A PRIME NUMBER INVESTIGATION USING BINARY STRINGS GENERATED BY APPLICATION OF THE EKADHIKENA PURVENA SUTRA

Marianne Fletcher

Abstract

The Ekadhikena Purvena sutra can be employed to calculate the number of digits before recurrence in the perfectly recurring decimal string for a rational number $1/N$, where N ends on the digits 1,3, 7 or 9. The number of digits, x, in one recurring cycle of the string can consequently be used to help determine the primality of N. This test is an application of Fermat′s Little Theorem. In his book, Vedic Mathematics, Sri Tirthaji gives examples of the working of the *Ekadhikena Purvena* sutra in base 10, with the result that decimal strings are calculated. This paper discusses the results obtained when the *Ekadhikena Purvena* sutra is applied to binary numbers (i.e. base 2), with the resultant generation of a recurring binary string for $1/N$. It was found that, when the computation is done in binary, a typical home computer can generate all the digits in the cyclic string related to $1/N$ at a rate several orders of magnitude higher than when the sutra is applied to decimal numbers. Such an application of the sutra thus hugely increases the rate at which the number N can be confirmed prime or not.

Background and Introduction: Prime Numbers and Fermat′s Little Theorem

Some important cryptographic algorithms critically depend on the fact that the prime factorization of very large numbers can take a long time. For many applications, the fast identification of large primes (often with several hundred digits) becomes very important.

Many primality tests have been developed and improved upon over the last century. The most obvious test if that of trial division: Given an input number n, check whether any prime integer m from 2 to $\sqrt{n}$ evenly divides n.

Probabilistic tests provide provable bounds on the probability of being fooled by a composite number. Some such tests are the Fermat Primality test and the Miller-Rabin test.

Deterministic Tests provide a definite determination of a prime number. Examples of such tests are the Pocklington Primality test, as well as the AKS Primality test.

Fermat's Primality test is based on Fermat's Little Theorem[1] which states that:

If p is prime and a is not divisible by p, then:

$$a^{p-1} \equiv 1(\mathrm{mod}\, p)$$

(a can be any base: a = 2, 3, 4, …)

This means that

$$\frac{a^{p-1}}{p} \text{ has a remainder of } 1$$

We can also write:

$$\frac{a^{p-1}}{p} = \text{Quotient} + \frac{1}{p} \quad \text{where the remainder} = 1$$

Some examples follow.

Example 1

7 is PRIME: Choose <u>base 10</u>

$$\begin{array}{r l} & 0,\ 1\ \ 4\ \ 2\ \ 8\ \ 5\ \ 7\ \ 1\ldots \\ 7 & \overline{)\,1,{}^{1}0\ {}^{3}0\ {}^{2}0\ {}^{6}0\ {}^{4}0\ {}^{5}0\,{}^{\boxed{1}}0\ldots} \end{array}$$

$$\frac{10^6}{7} = 142857.\dot{1}4285\dot{7}$$

$$= 142857 + \frac{1}{7} \qquad \text{where } \frac{1}{7} = 0.\dot{1}4285\dot{7}$$

Here, the remainder is 1.

It can, furthermore, be noted that the ***decimal*** string for $\frac{1}{7}$ has $7 - 1 = 6$ digits before recurrence. Also $(7 - 1)$ is divisible by the number of digits in the recurring ***decimal*** string:

$$\frac{(7-1)}{6} = 1$$

Example 2

79 is PRIME: Choose <u>base 10</u>

$$\begin{array}{r l} & 0,\ 0\ \ 1\ \ 2\ \ 6\ \ 5\ \ 8\ \ 2\ \ 2\ \ 7\ \ 8\ \ 4\ \ 8\ \ 1\ \ 0\ldots \\ 79 & \overline{)\,1,{}^{1}0\ 0\ {}^{21}0\ {}^{52}0\ {}^{46}0\ {}^{65}0\ {}^{18}0\ {}^{22}0\ {}^{62}0\ {}^{67}0\ {}^{38}0\ {}^{64}0\ {}^{8}0\,{}^{\boxed{1}}0\ldots} \end{array}$$

$$\frac{10^{79-1}}{79} = \frac{10^{78}}{79} = 1.26582278481\ 01265822\ldots \times 1076 + 0.\dot{0}12658227848\dot{1}$$

$$= 1.26582278481\ 0126582278481\ 0126582278481\ldots x\ 1076 + \frac{1}{79}$$

$$\text{where } \frac{1}{79} = 0.\dot{0}12658227848\dot{1}$$

Again, the remainder is 1. Also, the decimal string for $\frac{1}{79}$ has 13 digits before recurrence.

We find that

$$\frac{79-1}{13} = \frac{78}{13} = 6$$

So $(79 - 1)$ is divisible by the number of digits in the recurring ***decimal*** string.

Example 3

27 is not prime: Choose base 10

$$\frac{10^{27-1}}{27} = \frac{10^{26}}{27} = 3\ 703\ 703\ 703\ 703\ 703\ 703\ 703\ 703.\dot{7}0\dot{3}$$

$$= 3\ 703\ 703\ 703\ 703\ 703\ 703\ 703\ 703 \ + \frac{19}{27} \quad \text{where } \frac{19}{27} = 0.\dot{7}0\dot{3}$$

Here the remainder is 19, not 1. Also, the decimal string for $\frac{1}{27}$ has 3 digits before recurrence.

We find that

$$\frac{27-1}{3} = \frac{26}{3} = 8.\dot{6}$$

So (27 − 1) is NOT divisible by the number of digits in the recurring decimal string.

Fermat's Little Theorem in Terms of $(\frac{N-1}{x}) = k$

The preceding examples illustrate that Fermat's Little Theorem can be restated as follows: If *p* is prime and *a* is not divisible by *p*, then:

$$p - 1 \equiv k (\text{mod } x)$$

where *k* is a whole number and *x* is the number of recurring digits in the cyclic string for $\frac{1}{p}$, provided that the string is in the base a. This provision is linked to theorem 88 by Hardy and Wright[2] and is discussed in the section entitled "More on the *k*-values and the ekadhika".

Example 4

7 is PRIME: Choose base 2

$$\frac{2^{7-1}}{7} = \frac{2^6}{7} = \frac{64}{7} = 9.\dot{1}4285\dot{7}$$

$$= 9 + \frac{1}{7} \qquad \text{where } \frac{1}{7} = 0.\dot{1}4285\dot{7}$$

Using base 2, the remainder is still found to be 1.

However, to correctly investigate the *k*-value for a number *N* when using a base 2, it is necessary that the number of recurring digits *x* be determined for *N* in binary, not decimal. The binary representation of *N* = 7 is 111.

The binary string for $\frac{1}{7}$ is: $\frac{1}{111} = 0.\dot{0}0\dot{1}$. Here there are *x* = 3 recurring binary digits in the string.

Using $\frac{(N-1)}{x} = k$ we find that $\frac{7-1}{3} = \frac{6}{3} = 2$

So (7 − 1) is divisible by the number of digits in the recurring binary string.

Example 5

79 is PRIME: Choose base 2

$$\frac{2^{79-1}}{79} = \frac{2^{78}}{79}$$

$$= 3\,825\,714\,619\,033\,636\,628\,817.\dot{0}12658227848\dot{1}$$

$$= 3825714619033636628817 + \frac{1}{79}$$

where $\frac{1}{79} = 0.\dot{0}12658227848\dot{1}$

With base 2, the remainder is found to be 1. The binary representation of N = 79 is 1001111.

The *binary* string for $\frac{1}{79}$ is:

$$\frac{1}{1001111} = 0.\dot{0}0000011001111011001000111010010101000\dot{1}$$

Here there are x = 39 recurring digits in the binary string.

Using $\frac{(N-1)}{x} = k$ we find that $\frac{79-1}{39} = \frac{78}{39} = 2$

So (79 − 1) is divisible by the number of digits in the recurring binary string.

Example 6

27 is not prime: Choose base 2

$$\frac{2^{27-1}}{27} = \frac{2^{26}}{27} = 2485513.\dot{4}8\dot{1} = 2485513 + \frac{13}{27} \qquad \text{where } \frac{13}{27} = 0.\dot{4}8\dot{1}$$

Here the remainder is 13, not 1. The binary representation of 27 is 11011.

The binary string for $\frac{1}{27}$ is: $\frac{1}{11011} = 0.\dot{0}0001001011110110\dot{1}$. Here there are x = 18 recurring binary digits in the string.

Using $\frac{(N-1)}{x} = k$ we find that $\frac{27-1}{18} = \frac{26}{18} = 1.\dot{4}$

So (27 − 1) is NOT divisible by the number of digits in the recurring binary string.

The Fermat Primality Test and Pseudoprimes

Examples 1, 2, 4 and 5 demonstrate how, for any prime number N, (and any randomly chosen base a) $\frac{a^{N-1}}{N}$ always yields a remainder of 1. They also demonstrate how, for a prime, $\frac{(N-1)}{x} = k$ always yields a whole number k-value, where x is the number of recurring digits in the string for $\frac{1}{N}$, provided that the string is calculated in the base a.

Examples 3 and 6 demonstrate how most non-primes yield remainders which are unequal to 1, and k-values which are not whole numbers. So, the converse of Fermat's Little Theorem holds true in most cases,

i.e. *If* $\frac{a^{N-1}}{N}$ *yields a remainder of 1, or if (N − 1) is divisible by x, then N is prime.*

However, the conditions stated above cannot be used as a fool-proof test for primality, as some compound numbers - although in the minority - also meet these requirements.

