Neural, Parallel, and Scientific Computations 19 (2011) 331-344

SCOPE OF CONVENTIONAL PRIME TESTING AGAINST POLYNOMIAL-TIME AKS ALGORITHMS: MATLAB CODES

SUJA RAMAKRISHNAN AND SYAMAL K. SEN

Department of Mathematical Sciences, Florida Institute of Technology
150 W. University Blvd, Melbourne, Florida 32901, USA

ABSTRACT. The age-old conventional algorithm tests whether a given number n is prime or not by dividing n by all odd/prime numbers up to $\sqrt{n}$ and is exponential-time. In the background of polynomial-time AKS algorithms, the conventional algorithm still has practical/real-world usage. AKS method based algorithms do not appear to perform better than the conventional algorithm as long as the length of the number is not sufficiently large, the innovative implementations/variations of AKS algorithm have improved and are progressively improving and bringing down the complexity marginally lower than that of the original AKS algorithm though. In this paper we have explored the scope of conventional prime testing through the study of computational/time complexities of some of the recent implementations. It appears that the number n to be tested should be 23 digit or more long for polynomial-time algorithms to be less expensive than the conventional one computationally.

AMS (MOS) Subject Classification. 11A51. 11Y11. 03D15

Key words and phrases: AKS algorithm, Computational/time complexities, Matlab Implementation, Primality testing, Prime numbers.

1. INTRODUCTION

There exist various algorithms for testing/proving whether a number n is prime. These prime testing algorithms can be divided into two classes – deterministic and probabilistic. While the deterministic methods definitely declare whether a number n is prime or composite, the probabilistic tests, such as the Miller-Rabin test and Fermat's primality test based on Fermat's little theorem [14, 15, 16] declare a number to be definitely composite or only probably prime. These procedures use a probabilistic search for the proof of compositeness of n. Fermat's Little theorem results in some composite numbers identified as prime – such numbers being called pseudoprime.

The scope of this paper is restricted to the study of deterministic algorithms used for primality testing – the conventional algorithm and the AKS class of algorithms [1, 2, 3, 5, 9]. The conventional algorithm to be studied is the trial division where the number n to be tested is divided by every number starting from 2 to $\lfloor(\sqrt{n})\rfloor$. The computational complexity of this conventional algorithm is exponential-time $O(10^{0.5log_{10}(n)})$ in the number of digits, viz., $log_{10}(n)$ in n. The run time of the

Received August 29, 2011

AKS algorithms, on the other hand, is bounded by a polynomial in the number of digits of n. In this paper, we make a comparison of these deterministic algorithms, and determine under which circumstances the conventional algorithm performs better over the polynomial-time AKS class of algorithms.

All the algorithms proposed for primality testing prior to the AKS algorithms were random (Fermat's primality test), exponential(conventional algorithm) or conditional (Miller-Rabin based on the unproven General Reimann Hypothesis). The AKS algorithm, named after its discoverers Agarwal, Kayal and Saxena, is the first of its kind in the sense that, it is deterministic, polynomial, and unconditional.

In section 2 we present the conventional algorithm along with the AKS algorithm and two of its variants – one published later by the authors themselves and another proposed by Bernstein, with the concerned theoretical run time complexities. In section 3 we present the MATLAB code for the foregoing four algorithms. Numerical examples are included in section 4 along with the concerned computational time-complexities while section 5 comprises conclusions.

Notation. We use lg for logarithm to the base 2, ln for natural logarithm and log for logarithm to the base 10. $\phi(r)$ is Euler's totient function of integer r, $o_r(n)$ is the multiplicative order of n modulo r. An Appendix provided at the end gives more information on these terms involved in the AKS algorithms.

2. THE ALGORITHMS AND THEIR ASYMPTOTIC TIME COMPLEXITIES

2.1. Algorithm 1 - Conventional algorithm.

Input integer $n > 1$.

1. for $j = 2$ to $\lfloor(\sqrt{n})\rfloor$ do
2. if $\mathrm{mod}(n, j) = 0$ then output COMPOSITE; break; end if;
3. end for;
4. output PRIME.

The computational complexity of this algorithm is $O(10^{0.5d})$, where $d = log_{10}(n) =$ the number of digits of the given number n. This provides the lower bound on the time taken. It is clear that as n increases the number of operations also increases and for a very large n this conventional algorithm would practically be of no use because of intractability.

2.2. Algorithm 2 - Original AKS Algorithm.

Input integer $n > 1$.

