

AKS

Implementation of a Deterministic Primality Algorithm

R. Salembier & P. Southerington

AKS Primality Test

- Deterministic primality-proving algorithm
 - Manindra Agrawal, Neeraj Kaval, Nitin Saxena
 - "PRIMES is in P," 6 August 2002
- Determines whether a number is prime or composite.
 - There is no probability associated as with Miller-Rabin Test

Original AKS Algorithm

August 6, 2002 version:

- 1. If $(n = a^b \text{ for } a \in \mathbb{N} \text{ and } b > 1)$, output COMPOSITE.
- 2. Find the smallest r such that $O_r(n) > 4 \log^2 n$
- 3. If $1 < \text{GCD}(a, n) < n$ for all $a \leq r$, output COMPOSITE.
- 4. If $n \leq r$, output PRIME.
- 5. For $a = 1$ to $\lfloor 2 \sqrt{\varphi(r)} \log n \rfloor$
 - if $((x + a)^n \neq x^n + a \pmod{x^r - 1, n})$
 - output COMPOSITE.
- 6. Output PRIME.

Lenstra/Pomerance Variant

Revised version we are using:

- 1. If $(n = a^b$ for $a \in \mathbb{N}$ and $b > 1$), output COMPOSITE.
- 2. Find the smallest r such that $O_r(n) > \log^2 n$
- 3. If $\gcd(a, n) \neq 1$ for all $a \leq r$, output COMPOSITE.
- 4. For $a = 1$ to $\lfloor \sqrt{r} \log n \rfloor$
- if $((x + a)^n \equiv x^n + a \pmod{x^{r-1}, n})$, output PRIME.

Complexity

- Big-O Notation
 - Read as 'order of'
 - Indicates the dominant term
- AKS runs in polynomial time
 - Original: $\tilde{O}(\log^{12}(n))$
 - Lenstra/Pomerance: $\tilde{O}(\log^{7.5}(n))$

