A Polymomial Time Algorithm to Test if a Number is a Prime or Not

Manindra Agrawal (36), Professor of Computer Science and Engineering at the Indian Institute of Technology, Kanpur, along with two BTech students of his, Neeraj Kayal (22) and Nitin Saxena (21) have solved the age old problem of finding a 'polynomial time' algorithm that decides whether a given number is a prime or not. Such an algorithm has been eluding the best mathematicians and computer scientists for several decades. This problem besides being of intrinsic theoretical interest has many practical implications in designing encryption algorithms used to protect data stored in computers and transmitted via communication networks.

Since the late 1950s, with the advent of computers, there has been a shift of focus from finding a mathematical formula to devising an algorithm (a recipe of steps) that can test for primality of a given number, however large, in an efficient manner. This problem has received immense attention in recent years owing to the widespread use of prime numbers in encryption algorithms. Testing of primality thus became an important problem in theoretical computer science. If an algorithmic solution to the problem posed is found, then an executable computer programme can be written. From the perspective of theoretical computer science, therefore, two things are required. One, a certificate or proof that the algorithm does give the correct answer. Two, a measure of the efficiency of the algorithm, namely, how well the algorithm uses the computer resources – such as time or the number of steps in the algorithm, space or memory utilised – as a function of the 'input size' of the problem to obtain the solution. Consider the algorithm for ordinary multiplication. The input size for a given number n is the encoding of the number to execute the algorithm. For any given number n, the number of digits N in it is roughly $\log n$. As is easily seen, the number of steps involved in the simple school algorithm for multiplication of two N-digit numbers is proportional to N^2. The number of steps involved is thus at best some power of N; that is, a polynomial in the input size. Therefore, the time taken by a computer program, written to execute the multiplication algorithm, will also grow with the number of digits at best as some power of N. Contrast this with the school algorithm for primality testing. To test the primality of small numbers, say all those less than 10^{10}, one of the most efficient algorithms is the following ancient algorithm due to the Greek mathematician Eratosthenes (circa 240 BC). To test a number n for primality, just divide by all the primes less than the square root of n. For example, to determine if 211 is a prime, we just divide by 2, 3, 5, 7, 11 and 13. The number of steps involved here grows as the number of divisions that one has to do, namely $\sqrt{n}$ Given a number N digits long its magnitude n is approximately 10^N. Thus the

number of divisions needed to find if it is a prime number is $\sqrt{10^N}$ For $N = 100$ the algorithm grows exponentially with the length of the number. So if $N = 100$, the time taken for the algorithm to return an answer could take longer than one's lifetime. One can easily imagine input sizes for which programs would be running for ever!

All the earlier approaches for finding polynomial time algorithm for primality testing have started with the basic equation that is satisfied by all primes, known as the Fermat's little theorem. In 1976, G Miller used Fermat's little theorem and assumed the validity of the as-yet-unproven and deep conjecture in analytic number theory known as the extended Riemann hypothesis (ERH) and obtained a deterministic polynomial-time algorithm. This was modified in 1980 by M G Rabin, who introduced a probability function (random coin tosses) and obtained a much more efficient, unconditional (that is, without using any unproven hypotheses like ERH) but randomised polynomial-time algorithm. If the number was a prime, the Rabin algorithm gave the correct answer, but if it was composite, it sometimes (i.e., with very low probability) determined it to be prime. However, the Miller–Rabin algorithm is a very efficient test for practical real-life implementation in areas such as cryptography because the level of error is extremely small and the probability of getting a correct answer far outweighs the error probability. Since then a number of randomised algorithms in polynomial time have emerged. Indeed, if randomisation had to be given up it seemed one had to invoke some condition such as the ERH. A breakthrough came in 1983 when L M Adleman, C Pomerance and R S Rumeley (APR) gave a deterministic and unconditional algorithm but in an almost polynomial time – it gave answers in $N^{\log \log N}$ steps – while all the earlier deterministic and unconditional algorithms required exponential time. This was, however, much less efficient than the Miller–Rabin algorithm. This was further improved in 1984 by H Cohen and A K Lenstra to yield the primality of a 100-digit number in a matter of seconds. The next big leap in primality testing came in 1986 with the work of S Goldwasser and J Kilian who gave a randomised algorithm using some properties of a class of curves known as elliptic curves. Adleman and D Huang improved upon the Goldwasser–Kilian algorithm to derive a randomised polynomial-time algorithm that also always produced a proof of the correctness of the primality testing. Given the trend in the field during the 1980s, the general belief in the community was there should exist a polynomial time algorithm for proving primality, but finding it was going to be very difficult. The great strength of the Agrawal–Kayal–Saxena (AKS) algorithm is that it is relatively simple, not requiring fancy mathematics like elliptic curves, etc. It is deterministic and relies on no unproven assumptions. The reason for the AKS algorithm's success is simple. Unlike others who believed that the elliptic curve method was the only viable approach to the problem and, as in any other

field, the newer algorithms were basically incremental advances over the Goldwasser and Kilian approach, AKS chose to abandon the well-trodden approach altogether and explored new lines of attack on the problem using simple concepts of algebra. The key to AKS result is a new, generalised version of Fermat's little theorem, a polynomial equation. The algorithm has been shown to run in polynomial time at most as N^{12} where N is the number of digits. For a special class of primes known as Sophie–Germain primes, using a widely believed conjecture on their density, the algorithmic time complexity is reduced to N^6. AKS however believe that one should be able to do better than this and the time taken should go as N^4. The algorithm was posted on IIT/K website on Aug.6, 2002, and immediately attracted world-wide attention. It has been critically evaluated and called the best result in theoretical computer science this decade.

Manindra Agrawal is a product of IIT/Kanpur where he obtained his B.Tech in Computer Science and Engineering and a PhD in 1991. He worked with Prof. Somenath Biswas on complexity theory. After a few years at Chennai Mathematics Institute, he joined IIT/Kanpur faculty in 1996. Both Kayal and Saxena completed their BTech in Computer Science and Engineering at IIT/Kanpur in May 2002. The work on primality algorithm was their undergraduate project. They are currently pursuing PhD at IIT/Kanpur.

Left to right –
Nitin Saxena, Neeraj Kayal, Manindra Agrawal.

Communicated by Neeraj Kayal and Nitin Saxena, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur 208 016, India.

On Primes

"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length ... . Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated."

– Karl Friedrich Gauss
Disquisitiones Arithmeticae, 1801
(translation from D E Knuth, The art of computing, Vol.2, Addison Wesley, 1998)