1. if (n has the form a^b with $a > 1$ and $b > 1$) then output COMPOSITE; end if;
2. $r := 2$;

3. while $(r < n)$ do
4. if $(gcd(n, r) \neq 1)$ then output COMPOSITE; break; end if;
5. if (r is prime & $r > 2$) then let q be the largest factor of $r - 1$;
6. if $(q \geq 4 * \sqrt{r} lg(n))$ and $(n^{(r-1)/q} \neq 1 (\text{mod } r))$ then break; end if;
7. end if;
8. $r := r + 1$;
9. end while;
10. if $n \leq r$ then output PRIME; end if;
11. for $a = 1$ to $2 * \sqrt{r} * lg(n)$
12. if $((X - a)^n \not\equiv (X^n - a) (\text{mod } X^r - 1, n))$ output COMPOSITE; break; end if.
13. end for;
14. Output PRIME.

This original algorithm has a complexity of $O((lg(n))^{12})$

2.3. **Algorithm 3 - Modified AKS Algorithm.**

Input integer $n > 1$.

1. if (n has the form a^b with $a > 1$ and $b > 1$) then output COMPOSITE; end;
2. Find the smallest r such that $o_r(n) > lg^2 n$
3. if $1 < gcd(a, n) < n$ for some $a \leq r$ then output COMPOSITE; end if;
4. if $n \leq r$ output PRIME; end if;
5. for $a = 1$ to $\lfloor \sqrt{\phi(r)} * lg(n) \rfloor$
6. if $((X - a)^n \not\equiv (X^n - a) (\text{mod } X^r - 1, n))$ then output COMPOSITE; end if;
7. end for
8. Output PRIME.

This modified algorithm has a complexity of $O((lg(n))^{7.5})$

2.4. **Algorithm 4 - Bernstein variation of the AKS Algorithm.**

Input integer $n > 1$.

1. if (n has the form a^b with $a > 1$ and $b > 1$) then output COMPOSITE; end if;
2. Find prime r, q – the largest prime factor of $r - 1$ and S such that $\binom{S+q-1}{S} \geq n^{2*\lfloor \sqrt{r} \rfloor}$ & $n^{(r-1)/q} \bmod r > 1$
3. if $1 < gcd(a, n) < n$ for some $a \leq r$ then output COMPOSITE; end if;
4. if $n \leq r$ output PRIME; end if;
5. for $a = 1$ to S
6. if $((X - a)^n \not\equiv (X^n - a) (\text{mod } X^r - 1, n))$ output COMPOSITE; break; end if;
7. end for;
8. Output PRIME;

From our experimental results we observed that this algorithm's computational complexity lies between $O(lg^{12}(n))$ and $O(lg^{7.5}(n))$.

The run time of the AKS algorithms is polynomial in the number of bits in n [1, 2, 3, 9].

3. MATLAB CODES

The Matlab codes for all the algorithms are given in the Appendix. The Matlab codes for the AKS algorithms make two function calls – (i) 'perfectpower' to determine if input n is of the form m^b, $m, b > 1$ and (ii) 'Powermod' to determine the value of n^j mod k for integers n, j, and k. The programs coded in Matlab for these two functions are also given in the Appendix.

Even though the AKS algorithm by itself is deterministic, the Matlab's *isprime* function used in writing the AKS programs is based on the probabilistic Miller-Rabin Test. One way of making it fully deterministic is by by storing all the prime numbers up to a certain limit in an array and then looping through that array to find the correct r. In Matlab, the command 'primes' can be used for this purpose. For example, primes(9999) produces a list of all the prime numbers present in the interval $[2, 9999]$. For the modified AKS algorithm even an 11-digit decimal number n requires a value of r that is only about 4 digit long. For example, when $n = 61757044093$, $r = 1373$. Hence primes(99999) command could be used to store all the possible values of r when n is 11-digit long. In $[2, 99999]$, one can readily verify in Matlab that there are 9592 prime numbers. In the worst case the number of times we loop through the array would be equal to $O(N)$ where N is the parameter passed to the function 'primes'.

4. RESULTS

Tests were performed to determine the average time taken by the AKS algorithm to find whether a number is prime in the cases of 3 to 11 decimal digit numbers. 30 randomly generated prime numbers were chosen for this purpose. The 'randseed' command in Matlab was used. This is compared with the times taken by the conventional method. The modified AKS algorithm was considered for test purposes since the original AKS algorithm and Bernstein variation were becoming increasingly intractable as the number of digits in n increased. The command $randseed('', 1, 30, 100, 999)$ generates a 1 x 30 row vector of primes numbers between 100 and 999 in older version of Matlab, viz., version $2007a$. However, in Matlab version 2010, this command is not available. Similar commands were used to generate prime numbers upto 8 decimal digits. The 'variable precision integer arithmetic'(vpi) package [12] was used to generate numbers upto 16 decimal digits. The vpi command is not available in current standard Matlab version.