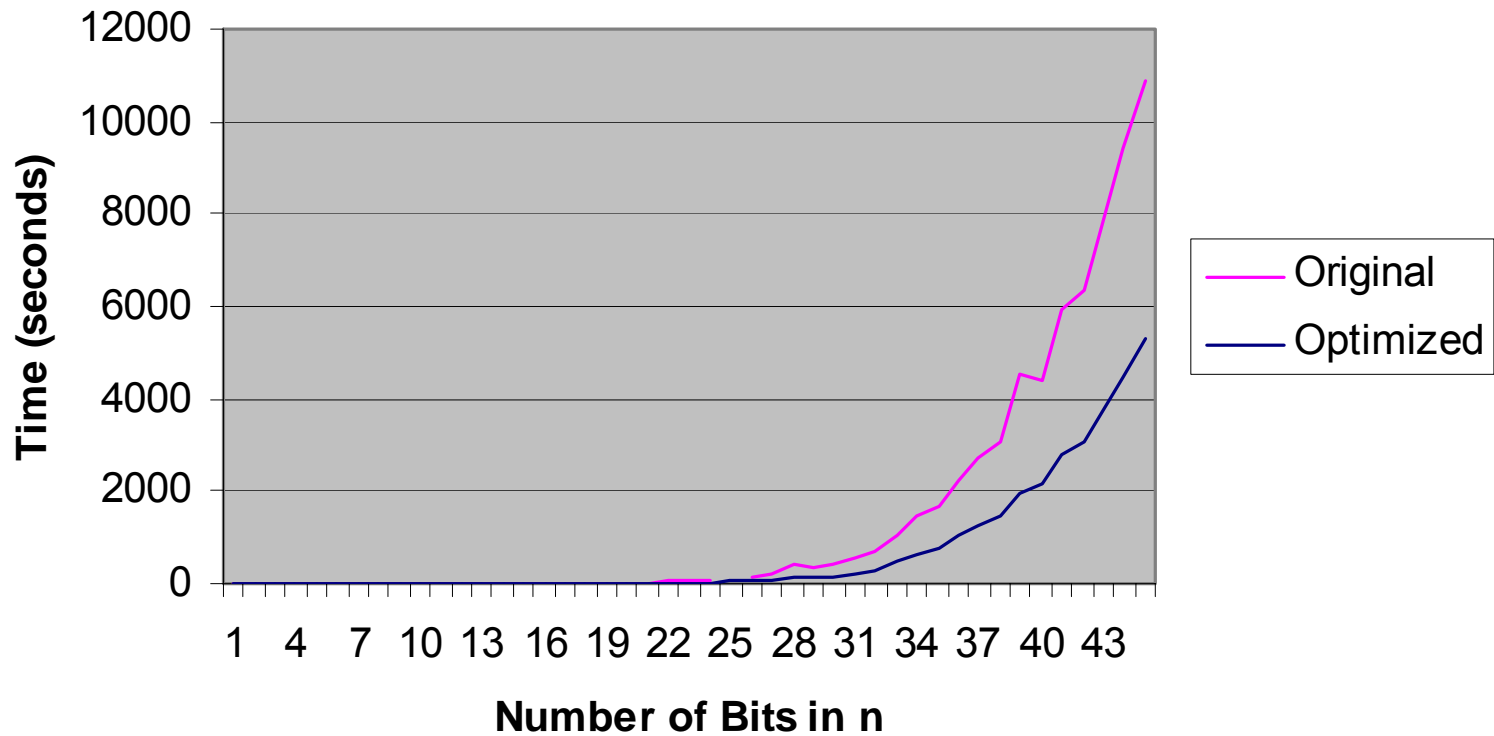
LiDIA Implementation

- Perfect Power Test
 - used native `is_power()` function
- GCD Test
 - Initially used `gcd()`
 - Traditional method of GCD calculation
 - Changed to `bgcd()`
 - Eliminates division
 - Uses shifts and adds
- Exponentiation
 - Left-to-Right Expansion
 - Modular reduction at each step

Division Optimization

- Split polynomial into three coefficient vectors
 - Terms with exponents $> r-1$
 - Term with exponent $r-1$
 - Terms with exponents $< r-1$
- Subtracted r from each exponent in first set
- Added first and third vectors together

