Department of Computer Science
Rochester Institute of Technology
Master's Project Proposal

Polynomial Time Primality Testing Algorithm

Takeshi Aoyama

February 6, 2003

Committee

Chairman: Stanisław P. Radziszowski
Reader: Edith Hemaspaandra
Observer: Peter G. Anderson

Abstract

Last summer (August 2002), three Indian researchers, Manindra Agrawal and his students Neeraj Kayal and Nitin Saxena at the Indian Institute of Technology in Kanpur, presented a remarkable algorithm (AKS algorithm) in their paper "PRIMES is in P." It is the deterministic polynomial-time primality testing algorithm that determines whether an input number is prime or composite. Until now, there was no known deterministic polynomial-time primality testing algorithm, thus it has fundamental meaning for complexity theory.

This project will center around the AKS algorithm from "PRIMES is in P" paper. The objectives are both the AKS algorithm itself and theorems and lemmas showing the correctness of the algorithm. The rst task of the project is to provide easy-to-follow explanations of the paper for average readers. The second task is to analyze the AKS algorithm in detail. Ideas and concepts in the algorithm will be studied and possibilities of improvement of the algorithm will be explored. Some simple programs will be made for these experiments according to need.

1 Overview of the area of the project

Primality testing is a decision problem because if we are asked whether the given positive integer is prime or not, the answer must be "yes" or "no." AKS algorithm proved primality testing is in the class P. That is why the title of the paper is "PRIMES is in P." PRIMES represents primality testing in their terminology. The class P problem can be solved in polynomial time by the size of the input of the problem. A method using the bit length of the given number n (i.e. $\log_2 n$) as the size is reasonable in PRIMES. Whether PRIMES is in the class P or not had been one of the most important problems in the theoretical computer science eld for a long time.[25]
Here is the brief history of the primaly testing.[26]

Eratosthenes, 276 BC-194 BC: the Eratosthenes Sieve

Pratt '75: in NP

Miller '76: $O((\log_2 n)^4)$-time solvable if the Extended Riemann Hypothesis is true

Solovay and Strassen '77; Rabin '80: in coRP, still the choice in applications

Adleman, Pomerance, and Rumely '83: deterministic $O((log_2 n)^{log_2 log_2 log_2 n})$-time

Goldwasser and Kilian, '86: "Almost all" primes can be proven to be prime $O((log_2 n)^{12})$ time

Adlema and Huang '87: in RP

Fellows and Koblitz '92: in UP

Agrawal, Kayal and Saxena 2002: in P, $O((\log_2 n)^{12} poly(\log_2 \log_2 n))$-time

Complexity Classes[25],[27]

NP(Nondeterministic Polynomial-time) If we can magically get some extra input, we can check if the given number is prime or not in polynomial time

coRP(complementary Randomized Polynomial-time) (The answer "prime" is probably right, "composite" is certainly right)

RP (Randomized Polynomial-time) Solvable in polynomial time by randomized algorithms (The answer "composite" is probably right, "prime" is certainly right)

UP(Unambiguous Polynomial time) The class of sets that are accepted by non-deterministic polynomial-time Turing machines that on no input have more than one accepting computation path [28]

P(polynomial-time) Solvable on a deterministic sequential machine in an amount of time that is polynomial in the size of the input

---

Many researchers have been interested in the primality testing algorithm from a theoretical standpoint. But some primality testing algorithm are also important from a practical standpoint because some cryptosystems such as RSA use this kind of algorithm for their key generation. RSA needs two large prime numbers for their key generation and a primality testing algorithm is used in this part of the system.[2]

Consequently, many researchers have provided various algorithms. There are some very ecient algorithms such as the Solovay-Strassen or Miller-Rabin algorithm. They are termed probabilistic polynomial time primality testing algorithms. (It is same as the class RP.) The term "probabilistic" implies that they can make mistakes sometimes. That is to say there is a possibility that the algorithm may claim the given number is prime when in fact it is composite. But this error possibility can be reduced to an arbitrarily small percentage by repeating the test enough times. The algorithm is basically very fast, so the algorithm is practical enough in real cryptosystems.