Example 7

33 is not prime: Choose base 10

$$\begin{array}{r|l} & 0\,,\,0\;\;3\;\;0\;\;3\;\;0\;\;3\;\;0\ldots \\ \hline 33 & 1\,,\,{}^{1}0\;\;0\;\;{}^{1}0\;\;0\;\;{}^{1}0\;\;0\;\;{}^{1}0\ldots \end{array}$$

$$\frac{10^{33-1}}{33} = \frac{10^{32}}{33} = 3030303030303030303030303030303.\dot{0}\dot{3}$$

$$= 3030303030303030303030303030303 + \frac{1}{33} \quad \text{where } \frac{1}{33} = 0.\dot{0}\dot{3}$$

Contrary to what might be expected, we see that, while 33 is not a prime number, the remainder is found to equal 1. Also, the decimal string for $\frac{1}{33}$ has 2 digits before recurrence.

Testing with $\frac{(N-1)}{x} = k$ we find that $\frac{33-1}{2} = \frac{32}{2} = 16$

So here, for a non-prime, *k* is a whole number.

Example 8

However, if we choose base 2:

$$\frac{2^{33-1}}{33} = \frac{2^{32}}{33} = 130150524.\dot{1}\dot{2}$$

$$= 130150524 + \frac{4}{33} \quad \text{where} \quad \frac{4}{33} = 0.\dot{1}\dot{2}$$

Here we find that the remainder is not 1. Furthermore, the binary representation of 33 is 100001. The binary string for $\frac{1}{33}$ is: $\frac{1}{33} = \frac{1}{100001} = 0.\dot{0}00001111\dot{1}$.

There are $x = 10$ recurring binary digits in the string.

Using $\frac{(N-1)}{x} = k$ we find that $\frac{33-1}{10} = \frac{32}{10} = 3.2$

So $(33 - 1)$ is NOT divisible by the number of digits in the recurring binary string.

Thus, by inspection, we see that - while a prime number always yields a remainder of 1 (or a whole number *k*-value) - a non-prime may do so as well, depending on which base is chosen.

A non-prime which yields a whole number *k*-value is called a Fermat pseudoprime. Below 50 000, only 1.23 % of numbers which yield whole number *k*-values (for base 10) are pseudoprimes.

The conventional Fermat Probability test applied to a number *N* follows the procedure outlined in Column 1 of Table 1.

Test using $a^{p-1} \equiv 1(mod\ p)$	Test using $\frac{(N-1)}{x} = k$
- Choose a random base a. - Find the remainder when a^{N-1} is divided by *N*. - If the remainder is not 1, *N* is not prime. (END OF TEST) - If the remainder is 1, there is a large probability that *N* is prime. In this case, choose another random base a and repeat the test until a remainder unequal to 1 is found. - The more random a-values having been tested, and a remainder of 1 having always been found, the greater the probability that *N* is indeed prime.	- Choose a base a. - Find the number of digits *x* in the recurring string ***related to the base a***. - Calculate $\frac{(N-1)}{x} = k$ - If *k* is not a whole number, *N* not prime. (END OF TEST) - If *k* is a whole number, and *k* = 1 then *N* is prime. - If *k* is a whole number and $k \neq 1$ (*) then apply one further test (involving *x*) to identify a factor of a potential pseudoprime. - If a factor is found, *N* is not prime. If a factor ($\leq \sqrt{N}$) is not found, *N* is prime.

Table 1

(*) It appears that, within the range of values thus far investigated, when $k < 8$ (for base 10) and when $k < 9$ (for base 2) then *N* is always prime – see discussion later.

The Fermat Prime test using $a^{p-1} \equiv 1(\text{mod}\ p)$ has Several Drawbacks:

1. It is not deterministic: an *N*-value can only be identified as having a high probability of being prime.
2. For very large numbers, many calculation steps are required (in a division process) to eventually yield a remainder of 1. This may take a long time and can use up a large amount of computing power.
3. There are some pseudoprimes (called Carmichael numbers) which *ALWAYS* yield remainders of 1, no matter what base is chosen. Such compound numbers are thus always incorrectly identified as primes.
4. Even if a number *N* is indeed prime, many bases may have to be investigated before its primality can be confirmed.
5. The test does not suggest a method to eliminate a pseudoprime - other than by eventually finding a remainder unequal to 1 - once the appropriate base (if it exists) happens to be chosen.

The Fermat Prime test using $k = \frac{N-1}{x}$ and Vedic Mathematics

This paper discusses the results of an investigation using Vedic Mathematics, in conjunction with the procedure outlined in Column 2 of Table 1, to help determine the primality of a number N. The Ekadikena Purvena sutra is employed to determine the number of recurring digits, *x,* in the *binary* string for $\frac{1}{N}$.

This value can be used to determine the k-value where $k = \frac{N-1}{x}$.

If k = 1, N is identified as prime. If k is a whole number greater than 1 (* see note previous page) an additional test, also using x, is carried out. This test can be used to successfully sift out pseudoprimes.

The method outlined in this paper (although it still has a probabilistic component, as will soon be explained) has been used to identify all prime numbers below 500 000. It is suggested that the Fermat Primality test done in this way, addresses some of the drawbacks listed in the previous section for the conventional Fermat test: Use of the algorithm stated in the sutra may reduce computing time and power, while the x-value (which is found by application of the sutra) is used to identify the factors of all pseudoprimes, even the Carmichael numbers.

The Ekadhikena Purvena Sutra

Sri Tirthaji[3] showed that the full recurring decimal string for every number N, ending on a 1, 3, 7 or 9, can easily be found by using the Ekadhikena Purvena division technique.

The sutra states: "By one more than the previous one".

The generation of the decimal string for $\frac{1}{19}$ is demonstrated below:

The denominator consists of the two digits 1 and 9. Defining the "previous one" as the digit before the 9, i.e. the 1 in the case of 19: "One more than the previous one" is: 1 + 1 = 2. This 2 (called the "ekadhika") is now the new divisor (from left to right), or also the new multiplier (from right to left).

For string generation from left to right, instead of attempting to divide 19 into 1 (according to the conventional method), the procedure is now to simply divide ***2*** into 1 instead, i.e.

2 divided into 1 equals 0 remainder 1. For this, write:

$$^{1}0$$

with the rem 1 a superscript to the left of the quotient 0. Then divide 2 into 10, giving 5 rem 0:

$$^{1}0\ ^{0}5$$

Then divide 2 into 5, giving 2 rem 1:

$$^{1}0\ ^{0}5\ ^{1}2$$

Division of 2 into 12 then yields 6 rem 0:

$$^{1}0\ ^{0}5\ ^{1}2\ ^{0}6$$

Division of 2 into 6 yields 3 rem 0:

$$^{1}0\ ^{0}5\ ^{1}2\ ^{0}6\ ^{0}3$$

Division of 2 into 3 yields 1 rem 1:

$$^{1}0\ ^{0}5\ ^{1}2\ ^{0}6\ ^{0}3\ ^{1}1$$

Division of 2 into 11 yields 5 rem 1:

$$^{1}0\ ^{0}5\ ^{1}2\ ^{0}6\ ^{0}3\ ^{1}1\ ^{1}5$$

Proceeding thus, the 18 digits of the recurring string

$^{1}0\ ^{0}5\ ^{1}2\ ^{0}6\ ^{0}3\ ^{1}1\ ^{1}5\ ^{1}7\ ^{1}8\ ^{0}9\ ^{1}4\ 7\ ^{1}3\ ^{1}6\ ^{0}8\ ^{0}4\ ^{0}2\ ^{0}1$are generated.

Alternatively, the digits in the decimal string can also be generated from right to left in the following way:

Starting with 1 on the very right, multiply the 1 by 2 to obtain 2; then multiply this product by 2 again to obtain 4; then multiply 4 by 2 to obtain 8; then multiply 8 by 2 to get 16,

i.e. $^{1}6\ 8\ 4\ 2\ 1$

where the ten′s digit of the 16 is written as a superscript 1, ready to be added (“carried over”) onto the product of the next multiplication by 2. The next step is to multiply only the 6 by 2 to get 12, after which the superscript 1 (from the ten′s digit of 16) is added onto 12 to get 13,

i.e. $^{1}3\ 6\ 8\ 4\ 2\ 1$

Now multiply only the 3 by 2, then add 1 to get 7. Multiply 7 by 2 to get 14, and write the ten′s digit of the 14 as a superscript 1:

$$^{1}4\ 7\ ^{1}3\ ^{1}6\ 8\ 4\ 2\ 1$$

Proceeding thus, the complete cyclic string is obtained:

$$^{1}0\ 5\ ^{1}2\ 6\ 3\ ^{1}1\ ^{1}5\ ^{1}7\ ^{1}8\ 9\ ^{1}4\ 7\ ^{1}3\ ^{1}6\ 8\ 4\ 2\ 1$$

The number of steps in the calculation can, furthermore, be halved by noting that the string of digits comprising the first half of the decimal expansion, added to the string of digits making up the second half, yields a sequence of nines, i.e.

$$\begin{array}{r} 052631578\ + \\ \underline{947368421} \\ \underline{999999999} \end{array}$$

This phenomenon is an application of the *Nikhilam Navatascaramam Dasatah* sutra:

“All from 9 and the last from 10” because, when the digits in the first half of the string are subtracted from 9, the digits in the second half of the string are obtained. “The last from 10” never features, as there is no last digit in a non-terminating string.

Explanation of the Working of the Ekadhikena Purvena Sutra

The recurring decimal string associated with a fraction is a geometric series of the numbers generated by dividing or multiplying successive terms by a common ratio related to the "ekadhika".