Test results are summarized in a tabular form and also in the form of graphs. For each algorithm, its theoretical computational complexity is compared with the actual time-complexity obtained and these are presented in the form of graphs.

Table 1 gives the average time taken to test primality for integers ranging from *3* to *11* digits for the four algorithms. Graphical representation of the results in the table are presented in Figs. 1 - 5. The original AKS algorithm was run only for numbers up to 5 digits since it was intractable for larger n. For the same reason, we considered Bernstein's version of the AKS algorithm up to 9 digits.

Table 1. Average time taken (in seconds) by the four algorithms for testing primality of 3 - 11 digit integers

	Conventional	Original AKS	Bernstein	Modified AKS
3 digits	0.0001445	0.0454	0.8346	0.5151
4 digits	0.00043513	0.3851	5.9452	1.1565
5 digits	0.0013	$1.4252*10^6$	$1.8863*10^2$	11.6403
6 digits	0.0036	–	$1.7396*10^3$	75.8863
7 digits	0.0094	–	$4.8617*10^3$	$1.8159*10^2$
8 digits	0.0385	–	$1.6365*10^4$	$5.6722*10^2$
9 digits	0.0814	–	$4.6421*10^4$	$1.3159*10^3$
10 digits	0.1962	–	–	$4.3265*10^3$
11 digits	0.5322	–	–	$1.1430*10^4$

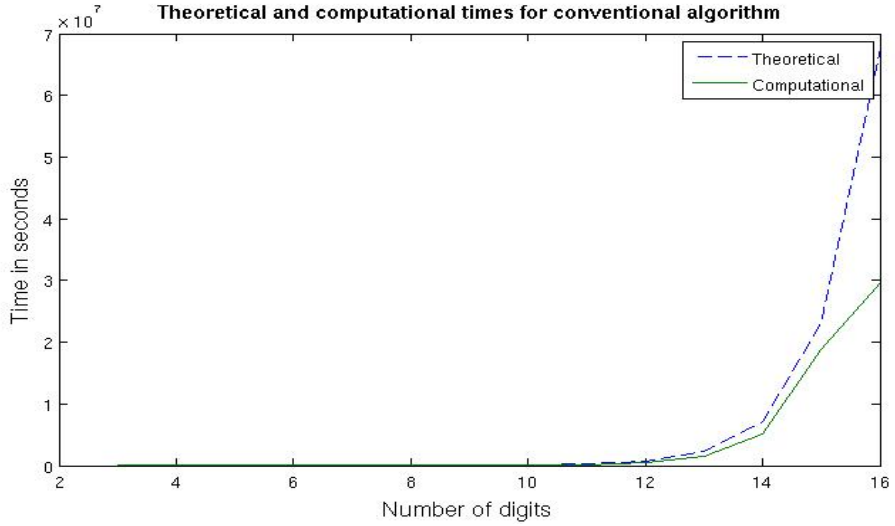

FIGURE 1. Theoretical computational complexity vs. actual time complexity of the conventional algorithm

5. CONCLUSIONS

It is clear that the AKS algorithm and its variations do not perform well for small numbers. Using Matlab we get an estimate on the value of n for AKS algorithms to