Division Optimization



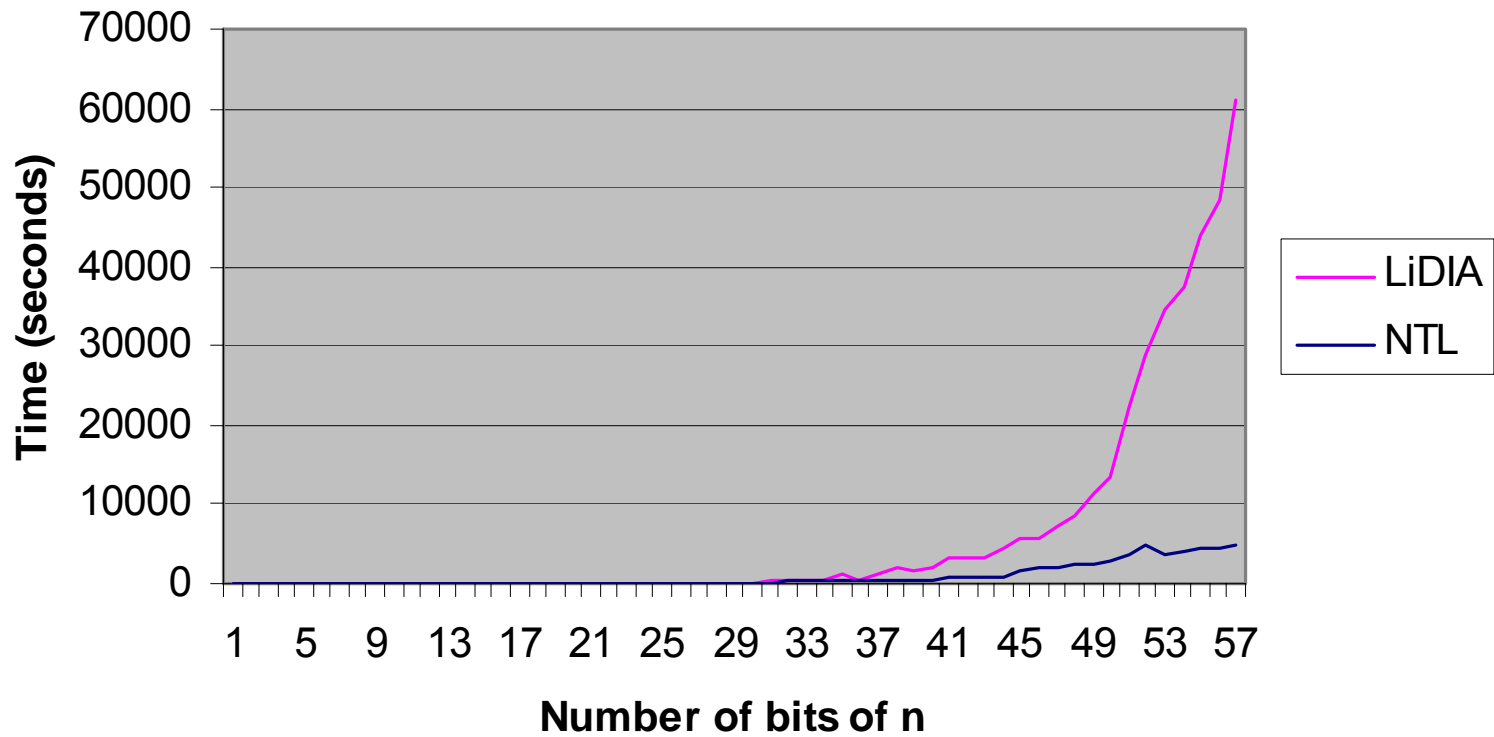
Squaring “Optimization”

- Constructed squaring routine
 - Dealt with first and last terms individually
 - Looped through all terms pairing together
- 91% of test in squaring function
- Operation times actually increased
 - Confirmed problem using GMP

NTL Implementation

- Efficient polynomial multiplication
 - LiDIA used only classical multiplication
 - NTL Automatically chooses from:
 - Classical algorithm
 - Karatsuba method
 - Two Fast Fourier Transform (FFT) methods
- Dramatic performance increase
 - 50% of time spent in FFT
 - May be able to adjust selection boundaries