In general, any perfectly recurring decimal for 1/N can be written as:

$$\frac{1}{N} = \sum_{n=1}^{\infty} \left(\frac{1}{N+1}\right)^{n} \qquad \text{The ekadhika is then } \frac{(N+1)}{10}.$$

The Ekadhikena Purvena Sutra applied to $\frac{1}{N}$ where *N* has a Final Digit 1, 3 or 7

When the sutra (used on numbers to base 10) refers to the "previous one" it must always be "previous to" the digit 10 − 1 = 9 in the denominator of a fraction. So that the sutra can be applied to decimal rational numbers (in form $\frac{a}{b}$) with denominators ending also on the digits 1, 3 and 7, such fractions can be manipulated as follows to have a last digit equal to 9:

$$\frac{1}{21} = \frac{1}{21} \times \frac{9}{9} = \frac{9}{189} \qquad \frac{1}{13} = \frac{1}{13} \times \frac{3}{3} = \frac{3}{39} \qquad \frac{1}{7} = \frac{1}{7} \times \frac{7}{7} = \frac{7}{49}$$

For instance: $\frac{1}{13} = \frac{1}{13} \times \frac{3}{3} = \frac{3}{39}$

For 39, "one more than the previous one" is 3 + 1 = 4. The ekadhika is thus 4. For *right to left* string generation, start with the numerator 3 as the last digit before recurrence, and then multiply successively with 4, thereby obtaining:

$$^{3}0\ ^{2}7\ ^{3}6\ 9\ ^{1}2\ 3$$

Thus $\frac{1}{13} = 0.\dot{0}7692\dot{3}$

The process repeats until a remainder of 3 is once again reached. Because 3 is the very first multiplicand, any further steps in the process yield the same sequence of digits again.

Note: Employing $\frac{1}{N} = \sum_{n=1}^{\infty} \left(\frac{1}{N+1}\right)^{n}$: $\frac{3}{39} = 3 \times \frac{1}{39} = 3 \times \sum_{n=1}^{\infty} \left(\frac{1}{40}\right)^{n}$

The ekadhika = 40/10 = 4

The ekadhika is seen here to be a multiple of 10 (as the decimal system employs a base 10).

Computer Program (to Generate *Decimal* Strings) and the Method of Rooting Out Pseudoprimes

The results of employing a computer program, using the Ekadhikena algorithm, to generate the decimal strings for all N-values (ending on 1, 3, 7 or 9 - this includes all primes excluding 2 and 5) between 3 and 10 000 have already been reported in a previous article[4]. The lengths x of the strings before recurrence were found, which then enabled the testing for

primality (testing whether $k = \frac{(N-1)}{x}$ is a whole number or not). The method of rooting out the pseudoprimes (with reasons) was also discussed in detail in the above-mentioned article. It was reported how all primes below 10 000 were successfully identified, and all pseudoprimes were easily eliminated using a method which is again briefly outlined (without proof) below:

If N happens to be a pseudoprime (base a) it satisfies the criterion that,

$$k = \frac{N-1}{x}$$

is a whole number. Thus also $N = (kx + 1)$. It can be shown that:

If N is a pseudoprime: It has factors in the form $(dx + 1)$ where $d < k$

Therefore, once N is found to have a whole number k-value greater than 1, it is tested for a factor $(dx + 1) \leq \sqrt{N}$. If no such factor is found, then N is prime.

Since the writing of the previous article, additional analysis has successfully revealed all the 5133 primes below 50 000, as well as all the pseudoprimes (base 10) below 50 000. The list of all the 64 Fermat pseudoprimes (base 10) below 50 000 is given in Table 2.

Note that the smallest k-value is 8. The second smallest is 15.

The 64 Pseudoprimes (base 10) below N = 50 000
(1.23% of the 5 197 numbers below 50 000 with whole number k-values)

N	x	k	N	x	k	N	x	k	N	x	k
9	1	8	3333	4	833	11111	5	2222	21931	30	731
33	2	16	3367	6	561	11169	16	698	23661	14	1690
91	6	15	4141	20	207	11649	32	364	24013	174	138
99	2	49	4187	13	322	12403	78	159	24661	30	822
259	6	43	4521	8	565	12801	400	32	27613	52	531
451	10	45	5461	42	130	13833	364	38	29341	60	489
481	6	80	6533	46	142	13981	30	466	34113	328	104
561	16	35	6541	30	218	14701	60	245	34133	1484	23
657	16	35	6601	330	20	14817	32	463	34441	60	574
703	18	39	7107	374	19	14911	30	497	35113	24	1463
909	4	227	7471	30	249	15211	390	39	38503	138	279
1233	8	154	7777	12	648	15841	120	132	41041	30	1368
1729	18	96	8149	28	291	19201	30	640	45527	26	1751
2409	8	301	8401	15	560	19503	98	199	46497	32	1453
2821	30	94	8911	198	45	20961	16	1310	46657	96	486
2981	10	298	10001	8	1250	21153	32	661	48433	48	1009

Table 2

The Generation of *Binary* Strings Using the Ekadhikena Purvena Sutra

The use of decimal strings for prime number identification was found not to be ideal. Using the computer program on a home PC to obtain the decimal strings of the reciprocals of (relatively small) primes close to 1 000 000, proved eventually to take up far too much time.

For example, while the prime 4007 (with 4006 recurring digits and thus $k = 1$) took but 0.47 seconds, and the prime 8017 (also with $k = 1$) took but 1.81 seconds to identify, the prime 80051 (also with $k = 1$) took 4 minutes 13 seconds to identify. The prime 999 983 (also with $k = 1$) took about 3.2 hours to identify. However, a prime with a much larger k-value, and thus less recurring digits x, took far less time to identify: for instance, 43 037 (with only 29 recurring digits and thus with $k = 1484$) took but 0.03 seconds to identify.

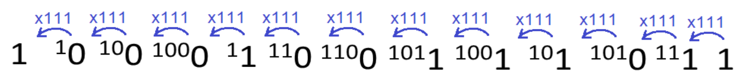
Besides attempts to improve the computer programming methods and efficiency, as well as using a faster and more efficient computer, it was decided to investigate whether application of the Ekadhikena Purvena Sutra on binary (instead of decimal) numbers might improve the speed of prime number identification. An example of how this can be done is given below.

Example 9

13 in binary is 1101. To find the binary string of $\frac{1}{1101}$ proceed as follows:

By one more than the one before "110": 110 + 1 = 111 The ekadhika is thus 111.

Starting with 1, and using 111 as the multiplier *from right to left*, and carrying and adding successive binary digits, the following binary string is obtained:



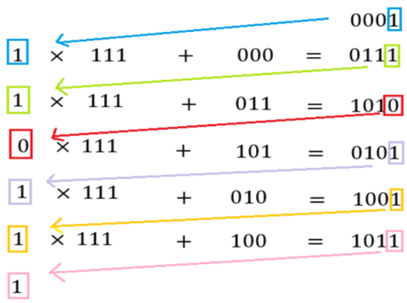
Thus: $$\frac{1}{1101} = 0.\dot{0}00100\,11101\dot{1}$$

There are $x = 12$ binary digits in one recurring cycle of this string.

Using $\frac{(N-1)}{x} = k$ we find that $\frac{13-1}{12} = 1$.

With $k = 1$, $N = 13$ is correctly identified as prime.

The algorithm of the sutra, employed to binary $\frac{1}{13}$, thus proceeds as follows:



etc.

Figure 1

Comparison of *k*-values in Binary and in Decimal

For the case of $\frac{1}{13}$:

In decimal: $\frac{1}{13} = 0.\dot{0}7692\dot{3}$ $\quad x = 6$ and $k = \frac{(N-1)}{x} = \frac{(13-1)}{6} = 2$

In binary: $\frac{1}{1101} = 0.\dot{0}00100\ 11101\dot{1}$ $\quad x = 12$ and $k = \frac{(N-1)}{x} = \frac{(13-1)}{12} = 1$

So, although the *k*-values differ, they are whole numbers in both bases. Furthermore:

For $\frac{1}{13}$:

$$\begin{array}{r} 076+ \\ \underline{924} \\ 999 \end{array}$$

while for $\frac{1}{1101}$:

$$\begin{array}{r} 000100+ \\ \underline{111011} \\ 111111 \end{array}$$

$\frac{1}{13}$ in decimal displays the working of the sutra because the first half of one cycle of therecurring string adds to the second half to yield a string of 9′s.

But $\frac{1}{13}$ in *binary* displays an adapted version of this sutra, i.e. *All from 1 and the last from 2*, because the two halves of its cyclic string yield a string of 1′s when added.

This last phenomenon is found to occur for all even-numbered strings related to prime numbers (proof given in a previous article[4]), but it only occurs occasionally for a pseudoprimes. This phenomenon is briefly touched upon later.

Using binary numbers, Fermat′s prime test is carried out in base 2. The Ekadhikena Purvena sutra, applied to binary numbers, uses: *By one more than the one before the 1*, instead of *By one more than the one before the 9*, as is the case for decimal numbers.

This addition effectively makes the right-to-left multiplier (or left-to-right divisor) - the Ekadhika - a multiple of 2 (instead of a multiple of 10, as is the case when the Ekadhika is used to generate a decimal string).

A binary string is an infinite geometric series based on powers of 2. The following example displays the working of the "by one more than the one before" concept employed by the Ekadhikena sutra applied to $\frac{1}{7}$ in binary. It shows how the recurring string for $\frac{1}{7}$ consists of an infinite number of successive powers of $\frac{1}{8} = \frac{1}{2^3}$ ($\equiv 0.001$ in binary) added together. It also clearly demonstrates why there can be only three digits before recurrence in the binary string for $\frac{1}{7}$ (thus making $k = 2$).