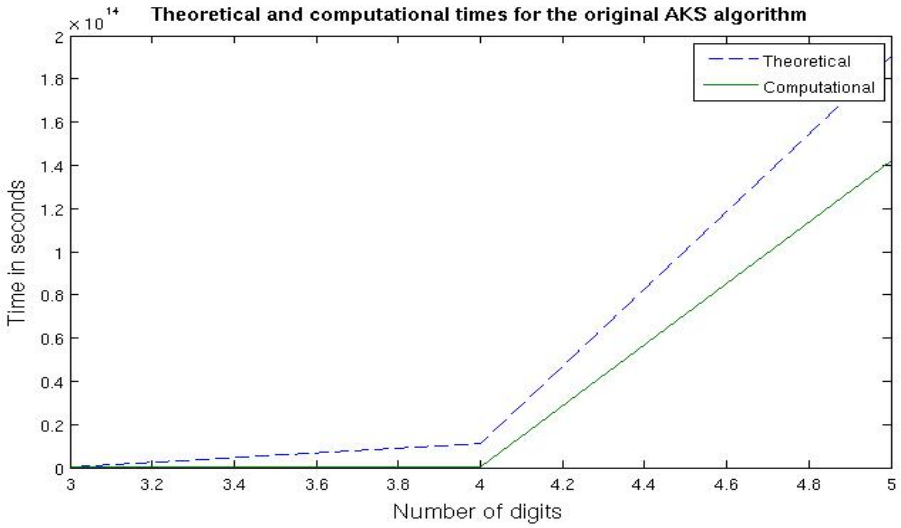

FIGURE 2. Theoretical computational complexity vs. actual time complexity of the original AKS algorithm

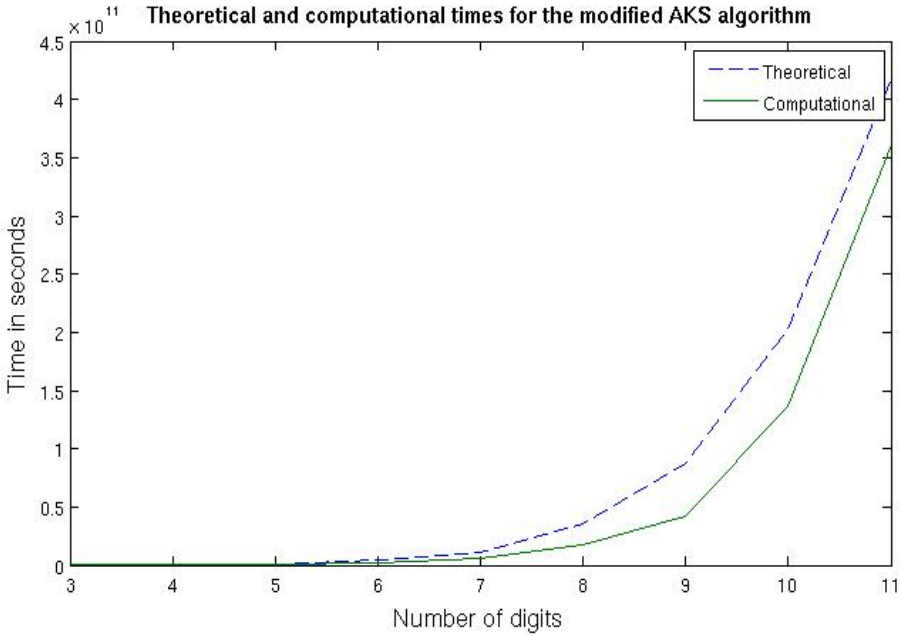

FIGURE 3. Theoretical computational complexity vs. actual time complexity of the modified AKS algorithm

perform better than the conventional algorithm by comparing $\sqrt{n}$ to $O(lg^6(n))$ as shown below.

```
>> n=vpa(2^74)
n = 18889465931478580854784.
>> lhs = 6*log(log(n)/log(2))/log(2)
lhs =
37.256720193773700227243995673419
```

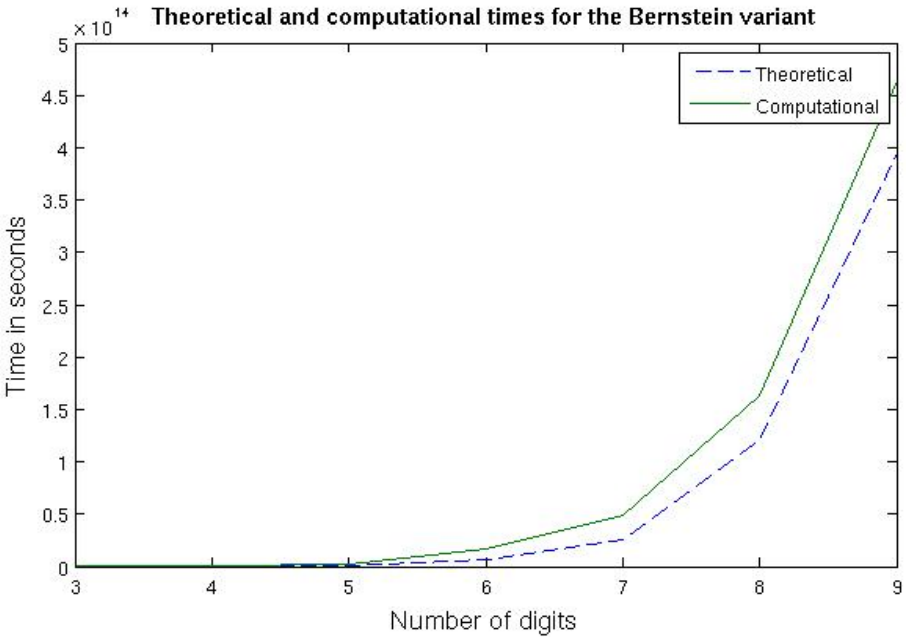

FIGURE 4. Theoretical computational complexity vs. actual time complexity of the Bernstein algorithm