Despite such eorts of many researchers, no one could nd the algorithm in the class P up to AKS algorithm. Therefore, people believed that very complex mathematical techniques were necessary to solve this problem. However, Agrawal et al. accomplished their important achievement by using only a relatively basic number theory and group theory. That further surprised many researchers.

Probabilistic primality testing

Is the given n prime?

Probability of n is (in fact) prime or composite

answer of the algorithm	prime	composite
yes(prime)	1-p	p
no(composite)	0	1

sometimes wrong ($p < 0.5$ is the error rate)

always correct

Repeating test

```
for i = 1 to k
    Test n;
    if the answer is yes
        continue;
    else
        output COMPOSITE;
}
output PRIME;
```

If the output is PRIME, the error rate is p^k.

---

2 Project Description

Studying this new AKS algorithm and the paper "PRIMES is in P" is the main purpose of the project. The project will consist of two phases.

2.1 Phase 1: Understanding the paper

The authors of the paper gave some theorems, lemmas and their proofs in the paper in order to show the correctness of their algorithm. But the paper itself is only 9 pages and the authors did not describe their proofs in detail, i.e. there are some gaps in their proofs. So, the paper is dicult to easily understand for average level readers.

In phase 1 the paper will be read in detail and a complete explanation will be given.

Some examples, charts and graphs will also be provided to help readers to understand the meaning of the paper more easily. Some basic programs may be provided if they are necessary to generate these examples or other charts, etc.

There are a number of theorems quoted from other papers, but the proofs are not given in this paper. Outlines of original papers will be presented to support readers' understanding of the materials. Lemma 3.2 and 3.3 will be included in that.

Lemma 3.2 [1] Let $P(n)$ denote the greatest prime divisor of n. There exist constants $c > 0$ and n_0 such that, for all $x > n_0$

$$|\{p|p \text{ is prime}, p \quad x \text{ and } P(p \quad 1) > x^{\frac{2}{3}}\}| \quad c\frac{x}{\log x}$$

Lemma 3.3 [1] Let (n) be the number of primes n. Then for n 1:

$$\frac{n}{6\log n} \qquad (n) \quad \frac{8n}{\log n}$$

Generally, binary logarithms are often used when we mention a complexity of an algorithm because they represent the bit length of numbers. On the other hand, natural logarithms are used for describing the distribution of prime numbers. In "PRIMES is in P" the same notation "$\log n$" is used for both binary logarithms and natural logarithms. This causes confusion to readers. They will be claried in the project by specifying the bases of these logarithms and the eect they have on the exactness of the proofs will also be considered.

Binary Logarithm: $\log_2 n$

Natural Logarithm: $\log_e n$

Moreover, minimal basics of number theory and group theory needed to understand the paper will be put in as an appendix of the project report. This will help readers to understand the contents without referencing other textbooks or documents as much. Anticipated contents are following.

Modulo Multiplication Group

Galois Extension Field

etc.

2.2 Phase 2: Analyzing the algorithm

AKS algorithm [1]

Input: integer $n > 1$

1. if (n is of the form $a^b, b > 1$) output COMPOSITE;
2. $r = 2$;
3. while($r < n$) {
4. if (gcd$(n, r) \neq 1$) output COMPOSITE;
5. if (r is prime)
6. let q be the largest prime factor of r 1;
7. if (q $4\sqrt{r}\log n$) and ($n^{\frac{r-1}{q}}$ / 1 (mod r))
8. break;
9. r $r + 1$;
10. }
11. for $a = 1$ to $2\sqrt{r}\log n$
12. if ($(x$ $a)^n$ / $(x^n$ $a)$ (mod x^r $1, n)$) output COMPOSITE;
13. output PRIME;

In phase 2, some important ideas or concepts of the algorithm will be analyzed more closely. Topics of this phase should be more specied through phase 1. At this time the following are planned. All of them are not necessarily included in the actual project. It depends on the whole size of the project and the relevance of the problem.