Example 10

$$\boxed{\frac{1}{N} = \sum_{i=1}^{\infty} \left(\frac{1}{N+1}\right)^{i}}$$

$$\frac{1}{7} \equiv \frac{1}{111} = \sum_{i=1}^{\infty} \left(\frac{1}{111+1}\right)^{i} = \sum_{i=1}^{\infty} \left(\frac{1}{\boxed{1000}}\right)^{i}$$

terminating
binary string

$$\frac{1}{8} \equiv \frac{1}{1000} = 0{,}001$$

the ekadhika

$$\sum_{i=1}^{\infty} \left(\frac{1}{1000}\right)^{i} = (0{,}001)^{1} + (0{,}001)^{2} + (0{,}001)^{3} + (0{,}001)^{4} + \cdots \infty$$

×100 ×100 ×100 ×100

$$\frac{1}{7} \equiv \frac{1}{111} = 0{,}001\;001\;001\;001\;001 \ldots \infty$$

When right-to-left string generation is done in binary, the very last digit before recurrence (i.e. the first digit to be multiplied by the Ekadhika) is always a 1. Because only 1′s and 0′s are employed, the algorithm is found to generate recurring strings much faster than is the case for decimal strings.

Computer Program to Generate Binary Strings for Numbers Below 500 000

A computer program was written which employs the Ekadhikena sutra adapted for binary numbers. All numbers ending in 1, 3, 7 or 9 below 500 000 (in decimal) were converted into binary, and the lengths of their binary strings were determined by the program. A time comparison was done between the rate of prime number identification in binary versus in decimal. An example of the identification of a prime number close to 500 000 follows.

Example 11

Test $N = 481\,433$ (prime)

Convert into binary: $\frac{1}{481433} \equiv \frac{1}{1110101100010011001}$

Calculate the ekadhika: 1110101100010011001100+1 = 1110101100010011101

Proceed with right to left multiplication of 1 with the ekadhika, to obtain the full cyclic string shown on the next two pages.

Number of recurring digits: $x = 8597$ $\quad k = \frac{481433-1}{8597} = 56$

Time for calculation of x: 0.11 seconds.

Time for confirmation that not pseudoprime (test for a factor $(dx+1)$ where $d < k$):~ 0.4

$$\frac{1}{481433} \equiv \frac{1}{1110101100010011001}$$

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1
0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1
1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 0 1 0
0 1 1 1 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 1
1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0
0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0
0 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0
1 0 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 1
0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 1
0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 1 0 1 1 1 0 1
1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1
1 1 1 1 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1
0 1 0 0 0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1
1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0
1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 1
0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 0 1 0 1 0
1 0 1 1 1 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 1 1 0 1
1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1
1 1 1 1 0 1 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 0 1
1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0
1 1 0 0 1 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0
0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 1 1
1 1 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0
0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1 1
0 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 0
0 0 1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0
0 1 0 1 1 1 1 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 0 0 1
0 1 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 1 1 0 0 1 1 1 0
1 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1
0 1 0 1 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 1
1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 1 0 1 0 1 1 1
1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0
0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0
1 0 0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 0 0 0 1
1 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1
1 1 1 1 1 0 0 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 0
1 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0
1 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1
1 1 1 0 0 0 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 1 1 1 1 0 0 1 1 1 0 0 0 0
0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0
1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 0 1 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1
1 0 0 0 1 1 1 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0
0 1 0 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1
0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0
1 0 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 1 0 1 0 1 0
0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1
1 0 0 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1
0 0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 1 0
1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 1
1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0
0 0 1 1 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1
0 1 1 1 1 0 1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1
0 0 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 0 1 0
0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0
1 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1
1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0
1 1 1 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 1 1 1 1
1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 0 0 1 0
1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1
0 0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 1 0 1 0 1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1
1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 0 1 0 0
1 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 1 0 1 0
0 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 1 0
1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1
0 1 1 0 0 0 0 0 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0
0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1
0 0 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1
1 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 1
1 1 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1
1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1
1 1 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1
0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 1 1 0 1
0 0 1 1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0
0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1
0 1 1 0 1 0 1 0 0 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1 1 1
0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 0 1 1 0 1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0
0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 1
1 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0
0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1
0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 0 0 0 0 1 1 0 1 1 1 0 1 1 0 0 1
1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 1 0 0 1
0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1 1 1 1 0 1 1 1
0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1
1 0 0 1 0 0 0 0 1 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0
1 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 0 1 0
1 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0
1 0 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 0
0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 1 0 0 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 0 1 1
1 1 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 1
1 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1
1 0 1 0 0 0 1 1 0 0 1 0 1 1 0 1 0 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0
1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0
0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0
1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0 1 1
1 1 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 1
1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0
1 1 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0
0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 1 1 0 0 1
1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 1 0 1 1 1 1 1 0 0 0
0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 0
0 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1
0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0
0 1 0 0 1 1 1 1 1 1 0 0 0 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 1
0 1 1 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1

```
0 1 1 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0
1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0
0 0 0 0 1 1 1 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 1 1 0
0 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 1 0 1
1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1
1 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 1 1 1 0 1 1 1
1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1
0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 0
1 0 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0
0 1 1 0 0 1 1 1 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0
1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1
1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1
0 1 1 1 0 1 1 0 1 0 1 0 1 1 0 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0
1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 0 0 0 0 1 1 0 1 1 1 0 0 1 0 1
0 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 0 1 0 1 1 1 1 0 1 0 1 0 0
0 1 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0
1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1
0 1 1 1 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 1
1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1
0 1 1 0 1 1 1 0 0 1 1 1 0 0 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1
0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 1 1
0 1 0 0 1 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0
1 1 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0
1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 0 1 0 1
1 1 1 1 0 1 0 0 0 1 0 1 0 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 1 1 1 0 0
1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 1 1
1 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 1 1 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1
0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0
0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0
0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 0 1 1 0 0 1 0 1 1 1
0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 0 1 1 1 1 1
1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 1 1 0 1
0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 1 1
0 0 1 0 0 0 1 1 1 1 1 0 1 0 1 0 1 0 1 1 0 0 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 0 1 0
1 0 1 1 0 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 1 1 0 0 0 1 1 1 1
1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 0 1
1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 0 0 0 1 1 1 0
1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0
0 1 0 0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1
1 0 0 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0
0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1
1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 1 1 0 1 0 1
0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 1 0 0
0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 0 1 0
0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0
1 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 0 1 0 1 0 0 0 1
1 1 1 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1
1 1 0 1 0 1 1 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1 0
0 0 0 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 1 1 0 1 1 1 0 1 0 0 0 1 0 1 0 0 0 0
1 0 0 0 1 0 1 0 1 1 1
```

Results of Testing for Primality, and the Time Period to Generate Binary Strings

Between 3 and 500 000: 41 681 numbers were found to have whole number k-values.

Of these, 144 (0.35%) were identified as base 2 pseudoprimes (Poulet numbers) by finding that they were divisible by a factor $(dx + 1)$ with $d < k$. Thus 41 681 − 144 = 41537 primes were found. Including 2, all the 41 538 PRIMES below 500 000 were thus successfully identified.

The time for identification using binary strings, is much faster than when using decimal strings (refer to Tables 3 and 4).

The identification time depends on the x-value, i.e. the number of binary digits occurring in one cycle.

Thus N–values with k = 1 take the longest time to confirm their prime status: the longest string below 500 000 (i.e. the string for 499 979) took about 5 seconds to generate. Such strings do not need to be subjected to a further pseudoprime test.

The strings for numbers with larger k-values (thus with relatively shorter cyclic strings) were generated much quicker. For instance, the string for 498961, with k = 10, was found within 0.56 seconds.

Although all numbers (ending on the digits 1, 3, 7 and 9) below 500 000 were tested, several larger numbers were tested at random: While the string for 1 900 043 (with k = 1) was found in 23 seconds, the string for 1 900 009 (with k = 72) was generated in 0.4 seconds. The generation of the string for a binary number near 10 000 000 (with k = 100) took only about 1 second.

Table 4 shows a comparison of the maximum time (i.e. with k = 1) it takes for approximately equal length strings to be generated in both base 10 and base 2. The difference between the generation time for decimal and binary strings was found to increase substantially as the numbers grow larger.