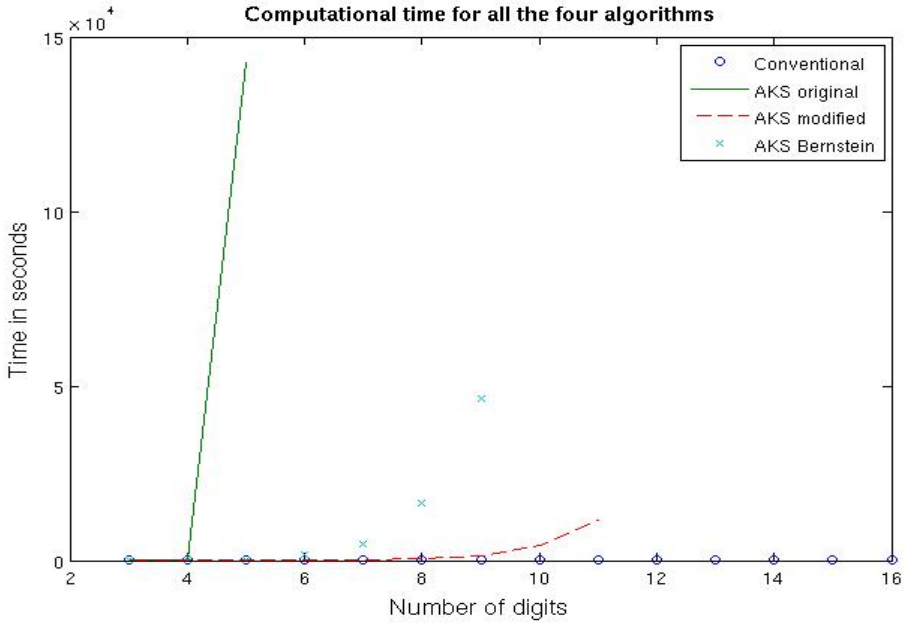

FIGURE 5. Actual time complexity for all the four algorithms

```
>> rhs = 0.5*log(n)/log(2)
rhs =
37.000000000000001237900615032400

>> n=vpa(2^75)
n = 37778931862957161709568.
```

```
>> lhs = 6*log(log(n)/log(2))/log(2)
lhs =
37.372912142975286803150845138231

>> rhs = 0.5*log(n)/log(2)
rhs =
37.500000000000001254629001722028
```

Hence $lhs < rhs$ when n has 23 digits. From the above results we can see that the AKS algorithm and its variations up till now is going to be competitive (more economical) only when number of digits of the prime number exceeds 23 (in decimal). The conventional method for testing primality of a number, which is exponential-time, will remain more efficient than the AKS algorithms so long as the number of digits of a prime number remains less than or equal to 23. Since logarithm to the base 2 of an integer (greater than 1) gives the number of binary digits in the integer, this also means that the integer to be tested must have greater than or equal to 75 bits in its binary representation. Since most functions of Matlab do not accept inputs greater than 2^{64}, we were not able to perform tests on such large numbers. Even though the AKS algorithms spend a long time in the polynomial test, as the value of n increases the value of r does not increase exponentially with the value of n.

Remark Changing the base of the logarithm from 2 to e in steps 2 and 6 of the modified AKS algorithm (Section 2.3) reduces the computation by 50%. If the base of the logarithms is made larger than e then the computation time will be reduced drastically. Unfortunately, this is not permitted by the AKS algorithms. For small numbers up to 8 digits we noticed that lg or ln do not make any difference.