LiDIA vs. NTL

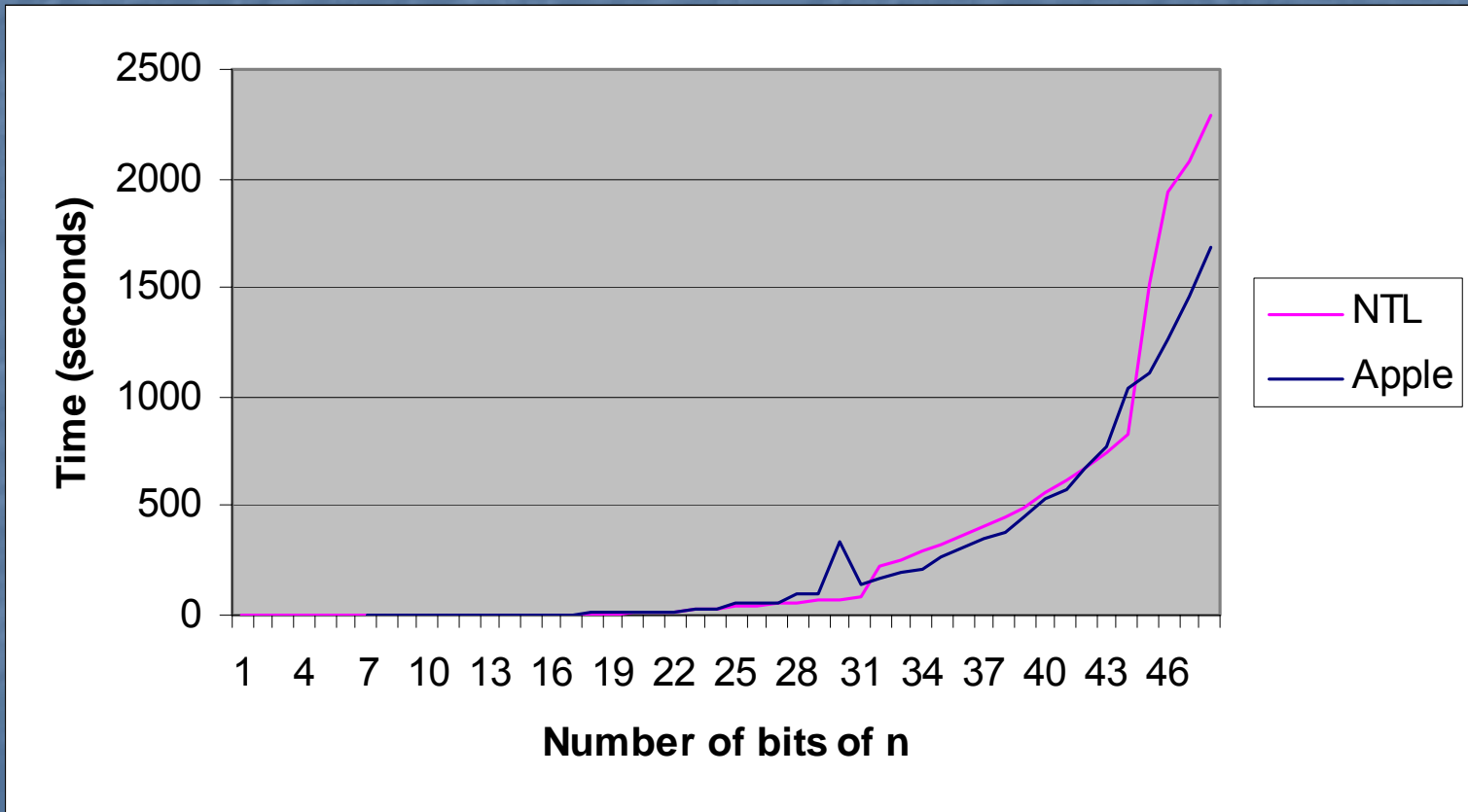


NTL Potential

- Implement native Perfect Power test
 - Currently falling back to GMP
- Possible Alternate Implementations
 - Manually reduce coefficients
 - Represent polynomial directly as a ring
 - Let NTL handle polynomial reduction