Table 3

k = 1				k = 2				k = 3				k = 10			
N	x	time(s)	x/time	N	x	time(s)	x/time	N	x	time(s)	x/time	N	x	time(s)	x/time
5003	5002	0.015	333467	5009	2504	0.015	333867	5101	1700	0.000	750000	4871	487	0.000	1800000
10037	10036	0.079	127038	10007	5003	0.051	196197	9973	3324	0.016	623250	9431	943	0.000	1500000
20029	20028	0.156	128385	19991	9995	0.088	227160	20011	6670	0.047	425745	19471	1947	0.015	1298000
30011	30010	0.188	159628	29983	14991	0.116	258466	30133	10044	0.078	386308	31391	3139	0.031	1012581
39989	39988	0.323	123802	40031	20015	0.172	232733	39883	13294	0.109	365890	41201	4120	0.031	1329032
50021	50020	0.36	138944	50033	25016	0.188	266128	49957	16652	0.141	354298	51071	5107	0.032	1595938
60029	60028	0.516	116333	60017	30008	0.234	256479	60013	20004	0.172	348907	60041	6004	0.047	1277447
70003	70002	0.625	112003	70001	35000	0.328	213415	70099	23366	0.188	372862	70991	7099	0.063	1126825
80021	80020	0.667	119970	80039	40019	0.328	244019	79693	26564	0.234	340564	80149	8015	0.078	1027538
90011	90010	0.812	110850	90007	45003	0.371	242604	89923	29974	0.25	359688	90071	9007	0.078	1154744
100003	100002	0.855	116961	100057	50028	0.453	220874	100069	33356	0.312	320731	100591	10059	0.094	1070106
120077	120076	1.033	116240	120047	60023	0.547	219463	120067	40022	0.328	366055	120041	12004	0.14	857429
140069	140068	1.266	110638	139999	69999	0.625	223997	140053	46684	0.406	344956	139991	13999	0.141	992837
160019	160018	1.391	115038	160159	80079	0.781	205068	160243	53414	0.5	320484	160031	16003	0.156	1025833
180043	180042	1.641	109715	180023	90011	0.862	208842	180811	60270	0.562	321726	180361	18036	0.187	964492
200003	200002	1.876	106611	200023	100011	0.886	225759	200293	66764	0.625	320467	200591	20059	0.187	1072674
220013	220012	2	110006	219983	109991	1.047	210107	220141	73380	0.687	320437	219871	21987	0.204	1077794
240011	240010	2.251	106624	239999	119999	1.094	219377	239893	79964	0.75	319856	238591	23859	0.261	914138
260003	260002	2.433	106865	260023	130011	1.187	219058	260011	86670	0.782	332494	260441	26044	0.312	834744
280013	280012	2.594	107946	280031	140015	1.406	199168	280069	93356	0.938	298580	279761	27976	0.297	941953
300043	300042	2.906	103249	300007	150003	1.469	204225	300109	100036	0.969	309709	300151	30015	0.313	958946
350003	350002	3.359	104198	350033	175016	1.578	221820	349813	116604	1.187	294703	350561	35056	0.375	934827
399989	399988	3.548	112736	400031	200015	1.985	201526	400051	133350	1.308	305849	400471	40047	0.407	983956
450101	450100	4.329	103973	450103	225051	2.031	221616	450019	150006	1.371	328241	450071	45007	0.438	1027557
499979	499978	5.079	98440	499943	249971	2.406	207790	499957	166652	1.621	308424	498961	49896	0.558	894194

Table 3
Some *k*-values for primes in range 5000 to 500 000

Decimal String with $k = 1$	Generation time (s)	Binary String with $k = 1$	Generation time (s)
20 047	11	20029	0.2
90 011	300	90 011	0.8
499 976	960	499 976	5.1
999 983	11 400	1 000 003	8.7
-	-	near ten million	117

Table 4

The Rooting out of the Pseudoprimes

With its whole number *k*-value status established, each of the 144 pseudoprimes below 500 000 was identified as such in an additional time of less than 0.001 seconds. This was done by testing all numbers with $k > 1$ for a factor $(dx + 1) < \sqrt{N}$. See **Table 5** for the results.

Some Important Observations with Regards to the Pseudoprimes and their Testing:

1. In decimal (for all numbers tested below 50 000) no pseudoprime revealed a *k*-value less than 8. In binary (for all numbers tested below 500 000) no pseudoprime was found to have a *k*-value less than 9. The calculation time for the recurring string of a pseudoprime is thus generally far less than for a real prime of approximately equal size. The reason for large pseudoprime *k*-values (thus relatively shorter recurring strings) lies in the fact that a pseudoprime must have at least two factors, both of which themselves, can be written in terms of *x*. An attempt at illustrating this is given below:

Say *N* has two unequal prime factors, i.e. $N = (d_1x + 1)(d_2x + 1)$

then $kx + 1 = (d_1x + 1)(d_2x + 1)$

which yields $k = d_1d_2x + d_1 + d_2$ if $x \neq 0$

If d_1 and d_2 are integers, their smallest possible values are 1 and 2 respectively.

Assuming $x \geq 2$ then yields: $k \geq (1)(2)(2) + 1 + 2$

Thus $k \geq 7$

However, many *d*-values are not integers greater than 1 - many have fractional values lying between 0 and 1. In such cases, the *x*-value must be big enough to still yield a relatively large *k*-value. The above illustration, therefore, does not cover all eventualities.

Furthermore, the smallest factor must have the property: $(d_1x + 1) < \sqrt{N}$

Thus $(d_1x + 1)^2 < kx + 1$

and $k > (d_1)^2x + 2d_1$ if $x \neq 0$

Substitution of $d_1 = 1$ then yields $k > x + 2$

This suggests that a pseudoprime with two factors having d_1 and d_2 greater or equal to 1, implies $x < k$. The data in Table 5 is in accord with this observation. Also, d_1 and d_2 are integers. The only three *N*-values (with two factors) in Table 5 with non-integer d_1 and d_2-values less than 1, have $x > k$. These numbers are 4371, 8911 and 25761.

Table 5 shows that pseudoprimes with three and four factors were also found. Except for one number (294409, with $x = 36$; factors 37, 73 and 109; and respective *d*-values 1, 2 and 3) all such pseudoprimes have one or more non-integer *d*-values. In fact (in the case of there being more than two factors), very few integer *d*-values occur, but when they do, it is invariably for cases where $x < k$.

If it can be conclusively established why $k \geq 9$ for base 2 pseudoprimes, the prime test can be adapted to immediately confirm the primality of any *N*-value with $k \leq 8$. Within the range investigated, about 88% of all whole number *k*-values lie below 9. If this trend extrapolates to larger numbers, many more primes may thus be identified without the need for any further testing.

2. Due to the necessity of testing for non-integer d-values during the pseudoprime test, the overall method still has a probabilistic component: If no factor has yet been found by the time one tests, say, for for d = 999999/1000000, what limit must be set to stop the test and confirm that N is indeed prime? (Or, a limit having been set and no factor found within that limit, what is the probability than N is prime?) The limit placed on both the numerator and the denominator of d in the computer program employed in the current investigation, was about 2000. Within the range of numbers investigated, no pseudoprime "slipped through unidentified" using this criterion.

3. Figure 2 shows the percentage occurrence of whole number *k*-values within the range investigated. For the 41 681 numbers below 500 000 with whole number *k*-values:

88% have $1 \leq k \leq 8$ (no pseudoprimes in this region).

11.2% have $9 \leq k \leq 200$ (28 pseudoprimes found)

0.8% have $k > 200$ (116 pseudoprimes found)

Also, 37.3 % have *k* = 1, thus confirming them prime.

Looked at in a different way: (See Figures 3 and 4).

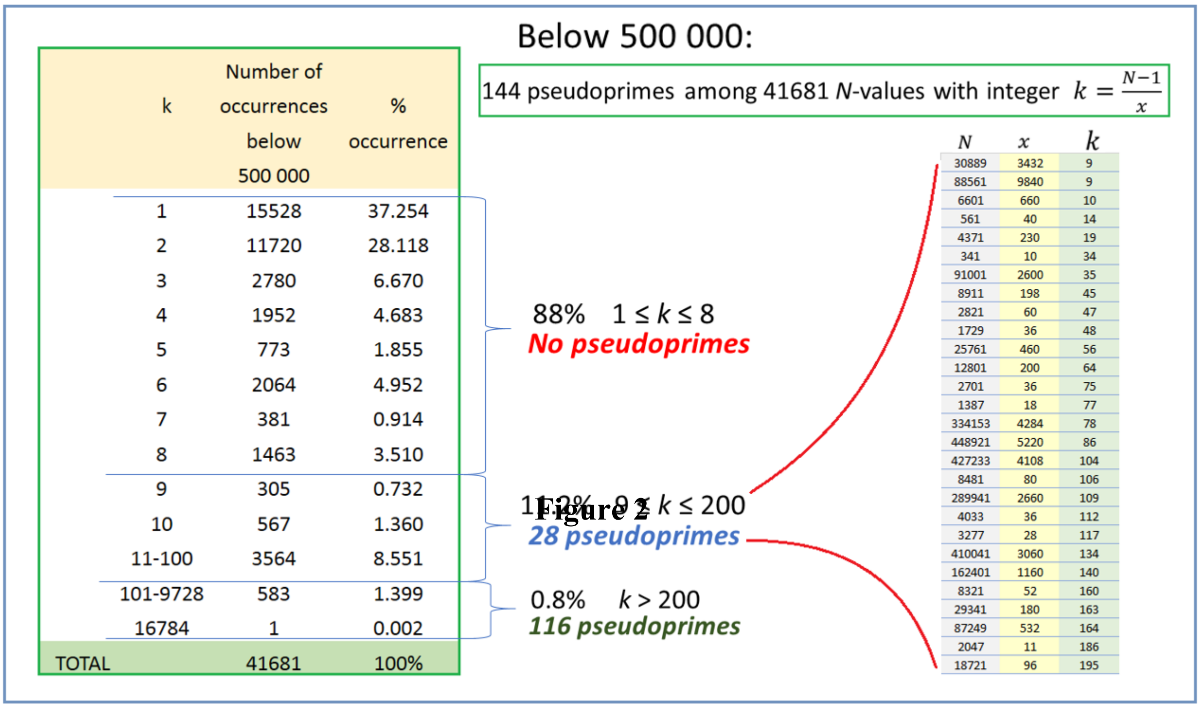
For binary numbers, only 0.35% (144 out of 41681) of all the whole number *k*-values below 500 000 are pseudoprimes.

Only 28 of these (~ 0.07%) are found amongst the 41373 numbers with *k*-values less than 221.