We have used *Maple* functions 'modp' and 'Powmod' [7] for performing polynomial exponentiation and modular reduction. There is still scope for improvement in the Matlab programs in terms of bringing down the computational complexity of the AKS algorithms. All improvements to the original AKS algorithm focus on determining a much smaller value of r so that the number of times the polynomial congruence test is executed becomes much less and the polynomial $x^r - 1$ itself is of a smaller degree. This combined with fast polynomial reduction techniques is what would make AKS algorithms much faster. The practical usage of AKS algorithms is limited unless such techniques are discovered. Also, in most of the real-world situations prime numbers up to 10 or 12 digits are all that is used since most of the laptop/desktop computers have 15 digits as their standard precision. For instance, in error-free computation using multiple modulus residue arithmetic we will use a set of ten 10-digit prime numbers. This is equivalent to a single 100-digit prime number. This obviates the need of very large prime number which cannot be accommodated

in a standard precision of most computers. However, variable precision arithmetic (vpa) available in Matlab in a limited way may be used.

REFERENCES

[1] M. Agrawal, N. Kayal and N. Saxena, PRIMES is in P, *Annals of Mathematics*, 160:781–793, 2004, No. 2.

[2] D. J. Bernstein, *Proving Primality after Agarwal-Kayal-Saxena*, 2003.01.25, available online at http://cr.yp.to/papers/aks.pdf.

[3] D. J. Bernstein. *An exposition of the Agarwal-Kayal-Saxena primality-proving theorem*, preprint, August 2002. Available online at http://cr.yp.to/papers

[4] Giles Brassard and Paul Bratley, *Fundamentals of Algorithmics*, Prentice Hall, 1996.

[5] Richard Crandall and Carl Pomerance, *Prime Numbers - A Computational Perspective*, Chapter 4, Springer, 2nd edition, 2005.

[6] M. Dietzfelbinger, *Primality testing in polynomial time: from Randomized algorithms to "PRIMES is in P" (Lecture Notes in Computer Science 3000)*, Springer, 2004.

[7] http://fatphil.org/maths/AKS/#Implementations.

[8] Donald E. Knuth, *The Art of Computer Programming*, Volume 2, Section 4.5.2, Algorithm X.

[9] H. W. Lenstra and C. Pomerance, *Primality Testing with Gaussian periods*, Manuscript 2005.

[10] http://www.mathworks.com/help/toolbox/mupad/stdlib/isprime.html.

[11] http://www.mathworks.com/help/techdoc/ref/gcd.html.

[12] http://www.mathworks.com/matlabcentral/fileexchange/22725.

[13] R. G. Salembier and P. Southerington, *An Implementation of the AKS Primality Test*, Manuscript at http://teal.gmu.edu/courses/ECE746/project/F06_Project_resources/Salembier_Southerington_AKS.pdf

[14] http://en.wikipedia.org/wiki/Miller-Rabin_primality_test.

[15] http://en.wikipedia.org/wiki/Fermat's_primality_test.

[16] http://en.wikipedia.org/wiki/Fermat's_little_theorem.

APPENDIX

1. *Euler's totient function* The Euler's totient function for an integer r, denoted by $\phi(r)$ is the number of integers that are coprime to it. Two integers are called coprime if the only common divisor they have is 1. Sometimes this is also referred to as one number being relatively prime to the other. For example the integer 7 is coprime to 1, 2, 3, 4, 5, 6 or in other words, 7 is relatively prime to 1, 2, 3, 4, 5, 6. In general, if r is prime then $\phi(r) = r - 1$. If r is composite, then $\phi(r) = r(1 - 1/p_1)(1 - 1/p_2).....(1 - 1/p_k)$ where $p_i,\ i = 1, 2, ..., k$ are the distinct prime factors of r. In most of the AKS algorithms, the integer r is chosen such that it is prime and hence $\phi(r) = r - 1$.

2. *gcd* In order to determine the greatest common divisor (gcd) of two integers the Matlab function *gcd* is used in the Matlab programs of the AKS algorithms. This

function in Matlab is based in the polynomial-time Extended Euclid's algorithm [8, 11]. The complexity of this algorithm is $O(\log n)$.

3. *isprime* The *isprime* function in Matlab is based on the fast probabilistic Miller-Rabin primality test. It takes as input an integer n and gives as output 1 (if n is prime) or 0 (if n composite). If it returns 0 then n is definitely composite, however, if it returns 1 then n is most probably prime. The AKS algorithms coded in Matlab make a call to this function in order to choose the right prime r. The complexity of the step that uses this function is $O(kn^3)$ [10]. Hence, the AKS algorithm still remains polynomial time.

