

An open source software package for primality testing of numbers of the form $p2^n+1$, with no constraints on the relative sizes of p and 2^n

Tejas R. Rao ^{Corresp.}

Corresponding Author: Tejas R. Rao
Email address: tejas_rao@comcast.net

We develop an efficient software package to test for the primality of $p2^n+1$, p prime and $p>2^n$. This aids in the determination of large, non-Sierpinski numbers p , for prime p , and in cryptography. It furthermore uniquely allows for the computation of the smallest n such that $p2^n+1$ is prime when p is large. We compute primes of this form for the first one million primes p and find four primes of the form above 1000 digits. The software may also be used to test whether $p2^n+1$ divides a generalized fermat number base 3.

An open source software package for primality testing of numbers of the form $p2^n + 1$, with no constraints on the relative sizes of p and 2^n

Tejas R. Rao

ABSTRACT

We develop an efficient software package to test for the primality of $p2^n + 1$, p prime and $p > 2^n$. This aids in the determination of large, non-Sierpinski numbers p , for prime p , and in cryptography. It furthermore uniquely allows for the computation of the smallest n such that $p2^n + 1$ is prime when p is large. We compute primes of this form for the first one million primes p and find four primes of the form above 1000 digits. The software may also be used to test whether $p2^n + 1$ divides a generalized fermat number base 3.

INTRODUCTION

Prime numbers are used in a large number of applications. For example, RSA authentication relies on large prime numbers and has numerous uses in cryptography (Ferguson, N. and Schneier, B., 1994). Securing communications between computers and large networks rely on this research. The novel idea presented in the article is explored for feasibility of finding additional large prime numbers to aid in encryption.

Only recently was the first polynomial time algorithm, the AKS, created for primes regardless of form, and it is still highly inefficient (Agrawal, M. and Kayal, N. and Saxena, N., 2004). However, there are many specific forms of prime numbers with their own primality tests (Grantham, J. (2001)). Of these, some of the least computationally complex are the Proth test and the Sophie-Germaine test, for finding primes of the form $k2^n + 1$, k odd and $2^n > k$, and $2p + 1$, p prime, respectively (Matthew, G. and Williams, H. C. (1977), Dubner, H. (1996)). In addition to the efficiency of the tests, they are of special importance to Sierpinski numbers of the second kind.

Sierpinski numbers of the second kind are odd numbers k where $k2^n + 1$ is composite for all n (Baillie, R. and Cormack, G. and Williams, H. C. (1981)). For large k , it is computationally prohibitive to test for the primality of $k2^n + 1$ without utilizing either Proth's test or the Sophie-Germaine test. However, these tests restrict the numbers able to be tested to $k2^n + 1$ where $n = 1$ or where $2^n > k$ (Matthew, G. and Williams, H. C. (1977), Dubner, H. (1996)). Thus, there is a unique need for an efficient algorithm for finding the primality of $k2^n + 1$ for $n > 1$ and $2^n < k$. In this paper, we introduce such an algorithm for numbers of the form $p2^n + 1$, p prime. Throughout this paper, p will refer to primes.

METHODS

The method section will be divided into two parts: the theory and the software package.

Theory

A necessary concept is that of *multiplicative order*. The multiplicative order of a modulo n is defined as the first positive integer m such that

$$a^m \equiv 1 \pmod{n}.$$

Alternatively, one may think of multiplicative order as the first m such that $a^m - 1$ is divisible by n . For a prime p , we take the definition of Legendre's symbol,

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}. \quad (1)$$

Also written as $(a|p)$, its possible values are contained in the set $\{-1, 0, 1\}$. Additionally, we utilize Gauss's quadratic reciprocity,

$$\left(\frac{a}{p}\right) = (-1)^{\frac{p-1}{2} \frac{a-1}{2}} \left(\frac{p}{a}\right), \quad (2)$$

provided both a and p are prime (Lemmermeyer, F. (2000)). Proth's Theorem is well known: where $P = k2^n + 1$, k odd, and $2^n > k$,

$$a^{\frac{P-1}{2}} \equiv -1 \pmod{P} \iff P \text{ prime},$$

for all a where $(a|p) = -1$. The theory section extends a similar criterion to that of Proth's theorem for all numbers of the form $p2^n + 1$, $n > 1$. Also, we utilize the definition of primover numbers found in (Shevelev, V. (2012)). Primover numbers in a base are numbers whose factors share the *same multiplicative order modulo that base*. Recognize that primover numbers are a type of probable prime (may be composite or prime). We furthermore denote $GF(a, z) = a^{2^z} + 1$, the generalized Fermat numbers of base a . Finally we take the following two lemmas from (Shevelev, V. (2012)) and (Lemmermeyer, F. (2000)), respectively.

Lemma 1. All factors of $GF(a, z)$ are primover in base a with 2^{z+1} as the multiplicative order of a modulo every factor.

Lemma 2. Where r is the multiplicative order of a modulo N , $r|z$ if and only if $N|a^z - 1$.