116 pseudoprimes are amongst the 308 numbers (~38% of them) which have the biggest *k*-values (with *k*'s between 221 and 16 789).

With reference to the last point, one might thus reasonably conclude that, if a tested binary number has a *k*-value greater than about 220, there is an approximate 40% chance that the number is a pseudoprime

Figure 4 illustrates the density of occurrence of base 2 pseudoprimes with $k \geq 221$.



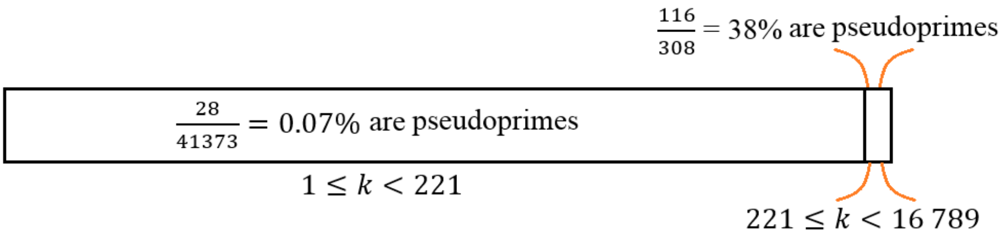
Below 500 000:

144 pseudoprimes among 41681 N-values with integer $k = \frac{N-1}{x}$

k	Number of occurrences below 500 000	% occurrence
1	15528	37.254
2	11720	28.118
3	2780	6.670
4	1952	4.683
5	773	1.855
6	2064	4.952
7	381	0.914
8	1463	3.510
9	305	0.732
10	567	1.360
11-100	3564	8.551
101-9728	583	1.399
16784	1	0.002
TOTAL	41681	100%

N	x	k
30889	3432	9
88561	9840	9
6601	660	10
561	40	14
4371	230	19
341	10	34
91001	2600	35
8911	198	45
2821	60	47
1729	36	48
25761	460	56
12801	200	64
2701	36	75
1387	18	77
334153	4284	78
448921	5220	86
427233	4108	104
8481	80	106
289941	2660	109
4033	36	112
3277	28	117
410041	3060	134
162401	1160	140
8321	52	160
29341	180	163
87249	532	164
2047	11	186
18721	96	195

Figure 3



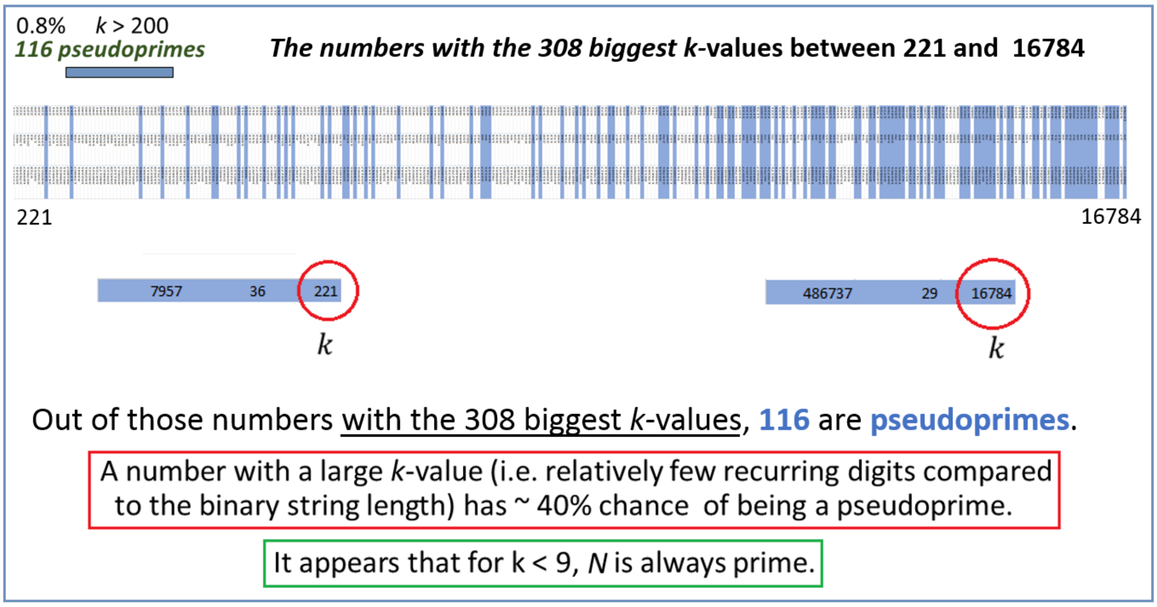


Figure 4

4. Using the $(dx + 1) < \sqrt{N}$ pseudoprime test, while all factors of pseudoprimes were found in less than 0.001 seconds, all actual prime numbers (with *k*-values greater than 1) took much longer – between about 0.3 to 0.6 seconds - to be checked for factors in form $(dx + 1)$. The 0.6 second maximum time was, of course, determined by the limit set on the *d*-value. When no such factor was found, the test was terminated. It was observed that: Just by ordering numbers in terms of the run-time for the pseudoprime test, all pseudoprimes were immediately sifted out. A possible criterion for pseudoprime identification could thus be: Within a certain range of numbers, if the run-time for the pseudoprime test is greater than a pre-determined cut-off time, one can almost be certain that *N* is prime.

5. All prime numbers N with even-numbered strings have the property that the first half of the binary string for 1/*N* adds to the second half to give a string of 1′s (the adapted Nikilam sutra: "All from 1 and the last from 2"). There are 134 pseudoprimes (out of the total 144 pseudoprimes below 500 000) with even-numbered strings. Only 22 of them concur with the adapted Nikilam sutra. Thus, the overwhelming majority of pseudoprimes (84%) do not possess this property. This difference between a pseudoprime and a prime might be usefully employed as an additional factor in a primality test procedure.

6. A study of the data in Table 5 shows that: For a pseudoprime *N*, the number of recurring digits in 1/*N* (i.e. *x*) is the lowest common multiple (LCM) of the number of recurring digits (say, x_1,x_2, etc.) in each of the cyclic binary strings related to the respective prime factors (say $(d_1x + 1)$, $(d_2x + 1)$ etc.) of *N*. This accords with the fact that each factor must itself be one plus a multiple (or submultiple) of *x*. The next example serves to illustrate this observation.

Example 12

For *N* = 90 751 with *x* = 75 and *k* = 1210 and factors 151 and 601 (both prime), the respective values for d_1 and d_2 are 2 and 8.

$$90751 = kx + 1 = (1210)(75) + 1$$

Also $d_1x + 1 = (2)(75) + 1 = 151$ and $d_2x + 1 = (8)(75) + 1 = 601$

The prime factors 151 and 601, themselves, have respective x-values (i.e. number of recurring digits in the strings for $\frac{1}{151}$ and $\frac{1}{601}$) of $x_1 = 15$ and $x_2 = 25$.

Thus $k_1x_1 + 1 = (10)(15) + 1 = 151$ and $k_2x_2 + 1 = (24)(25) + 1 = 601$

Both 15 and 25 are divisors of 75. Thus x_1 and x_2 are divisors of *x*. Thus, *x* is the LCM of x_1 and x_2.

More on k-values and the Ekadhika

To paraphrase J. Pickles (2000)[5]:

> *It is evident that a cycle of length (N – 1) is the longest that can be achieved with the divisor N. The successive steps of division by N can never be exact, or the decimal*

would terminate, and there are only (N – 1) possible remainders. Once these are exhausted, the sequence must repeat.

The maximum number of digits must be $(N-1)$. Why do some numbers have the property that the sequence of numbers in a cycle is only half this maximum limit (i.e. $k = 2$) or one third of this maximum limit (i.e. $k = 3$) etc.?

The answer to this question lies in what has already been pointed out in Example 10, where 1/7 in binary must necessarily only have 3 binary digits before recurrence. (It is the same reason why $\frac{1}{999}$ in decimal must also have only three decimal digits in one recurring cycle.)

Hardy and Wright[2] address the answer to this question in Theorem 88 of the book *The Theory of Numbers*.

Theorem 88 of Hardy and Wright:

$a^x \equiv 1\ (mod\, N)$ *has a smallest solution x which is a divisor of Φ(N)*

where Φ(N) is equal to the number of integers smaller than N which are relatively prime (co-prime).

Stated in another way: If $\frac{a^x}{N}$ is found and a remainder equal to 1 is obtained (i.e. recurrence occurs), then the smallest possible value for x is a divisor of $\Phi(N)$. Thus $\Phi(N)$ divided by x must have an integer value, i.e. $\frac{\Phi(N)}{x} = k$. Since ***all*** integers smaller than a prime number are relatively prime to it, $\Phi(N) = N - 1$ for a prime. Therefore, for a prime number: $\frac{N-1}{x} = k$. It also follows that, if $x = N - 1$, (thus $k = 1$) N is always prime.

For a compound number, since there are some numbers below it which are not relatively prime, it follows that $(N-1) = \Phi(N) + n_p$, where n_p represents the number of integers smaller than N which are not co-primes. In the case of a pseudoprime, it just so happens that n_p is also divisible by N, which is usually not the case for a non-prime. This has been illustrated in a previous article[4].

In a brief analysis, Jeremy Pickles[5] uses several examples to show that there is a relationship between the ekadhika and the k-value: The ekadhika is either a perfect power of k, or it is a polynomial residue of the divisor N. The following examples illustrate this phenomenon:

Example 13

In base10: $\frac{1}{163} = \frac{1}{163} \times \frac{3}{3} = \frac{3}{489}$ Use of the sutra shows that $x = 81$, thus $k = 2$.