4. *Matlab code for Conventional Algorithm*

```
function conventionaltest(n)
prime = 1; tic;
if mod(n,2) == 0, display('The number is composite'), else j=3;
while j < floor(sqrt(n)), if mod(n,j)==0,
display('The number is composite'), prime = 0; break; end; j=j+2;
end; if prime == 1, display('The number is prime'), end; end; toc
```

5. *Matlab code for Original AKS Algorithm.*

```
function primesinpAKS(n)
tic, prime = 1;
%Checks if n is a perfect power of the form m^b
pp = perfectpower(n);  if pp == 1, display('Number is composite');
prime = 0;else, if n == 2, display('Number is prime'); prime = 0;
else, if mod(n,2) == 0, display('Number is composite'); prime = 0;
end; end; end; if prime==1, r=3; while r < n, if gcd(r,n) ~= 1,
disp('Number is composite'); prime = 0; break; else, if r > 2 &&
isprime(r)==1, q = max(factor(r-1)); if (q > 4*sqrt(r)*log2(n) &&
Powermod(n,(r-1)/q,r) ~= 1) s=floor(2*sqrt(r)*log2(n)); break; end;
end; end; r = r + 2; end; if n <= r, display('number is prime');
prime = 0; end; end; if prime == 1, syms x, for a = 1:s,
%Determine the binomial coefficient which exceeds the lower bound
%lb. Compute binomial coefficients taking logarithm on both sides
lhs = maple('modp',maple('Powmod',x - a, n, x^r-1, x),n);
rhs = maple('modp',maple('Power',x, mod(n,r)) - a,n);
if lhs ~= rhs, display('Number is composite'); prime = 0; break;
end;end; if prime == 1, display('The number is prime');end;end;toc
```

6. *Matlab code for Modified AKS algorithm*

```
function primesinpLenstra(n)
```

```
tic, prime = 1; %Check if n is a perfect power of the form m^b
pp = perfectpower(n); if pp == 1, prime = 0;
display('Number is composite'), else, if n == 2
display('Number is prime'); prime = 0; else, if mod(n,2) == 0
display('Number  is composite'); prime = 0; end; end; end;
if prime == 1 % Find smallest r such that multiplicative order of
%n mod r >= (log2(n))^2
log n2=log2(n)^2;q=ceil(log n2);while 1,foundr=1;for j=1:floor(log n2),
if Powermod(n,j,q)==1, foundr=0; break;end;end; if foundr==1, r=q;
break;end;q=q+1;end;display(r);if n <= r, display('Number is prime');
prime=0; else %If gcd(a,n) ~= 1 for any a <= r then output composite
m = 3; while m < r, if gcd(m,n) ~= 1, disp('Number is composite')
prime = 0; break; end; m = m + 2; end; end; end;
% Determine the Euler's totient function for r
if prime == 1, if isprime(r), etote = r - 1; else, factr=factor(r);
p(1) = factr(1); ct = 2; etote = r*(1 - (1/p(1)));
for i = 2:length(factr), if factr(i) ~= p(ct - 1), p(ct) = factr(i);
etote = etote*(1-(1/p(ct))); ct = ct + 1; end; end; end;
s = floor(sqrt(etote)*log2(n));
syms x, display(s), for a = 1:s
%If polynomial congruence relation holds for all a <=
%floor(sqrt(r-1)*log(n)) then output prime
lhs = maple('modp',maple('Powmod',x + a, n, x^r-1, x),n);
rhs = maple('modp',maple('Power',x, mod(n,r))+ a,n);
%If polynomial congruence relation fails for any a output composite
if lhs ~= rhs, prime=0; display('Number is composite'); break; end
end; end; if prime == 1, display('The number is prime'); end; toc
```

7. *Matlab code for Bernstein variation*

```
function primesinpDJB(n)
% This functions accepts input n > 1 to be tested for primality
tic, prime = 1; bool=0;
%Check if n is perfect power of the form m^b
pp=perfectpower(n);if pp==1,display('Number is composite');
prime = 0; else,if n==2,display('Number is prime');prime = 0;
else, if mod(n,2)==0,display('Number is composite');prime = 0;
end;end;end; if prime==1,r = 3; while r < n, if gcd(n,r) ~= 1,
disp('Number is composite');prime = 0;break;end; if isprime(r)
%Find q - the largest prime factor of r-1
q=max(factor(r-1)); %Call Powermod to compute mod(n^(r-1/q), r)
```

```
if Powermod(n,(r - 1)/q,r) > 1 %Compute lower bound lb to find s
lb=2*floor(sqrt(r))*log(n); l=0; for s = 1 : q
%Determine binomial coefficient which exceeds the lower bound lb.
%Easier to compute binomial coefficients taking log on both sides
l = l + log(q+s-1)-log(s); if(l >= lb),bool = 1; break;end;end;
end;end; if bool==1, break; end, r = r + 2; end; if n <= r,
display('the number is prime'); prime = 0; end; end; if prime==1
&& bool==1, display(r), display(s), syms x, for a = 1:s
%Make a call to maple library for computing the left and right
%sides of polynomial congruence relation
lhs = maple('modp',maple('Powmod',x + a, n, x^r-1, x),n);
rhs = maple('modp',maple('Power',x, mod(n,r))+ a,n);
if lhs ~= rhs, display('Number is composite'); prime = 0; break;
end;end;end; if prime==1, display('The number is prime');end;toc
```