2.2.1. Analyze the relation between n and r n is the number whose primality should be tested in the algorithm. The while-loop in line 3 to 10 tries to nd r such that q is the largest prime factor of r, q $4\sqrt{r}\log n$ and $n^{\frac{r-1}{q}}$ / 1 (mod r). Since the upper bound of the for-loop in line 11 to 12 and the amount of the modular computation in line 12 depend on r, the size of r is crucial for the complexity of the algorithm.

The fact that at least one such r always exists in the interval [$c_1(\log n)^6, c_2(\log n)^6$] is proved in the paper. Consequently, the whole complexity of the algorithm is estimated at $\tilde{O}(\log_2^{12} n)$, which is polynomial time, by the authors. ($\tilde{O}(t(n)) = O(t(n)poly(\log_2 t(n)))$

The existence of r in [$c_1(\log n)^6, c_2(\log n)^6$] is theoretically guaranteed. However, actually we may be able to nd much smaller r. That implies the possibility of making the complexity of the algorithm smaller. First of all, the relation between r and n is not obvious. One r is expected to correspond to some $n's$ and vice versa. The project will examine that relationship experimentally and check to see if some patterns exist in this relationship.

2.2.2. Change the upper bound of the for-loop The required upper bound of the for-loop in the paper is $2\sqrt{r}\log_2 n$. This is the theoretical upper bound for working the algorithm correctly. In other words, following congruence holds for all a's such that $1 \quad a < 2\sqrt{r}\log_2 n$, if and only if n is a prime or a composite number of the form p^k. (p is prime)

$$(x \quad a)^n \quad (x^n \quad a) \pmod{x^r \quad 1, n}$$

In this part the upper bound will be made lower. Then the point of which the correctness of the algorithm is broken will be found. That means if the upper bound is changed to lower, some non p^k type composite numbers can satisfy the congruence for all a. Thus, the algorithm can not detect these numbers as composite numbers.

Although such composite numbers may not be found even if the upper bound is down to a certain point, and the algorithm still can work correctly. Again, that would suggest a possibility of the improvement of the algorithm.

2.2.3. Options The following ideas are optional. They may be included if time permits. Otherwise, they will be left as future work.

1. Find the real bounds of some constants described in the paper theoretically.
2. Measure the cpu-time of each part of the algorithm and compare them with theoretical values.
3. Think of average-case time complexity not but worst-case time complexity.
4. Examine eects that dierent types of composite numbers give on the performance of the algorithm. Dierent types mean that a composite number has many factors or few factors and the variance of the size of factors is large or small etc.

3 Principal Deliverables

A project report (LaTeX, PDF) that contains graphs, charts etc.

1. Detailed explanation of "PRIMES is in P"
2. Design and result of experiment

Program source code used for analysis

1. Binary exponentiation for numbers and polynomials
2. Find the largest prime factor
3. Calculate the GCD
4. Test if the given number is of the form a^b

Program guide and usage

4 Time Table

Components	**Anticipated Completion Date**
Phase 1	
Read the paper	Nov./17/02
Write	Dec./01/02
Proposal	Jan./19/03
Phase 2 (n and r)	
Experimental design and software implementation	Jan./26/03
Experiment	Feb./09/03
Phase 2 (change upper bound)	
Experimental design and software implementation	Feb./23/03
Experiment	Mar./09/03
Write up	Mar./23/03
Defense	Mar./30/03

5 References

References

[1] Manindra Agrawal, Neeraj Kayal and Nitin Saxena. PRIMES is in P. Available at http://www.cse.iitk.ac.in/news/primality.html.

[2] T. M. Apostol. *Introduction to Analytic Number Theory.* Springer-Verlag, 1976.

[3] R. C. Baker and G. Harman. The Brun-Titchmarsh Theorem on average. In *Proceedings of a conference in Honor of Heini Halberstam, Volume 1,* pages 39-103, 1996.