Here the ekadhika is $48 + 1 = 49 = 7^2$. We thus see that the ekadhika is a power of $k = 2$.

Example 14

In base10: $\frac{1}{127} = \frac{1}{127} \times \frac{7}{7} = \frac{7}{889}$ Use of the sutra shows that $x = 42$, thus $k = 3$.

Here the ekadhika is $88 + 1 = 89 = (6)^3 - 1(127)$

We can also write: $\frac{89}{127} = \frac{6^3}{127} - 1$

We see thus that the ekadhika is the cubic (i.e. power $k = 3$) residue of the divisor 127.

Example 15

In base 2: for $\frac{1}{113} \equiv \frac{1}{1110001}$: Use of the sutra shows that $x = 28$, thus $k = 4$.

Here the ekadhika is $111000 + 1 = 111001 \equiv 57 = (12)^4 - 183(113)$

We can also write: $\frac{57}{113} = \frac{12^4}{113} - 183$

We see thus that the ekadhika is the quartic (i.e. power $k = 4$) residue of the divisor 113.

In general, the relationship between the ekadhika and the k-value (and thus also the number of digits before recurrence) is thus:

$$\text{ekadhika} = y^k - qN$$

where both y and q are integers, and q is the quotient when y^k is divided by N.

We can also write: $y^k \equiv \text{ekadhika} (\text{mod } N)$

It is of interest to compare this to Fermat's Little Theorem: $a^{p-1} \equiv 1 (\text{mod } p)$

The k-values for one number also differ depending on the base used, as shown in Figure 4.

For $\frac{1}{7}$ in decimal: $k = 1$ ekadhika $= 5 = (12)^1 - 1(7)$

For $\frac{1}{7}$ in binary: $k = 2$ ekadhika $= 4 = (2)^2$

$N = 7 \equiv$ *Decimal:*

$\frac{1}{7}$ = 0.142857 ....
$\frac{2}{7}$ = 0. 285714 ....
$\frac{3}{7}$ = 0.428571....
$\frac{4}{7}$ = 0. 571428....
$\frac{5}{7}$ = 0.714285 ....
$\frac{6}{7}$ = 0. 857142 ....
$x = 6$
$k = 1$

$N = 111 \equiv$ *Binary:*

$\frac{1}{111}$ = 0.001 ....
$\frac{10}{111}$ = 0.010 ....
$\frac{11}{111}$ = 0.110....
$\frac{100}{111}$ = 0. 100....
$\frac{101}{111}$ = 0.101 ....
$\frac{110}{111}$ = 0. 110 ....
$x = 3$
$k = 2$

Figure 4

Summary and Suggestions for Further Investigation

1. Compared to decimal string generation, a computer program (employing the Ekadhikena Purvena sutra) can generate the number of of recurring digits in the binary string for $\frac{1}{N}$ at a much higher rate. The difference in the generation rate of binary versus decimal strings increases with N.

2. Binary numbers up to about 10 million have been investigated. The maximum time (if $k = 1$) to generate the recurring string for a number close to 10^7 is approximately 2 minutes. Although this is several orders of magnitude faster than when the sutra is applied to decimal numbers, the current method would still take an impossibly long time to identify primes with several hundred digits (i.e. those used in cryptology).

3. The programming for this investigation was done using TrueBasic. The algorithm is currently being rewritten in C++. By using more efficient data structures, calculation times may be reduced significantly.

4. Employment of hogher-performance computing hardware and more efficient implementation of the algorithms should enable far larger prime numbers to be identified using the method outlined in this paper. The home PC that was used in the current investigation has the following characteristics:

Intel Core i5-7200U 2,5 GHz with Turbo Boost up to 3.1 GHz

4 GB DDR4 Memory; 1000 GB HDD

The new C++ program will be run using a faster processor and larger RAM.

5. All prime numbers N (and several pseudoprimes) with even-numbered strings have the property that the first half of the binary string for $1/N$ adds to the second half to give a string of $1'$s. It is planned to incorporate this property (propounded by the Nikilam sutra) into the improved C++ program – thus possibly reducing by up to a half the time required to generate recurring strings.

6. Once the improvements suggested in points 3, 4 and 5, have been implemented, it is planned that the rate at which the improved program generates binary strings, be compared with the rates at which other existing methods achieve the same result. This has not yet been done.

7. If it can be conclusively established why $k \geq 9$ for base 2 pseudoprimes, the prime test can be adapted to immediately confirm the primality of any N-value with $k \leq 8$. (See Figure 5)

8. Employment of the Ekadhikena Purvena sutra to find the number of recurring digits x in a binary string, serves a dual purpose: Besides being able to prove primality when $x = 1$ (or perhaps even when ≤ 8), the x-value can also be used in the search for factors of pseudoprimes in the form $(dx + 1)$. However, due to the necessity of testing for non-integer d-values during the pseudoprime test, the overall method still has a probabilistic component. Further investigation needs to be done to find out if a limit can be set on d, below which the test can be terminated, and N confirmed, unquestionably, to be prime. Alternatively, a limit to d having been set, and no factor found within that limit, can one set a value to the probability that N is prime?

9. Do the Carmichael numbers have any particular pattern in their k-values which make them always pass the conventional Fermat Primality test, no matter what base is chosen? This is a topic for further investigation.

10. Another topic for investigation is the relationship between the formulas:

$y^k \equiv \text{ekadhika}(\text{mod } N)$ and $a^{p-1} \equiv 1(\text{mod } p)$ discussed in the previous section.

11. It is proposed that, within a large range of data, a Fourier analysis be done on calculated k-values to see whether any pattern in their occurrence emerges.

References

1. Fermat's Little Theorem: first stated in a letter dated October 18, 1640. Euler provided the first published proof in 1736.

2. Theorem 88 in "The Theory of Numbers", G.H. Hardy and E.M. Wright (1938)

3. Vedic Mathematics, Bharati Krishna Tirtha (1965)

4. An Investigation into the working of the Ekadhikena Purvena Sutra, and how it can be used to identify Prime Numbers, M. Fletcher (Proceedings of 16th World Sanskrit Conference, Bangkok (2015)

5. Jeremy Pickles, IAVM website (June 2000)

Appendix

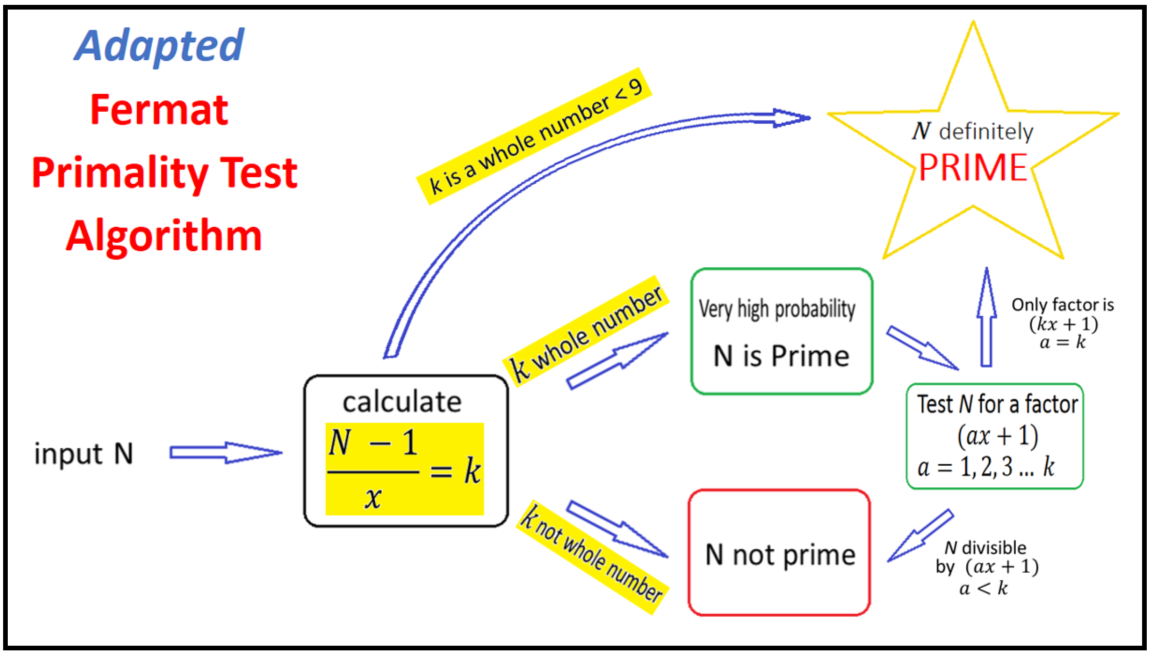


Figure 5
A proposed, adapted, Fermat Primality test
(if it can conclusively be shown that no non-prime exists with k ≤ 8)