8. *Multiplicative Order* The multiplicative order of n modulo r denoted by $\tilde{o}_r(n)$ is the smallest positive integer j such that $n^j \bmod r \equiv 1$. Since our requirement is that multiplicative order j should be an integer greater than $lg^2(n)$, we check if for any value $j = 1, 2, ..., \lceil lg^2(n) \rceil$, $n^j \equiv 1 (\bmod\ r)$. If it is so, then it means we have found a multiplicative order $r < lg^2(n)$ and hence we increment r by 1 and perform the same test again. Once an r, such that for all values of j, $n^j \bmod r \not\equiv 1$ is obtained, then we have found the smallest r [13].
9. *'perfectpower'* This function written in Matlab is the first step in AKS algorithms which determines if a number n to be tested is of the form m^b, $m, b > 1$, which if it is, then the output is COMPOSITE.

   ```
   function pp = perfectpower(n)
   pp=0; b=2; while 2^b <= n && pp==0, a=1; c=n; while (c - a) >= 2,
   m = floor((a+c)/2); p=min(m^b,n+1); if p == n,
   display('perfect power of the form m^b'); display(m); display(b);
   pp=1;break;else, if p < n, a=m;else, c = m;end;end;end;b=b+1;end
   ```

   This above algorithm has a complexity time of $O(\lg^4 n \lg \lg n)$ [6]. This can be brought down to $O(lg^2(n))$ using Newton iteration approach. This function is called by the AKS Matlab programs and has to be included in the appropriate folder prior to running the AKS programs.

10. *Polynomial congruence test.* For the AKS algorithm, the maximum time is utilized in testing the congruence relation $(X-a)^n \equiv (X^n-a) (\bmod\ X^r-1, n)$. This can also be stated as $\bmod(\bmod((X-a)^n, X^r-1)), n) \equiv \bmod(\bmod((X^n-a), X^r-1), n)$. To illustrate with an example, let the number n to be tested equal 7, which

we know is prime. Since r has to be a prime less than 7 (in the original AKS algorithm), let $r = 3$. Then the number of times the polynomial test has to be executed is $\lfloor\sqrt{\phi(r)} * lg(n)\rfloor = \lfloor\sqrt{2} * lg(7)\rfloor = 3$. Consider a = 1. Then the polynomial congruence relation that needs to be tested is $(X-a)^7 \equiv (X^7 - a)(\text{mod } X^3 - 1, 7)$. Now $(X-a)^7 = X^7 - 7X^6 + 21X^5 - 35X^4 + 35X^3 - 21X^2 + 7X - 1$. $((X-1)^7 \text{ mod } X^3 - 1) = X^4 + X$ and the next step is to perform mod operation on $X^4 + X$ with divisor 7 i.e. $(X^4 + X) \text{ mod } 7 = X^4 + X$. Similarly the right hand side of the congruence test gives $((X^7 + 1) \text{ mod } (X^3 - 1)) = X^4 + X$ and $(X^4 + X) \text{ mod } 7 = X^4 + X$. Since they are equal, the number is prime. It can be verified that this test will be satisfied for all values of $a \in [1, 3]$. Although the in the original AKS algorithm the polynomial test kicks in only for fairly large values of n, such small values were chosen to easily understand the test.

11. *'Powermod'* This function written in Matlab determines the remainder when n^j is divided by another integer q. This has a complexity of $O(\ln\ n)$ [4]. For a typical j the number of modular multiplications required is only $\frac{3}{2}lgn$

```
function b = Powermod(a,n,m)
n1=n; b=1; c=mod(a,m); while n1 > 0, if (rem(n1,2) == 1)
b=mod(b*c,m); end; c=mod(c*c,m); n1=floor(n1/2); end
```

This function is called by the AKS Matlab programs and has to be included in the appropriate folder prior to running the AKS programs.