[4] Rajat Bhattacharjee and Prashant Pandey. Primality testing. Technical report, IIT Kanpur, 2001. Available at http://www.cse.iitk.ac.in/research/btp2001/primality.html.

[5] Chris Caldwell. *The Prime Pages* http://www.utm.edu/research/primes/

[6] E. Fouvry. Theoreme de Brun-Titchmarsh; application au theoreme de Fermat. *Invent. Math.,*79:383-407, 1985.

[7] Hisashi Fukagawa *Miller-Rabin ni yoru sosuu no kakuritsuteki hanteihou.* http://www.h-fukagawa.com/documents/miller-rabin.pdf

[8] Yves Gallot. *Implementation of the AKS algorithm.* http://perso.wanadoo.fr/yves.gallot/src/

[9] Eitan Gurari. *An Introduction to the Theory of Computation.* Ohio State University http://www.cis.ohio-state.edu/~gurari/theory-bk/theory-bk.html

[10] G. H. Hardy and J. E. Littlewood. Some problems of 'Partitio Numerorum' III: On the expres-sion of a number as a sum of primes. *Acta Mathematica*, 44:1-70, 1922.

[11] Neeraj Kayal and Nitin Saxena. Towards a deterministic polynomial-time test. Technical report, IIT Kanpur, 2002. Available at http://www.cse.iitk.ac.in/research/btp2002/primality.html.

[12] R. Lidl and H. Niederreiter. *Introduction to nite elds and their applications* . Cambridge University Press, 1986.

[13] Hisanori Mishima. *Sugakusya no missitsu.* http://www.asahi-net.or.jp/˜KC2H-MSM/index.htm

[14] Mitsunori Ogihara and Stanisław Radziszowski. Agrawal-Kayal-Saxena Algorithm for Testing Primality in Polynomial Time. Available at http://www.cs.rit.edu/ spr/PUBL/primes.html

[15] Kabocha pi. *Kabocha suuron.* http://www.hcn.zaq.ne.jp/caaid806/math/num_thr.html

[16] Jorg Rothe. *On Some Promise Classes in Structural Complexity Theory.* http://www.informatik.uni-trier.de/GI/FG-014/PhDs/JoergRothe.html

[17] Douglas R. Stinson. *Cryptography : theory and practice.* CRC press LLC, 1995.

[18] Hiranouchi Toshiro. *Sosu daisuki.* http://homepage2.nifty.com/hiranouchi/

[19] Ken Uehara. *Coding Theory.* Department of Information Science, Saga University. http://www.ma.is.saga-u.ac.jp/uehara/coding.html

[20] Tomohisa Wada *Galois tai no kantanna setsumei.* Department of Information Engineering University of the Ryukyus. http://bw-www.ie.u-ryukyu.ac.jp/˜wada/vhdl/GaloisField.html

[21] Eric Weisstein. *World of Mathematics.* Wolfram Research. Dec.15 2002 http://mathworld.wolfram.com/

[22] Koichi Yamazaki. *Kakuritsu algorithm ni tsuite.* Department of Computer Science, Gunma University http://www.comp.cs.gunma-u.ac.jp/˜koichi/RAND/rand/rand.html

[23] *Shoto seisuron.* Aozora Gakuen Sugakuka http://www80.sakura.ne.jp/˜aozora/suuron/suuron.html

[24] *Kokaikagi ango RSA nyumon.* Yousei genjitsu. http://www.faireal.net/articles/5/24/#d20519

[25] *Seisuron.* Kiso Lab, Dept. of Information Engineering, Shinsyu university. http://markun.cs.shinshu-u.ac.jp/learn/integer/index.html

[26] *Math and Computing.* Nara University of Education.
http://www.nara-edu.ac.jp/~asait/

[27] *The PRIMES is in P little FAQ.* http://crypto.cs.mcgill.ca/ stiglic/PRIMES_P_FAQ.html

[28] *Computational complexity theory.* Wikipedia.
http://www.wikipedia.org/wiki/Complexity_theory_in_computation