any condition fails then \( n \) is composite
- Otherwise \( n \) is prime
What remains?

- Proof of correctness
  - Algorithm always terminates
  - Algorithm always answers correctly
    - composite or prime
- Analysis
  - How to implement efficiently
  - Show running time is $O(b^{12}) = O(b^{12+o(1)})$
    - Actually $O(b^{12} \log^k b)$ time for some constant $k$

Recall: High-level view of algorithm (part 1)

- Find the smallest prime $r$ such that
  - $n$ is not divisible by any of 2, 3, ..., $r$
  - $r-1$ has a prime factor $q \geq 2 \sqrt{r \log n} + 2$
  - $n^{(r-1)/q} \neq 1 \pmod{r}$
- If no such $r$ exists then $n$ is composite
- Otherwise proceed to next step

Medium-level view of algorithm (part 1a)

- Try successive values of $r = 2, 3, 4, 5, ...$
- Sufficient to consider only $O(b^6)$ values of $r$
  - Justification uses techniques from abstract algebra and number theory
- By the time $O(b^6)$ values of $r$ are considered, one of the following occurs:
  - We find a value of $r$ that satisfies the stated conditions, or
  - We find a value of $r$ that is a divisor of $n$, in which case we know that $n$ is composite

Medium-level view of algorithm (part 1b)

- For each value of $r$ that is considered
  - Compute $\gcd(n, r)$ by Euclid's algorithm in $O(1)$ time
  - Factor $r$ and $r-1$ by brute force in $O^{(\sqrt{r})} = O^{(b^3)}$ time
  - Let $q$ denote the largest factor of $r-1$
  - Compute $n^{(r-1)/q} \pmod{r}$ by repeated squaring in $O(1)$ time
- Total time over all $O(b^6)$ values of $r$ is $O(b^6) \times O^{(b^3)} = O^{(b^9)}$

Recall: High-level view of algorithm (part 2)

- For each $a$ in $1 \leq a \leq 2 \sqrt{r \log n} + 1$
  - Verify that $(x+a)^n = (x^n+a) \pmod{n} \pmod{x^{r-1}}$
  - Note: these are polynomials over variable $x$
- If any condition fails then $n$ is composite
- Otherwise proceed to next step

Example of algorithm computation (part 2)

- Example
  - Suppose $n=65$, $r=7$, $a=2$
  - $(x+2)^{65} = (2x^6+2x^5+53x^4+49x^3+14x^2+52x+6) \pmod{65} \pmod{x^{7-1}}$
  - $(x^{55}+2) = (x^2+2) \pmod{65} \pmod{x^{7-1}}$
  - $(2x^4+2x^5+53x^4+49x^3+14x^2+52x+6) \neq (x^2+2) \pmod{65} \pmod{x^{7-1}}$
  - So 65 is composite
Medium-level view of algorithm (part 2a)

• Consider only $O(\sqrt[4]{\log n}) = O(b^3 \cdot b) = O(b^4)$ values of $a$
  
  - If any condition fails then $n$ is composite
    
    • Justification uses techniques from abstract algebra and number theory
  
  - If all conditions succeed then $n$ is either prime or some power of a prime
    
    • Again, justification uses techniques from abstract algebra and number theory

Medium-level view of algorithm (part 2b)

• For each value of $a$ that is considered
  
  - Compute $(x+a)^n \pmod{n} \pmod{x^r-1}$ by repeated squaring
  
  - There are $O(\log n) = O(b)$ squarings, and each squaring can be performed via FFT
  
  - Each squaring multiplies polynomials with degree $< r$; hence FFT requires $O(r \log r) = O(b^3)$ scalar products
  
  - Each scalar product involves coefficient values $\cdot n$, so it can be done using FFT in $O((\log n) = O(b)$ time

• Total time over all $O(b^4)$ values of $a$ is $O(b^4) \cdot O(b) \cdot O^*(b^6) \cdot O^*(b) = O^*(b^{12})$

Recall: High-level view of algorithm (part 3)

• For each $p$ in $2 \leq p \leq \log n$
  
  - Compute $m = \text{floor}(n^{1/p})$
  
  - Verify that $m^p \neq n$

• If any condition fails then $n$ is composite

• Otherwise $n$ is prime

Medium-level view of algorithm (part 3a)

• Consider only $O(\log n) = O(b)$ values of $p$
  
  - Compute $m = \text{floor}(n^{1/p})$ using numerical analysis techniques
    
    • Variation of Newton-Raphson algorithm for finding the solution to equation $x^p - n = 0$
    
    - Uses derivatives (slopes) to approximate a solution
    
    • Takes $O(\log n) = O(b)$ iterations to converge
    
    • Each iteration takes $O^*(b^2)$ time for arithmetic operations
    
    - Also compute $m^p$ by repeated squaring
      
      • $O(\log p) = O^*(1)$ iterations, each taking $O^*(b)$ time

Medium-level view of algorithm (part 3b)

• Total time over all $O(b)$ values of $p$ is $O(b) \cdot O^*(b^2) = O^*(b^3)$

• If $n = m^p$ for some $m \geq 2$, $p \geq 2$ then algorithm correctly reports that $n$ is composite

• Otherwise algorithm correctly reports that $n$ is prime

• Finally, total time for entire algorithm is $O^*(b^{12})$, due to dominant cost in part 2

More recent history

• Improved proof of correctness and analysis for Agrawal et al.’s algorithm
  
  
  - Algorithm runs in $O^*(b^{10.5}) = O(b^{10.5+o(1)})$ time

• Faster algorithm
  
  - “Primality Testing with Gaussian Periods”, Lenstra/Pomerance, 2003
  
  - Algorithm runs in $O^*(b^6) = O(b^6+o(1))$ time
Opportunity for volunteers

• Select one (or more) of the loose ends in medium-level description of algorithm
  - Euclid’s algorithm
  - Repeated squaring
  - FFT for multiplying scalars
  - Integer variation of Newton-Raphson method
  - Correctness of compositeness in part 1
  - Correctness of compositeness in part 2
  - Correctness of primality at the end of part 3

• Provide further explanation
  - Missing pieces of algorithm, proof of correctness, and/or analysis details