<u>Table 5</u>

The 144 Pseudoprimes base 2 below 500 000

Time to find x (sec)	N	x	k		Factors				a - values			Time to find factor a
0.011	341	10	34	11	31			1	3			0
0.011	561	40	14	11	17	3		1/4	2/5	1/20		0
0.011	1387	18	77	19	73			1	4			0
0.012	1729	36	48	19	13	7		1/2	1/3	1/6		0
0.018	2047	11	186	23	89			2	8			0
0.012	2701	36	75	37	73			1	2			0
0.013	2821	60	47	31	13	7		1/2	1/5	1/10		0
0.014	3277	28	117	29	113			1	4			0
0.012	4033	36	112	37	109			1	3			0
0.014	4369	16	273	17	257			1	16			0
0.024	4371	230	19	93	47			2/5	1/5			0
0.014	4681	15	312	31	151			2	10			0
0.013	5461	14	390	43	127			3	9			0
0.034	6601	660	10	23	41	7		1/30	2/33	1/110		0
0.014	7957	36	221	73	109			2	3			0
0.015	8321	52	160	53	157			1	3			0
0.016	8481	80	106	11	3	257		1/8	1/40	16/5		0
0.02	8911	198	45	67	133			1/3	2/3			0
0.017	10261	30	342	31	331			1	11			0
0.026	12801	200	64	17	3	251		2/25	1/100	5/4		0
0.02	13741	60	229	13	7	151		1/5	1/10	5/2		0
0.017	13747	58	237	59	233			1	4			0
0.018	13981	20	699	41	11	31		2	1/2	3/2		0
0.018	14491	42	345	43	337			1	8			0
0.018	15709	22	714	23	683			1	31			0
0.019	15841	45	352	31	7	73		2/3	2/15	8/5		0
0.021	18721	96	195	97	193			1	2			0
0.022	19951	70	285	71	281			1	4			0
0.017	23001	40	575	41	11	17	3	1	1/4	2/5	1/20	0
0.018	23377	48	487	97	241			2	5			0
0.032	25761	460	56	93	277			1/5	3/5			0
0.023	29341	180	163	61	37	13		1/3	1/5	1/15		0
0.022	30121	60	502	13	7	331		1/5	1/10	11/5		0
0.144	30889	3432	9	79	23	17		1/44	1/156	2/429		0
0.02	31417	88	357	89	353			1	4			0
0.021	31609	72	439	73	433			1	6			0

<u>Table 5</u> (continued/…)

The 144 Pseudoprimes base 2 below 500 000

Time to find x (sec)	N	x	k	Factors				a - values				Time to find factor a
0.021	31621	102	310	103	307			1	3			0
0.022	33153	112	296	43	3	257		3/8	1/56	15/7		0
0.019	35333	44	803	89	397			2	9			0
0.021	41041	60	684	41	13	11	7	2/3	1/5	1/6	7/60	0
0.027	42799	21	2038	127	337			6	16			0
0.025	46657	144	324	97	37	13		2/3	1/4	1/12		0
0.026	49141	156	315	157	313			1	2			0
0.021	49981	30	1666	151	331			5	11			0
0.027	52633	153	344	103	73	7		2/3	8/17	2/51		0
0.03	57421	180	319	13	7	631		1/15	1/30	7/2		0
0.026	60701	100	607	101	601			1	6			0
0.024	60787	22	2763	89	683			4	31			0
0.024	63973	36	1777	37	19	13	7	1	1/2	1/3	1/6	0
0.025	65077	58	1122	59	1103			1	19			0
0.027	65281	48	1360	97	673			2	14			0
0.025	68101	60	1135	41	11	151		2/3	1/6	5/2		0
0.029	75361	120	628	31	13	11	17	1/4	1/10	1/12	2/15	0
0.034	80581	60	1343	61	1321			1	22			0
0.057	83333	166	502	167	499			1	3			0
0.033	85489	52	1644	53	1613			1	31			0
0.057	87249	532	164	229	127	3		3/7	9/38	1/266		0
0.035	88357	148	597	149	593			1	4			0
0.498	88561	9840	9	83	97	11		1/120	2/205	1/984		0
0.03	90751	75	1210	151	601			2	8			0
0.14	91001	2600	35	101	53	17		1/26	1/50	2/325		0
0.137	93961	360	261	31	7	433		1/12	1/60	6/5		0
0.051	101101	300	337	101	13	7	11	1/3	1/25	1/50	1/30	0
0.034	104653	76	1377	229	457			3	6			0
0.044	113201	100	1132	41	11	251		2/5	1/10	5/2		0
0.036	115921	72	1610	37	13	241		1/2	1/6	10/3		0
0.036	123251	58	2125	59	2089			1	36			0
0.034	126217	36	3506	73	19	13	7	2	1/2	1/3	1/6	0
0.041	129889	96	1353	193	673			2	7			0
0.036	129921	70	1856	11	31	3	127	1/7	3/7	1/35	9/5	0
0.037	130561	68	1920	137	953			2	14			0
0.118	137149	66	2078	23	67	89		1/3	1	4/3		0

Table 5 (continued/…)

The 144 Pseudoprimes base 2 below 500 000

Time to find x (sec)	N	x	k		Factors				a - values			Time to find factor a
0.04	149281	60	2488	11	41	331		1/6	2/3	11/5		0
0.042	150851	50	3017	251	601			5	12			0
0.06	154101	460	335	3	31	1657		1/230	3/46	18/5		0
0.05	157641	280	563	3	11	17	281	1/140	1/28	2/35	1	0
0.04	158369	28	5656	29	43	127		1	3/2	9/2		0
0.042	162193	48	3379	241	673			5	14			0
0.103	162401	1160	140	17	41	233		2/145	1/29	1/5		0
0.042	164737	64	2574	257	641			4	10			0
0.051	172081	60	2868	7	13	31	61	1/10	1/5	1/2	1	0
0.055	176149	126	1398	19	73	127		1/7	4/7	1		0
0.046	181901	100	1819	101	1801			1	18			0
0.045	188057	44	4274	89	2113			2	48			0
0.049	188461	36	5235	109	19	13	7	3	1/2	1/3	1/6	0
0.051	194221	166	1170	167	1163			1	7			0
0.075	196021	660	297	7	41	683		1/110	2/33	31/30		0
0.052	196093	156	1257	157	1249			1	8			0
0.067	204001	480	425	7	151	193		2/67	3/4	24/25		0
0.056	206601	200	1033	3	17	4051		1/100	2/25	81/4		0
0.058	212421	260	817	3	11	41	157	1/130	1/26	2/13	3/5	0
0.051	215749	78	2766	79	2731			1	35			0
0.062	219781	270	814	271	811			1	3			0
0.051	220729	51	4328	103	2143			2	42			0
0.068	226801	336	675	337	673			1	2			0
0.083	228241	720	317	13	97	181		1/60	2/15	1/4		0
0.054	233017	42	5548	43	5419			1	129			0
0.063	241001	200	1205	401	601			2	3			0
0.061	249841	144	1735	433	577			3	4			0
0.067	252601	300	842	41	61	101		2/15	1/5	1/3		0
0.058	253241	52	4870	157	1613			3	31			0
0.058	256999	29	8862	233	1103			8	38			0
0.064	258511	210	1231	11	71	331		1/21	1/3	11/7		0
0.067	264773	148	1789	149	1777			1	12			0
0.062	271951	75	3626	151	1801			2	24			0
0.067	272251	198	1375	7	19	23	89	1/33	1/11	1/9	4/9	0
0.07	275887	262	1053	263	1049			1	4			0
0.068	276013	198	1394	199	73	19		1	4/11	1/11		0

Table 5 (continued/…)

The 144 Pseudoprimes base 2 below 500 000

Time to find x (sec)	N	x	k		Factors				a - values			Time to find factor a
0.207	280601	92	3050	277	1013			3	11			0
0.078	282133	306	922	307	919			1	3			0
0.084	284581	180	1581	11	41	631		1/18	2/9	7/2		0
0.552	289941	2660	109	3	127	761		1/1330	1/21	2/7		0
1.065	294271	102	2885	103	2857			1	28			0
0.245	294409	36	8178	37	73	109		1	2	3		0
0.016	314821	660	477	13	61	397		3/165	1/11	3/5		0
0.015	318361	120	2653	241	1321			2	11			0
0.016	323713	144	2248	13	37	673		1/12	1/4	14/3		0
0.032	332949	308	1081	3	29	43	89	1/154	1/11	3/22	2/7	0
0.047	334153	4284	78	19	43	409		1/238	1/102	2/21		0
0.015	340561	264	1290	13	17	23	67	1/22	2/33	1/12	1/4	0
0.016	341497	72	4743	13	109	241		1/6	3/2	10/3		0
0.016	348161	340	1024	11	31	1021		1/34	3/34	3		0
0.031	357761	130	2752	131	2731			1	21			0
0.015	367081	120	3059	11	13	17	151	1/12	1/10	2/15	5/4	0
0.032	387731	70	5539	71	43	127		1	3/5	9/5		0
0.015	390937	156	2506	313	1249			2	8			0
0.032	396271	74	5355	223	1777			3	24			0
0.031	399001	420	950	31	61	211		1/14	1/7	1/2		0
0.016	401401	600	669	401	7	11	13	2/3	1/100	1/60	1/50	0
0.079	410041	3060	134	41	73	137		2/153	2/85	2/45		0.015
0.016	422659	162	2609	163	2593			1	16			0
0.016	423793	48	8829	17	97	257		1/3	2	5 1/3		0
0.047	427233	4108	104	3	53	2687		1/2054	1/79	17/26		0
0.015	435671	190	2293	191	2281			1	12			0
0.016	443719	166	2673	167	2657			1	16			0
0.063	448921	5220	86	11	37	1103		1/522	1/145	19/90		0
0.016	452051	50	9041	251	1801			5	36			0
0.031	458989	92	4989	277	1657			3	18			0
0.031	476971	390	1223	11	131	331		1/39	1/3	11/13		0
0.031	481573	84	5733	337	1429			4	17			0
0.032	486737	29	16784	233	2089			8	72			0
0.016	488881	180	2716	181	37	73		1	1/5	2/5		0
0.031	489997	156	3141	157	3121			1	20			0
0.032	493697	112	4408	113	17	257		1	1/7	16/7		0