Performance Comparison

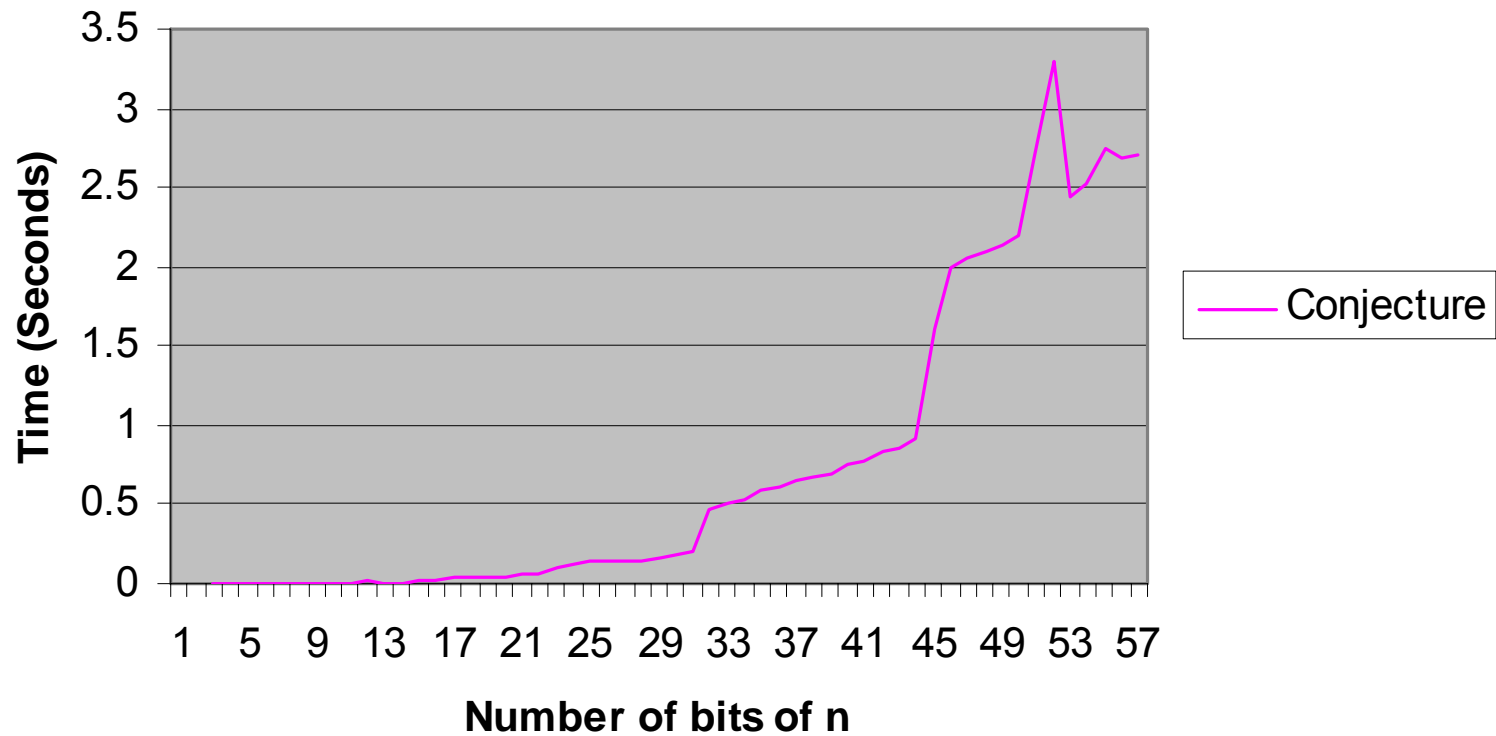
Crandall & Papadopoulos
Apple / UMD



Conjecture Improvement

- Improves time to $\tilde{O}(\log^3(n))$
- Conjecture:
 - If $n \bmod r \neq 0 \bmod r$ and
 - If $(X - 1)^n \equiv (X^n - 1) \pmod{(X^r - 1, n)}$
 - And $n^2 \bmod r \neq 1 \bmod r$
 - n is PRIME
- From tests this appears to be true
 - If tests holds for $a = -1$ then it holds for all a
- Conjecture is still not proven

Extrapolation with Conjecture



Future Work

- Native GMP Implementation
 - Allow total control over polynomial operations
 - Very difficult to do more efficiently than NTL
- Bernstein Exponentiation Optimization
 - Map polynomial ring onto integer ring
- Qi Cheng Method
 - One round of Elliptic Curve Primality Test
 - One iteration of AKS

Conclusions

- Our implementation comparable to others'
- NTL better suited to AKS than LiDIA
- AKS still too slow for practical use
 - Conjecture
 - Bernstein / Qi Cheng
- Focus should be on algorithmic improvements