Let $R = p2^n + 1$, $p > 3$ prime and $n > 1$.

Proposition 1. For all such R ,

$$3^{\frac{R-1}{2}} \equiv -1 \pmod{R} \iff R \text{ is prime or } R \text{ divides } GF(3, n-1) \text{ and is primover.}$$

For necessity, first assume R is prime. We can also assume $3 \nmid R$, because then the equivalence above would not hold. Since $p > 3$, we thus have that $k \equiv \pm 1 \pmod{3}$ and $2^n \equiv \pm 1 \pmod{3}$. We cannot have $k \not\equiv 2^n \pmod{3}$, because then $3|R$. Therefore,

$$\begin{aligned} R &\equiv 1 + 1 \\ &\equiv -1 - 1 \\ &\equiv -1 \pmod{3}. \end{aligned}$$

This means that $(R|3) = -1$, by Equation 1. Additionally, we have $R \equiv 1 \pmod{4}$ since $n > 1$. This means we can write $R = 4m + 1$, for $m \in \mathbb{N}$. Using Equation 2,

$$\begin{aligned} \left(\frac{3}{R}\right) &= (-1)^{\frac{3-1}{2} \frac{R-1}{2}} \left(\frac{R}{3}\right) \\ &= (-1)^{(1) \frac{4m}{2}} (-1) \\ &= (-1)^{2m+1} \\ &= -1. \end{aligned}$$

Since we assume R is prime, the necessity for prime numbers follows from Equation 1. For the second part of the necessity, if R is composite but divides $GF(3, n-1)$, the order of 3 modulo f is 2^n , for all factors f of R , from Lemma 1.

For sufficiency, we assume R is composite and does not divide $GF(3, n-1)$. Due to the conditions, we have $3^{R-1} \equiv 1 \pmod R$. Therefore, the multiplicative order of $3 \pmod R$ divides $R-1 = p2^n$ but does not divide $p2^{n-1}$ by Lemma 2 and the conditions specified in the theorem. The multiplicative order is thus precisely 2^n . Furthermore, $3^{\frac{R-1}{2}} \equiv -1 \pmod R$ implies that all factors of R divide $3^{\frac{R-1}{2}} + 1$. This means that no factors divide $3^{\frac{R-1}{2}} - 1$. But since the order of 3 modulo R is 2^n , all factors of R must therefore share this order. Therefore all of the factors f of R have multiplicative order of 3 modulo f as 2^n . Therefore we can write

$$3^{2^n} - 1 = (3^{2^{n-1}} - 1)(3^{2^{n-1}} + 1) \equiv 0 \pmod R.$$

and deduce that $3^{2^{n-1}} + 1 \equiv 0 \pmod R$ by Lemma 2. We arrive at a contradiction: a composite solution must divide $GF(3, n-1)$. Since all divisors of Fermat numbers are primover, we prove the conditions.

Remark 1. If the conditions of Proth's theorem are satisfied and/or if $p > 1/2(3^{2^n} + 1)$, we know R is prime iff it satisfies the above condition.

Remark 2. We can alternatively check that the number R is not a Fermat factor of $GF(3, n-1)$ to prove that R is prime after it passes the initial test.

Code

First, it is necessary to understand and utilize the BigInteger Java class, which is a class of immutable, arbitrary-precision integers (Oracle (n. d.)). In addition to handling arbitrarily large integers, the class also contains multiple optimization techniques for modular multiplication, such as the Schönhage–Strassen algorithm.

Next, we must utilize repeated modular exponentiation to find the large power of 3, for any number with over a thousand digits will overload most systems as an exponent. By repeatedly squaring and reducing modulo R at each step, one can calculate $3^{2^n} \pmod R$:

```
public static BigInteger result (BigInteger base, int n, BigInteger R) {
    for (int i = 0; i < n; i++) {
        base = base.pow(2).mod(R);
    }
    return base;
}
```

It is a well known result that every integer can be written as the sum of powers of two. To write a number in binary, simply decompose it as the sum of powers of 2 as follows. The code returns an array of the powers of 2 that make up R from least to greatest. For example, if $R = 11$, then the method will return $[0, 1, 3]$, because $11 = 2^3 + 2^1 + 2^0$.

```
private static Integer [] findpowers (BigInteger R) {
    int c = 0;
    ArrayList<Integer> list = new ArrayList<Integer>();
    String r = R.toString(2);
    for (int i = 0; i < r.length(); i++) {
        if (r.charAt(i) == '1') {
            list.add(r.length()-(i+1));
        }
    }
    arrlis = list.toArray(new Integer[list.size()]);
    Integer[] arrlisFlipped = new Integer[arrlis.length];
    for (int i = arrlis.length - 1; i >= 0; i-- ) {
        arrlisFlipped[c++] = arrlis[i];
    }
    return arrlisFlipped;
}
```

106 This methodology becomes useful in discerning $3^{\frac{R-1}{2}} \bmod R$, which is what we must compute. We
 107 find the binary form of $\frac{R-1}{2}$ and then calculate and multiply together 3 to the resulting powers of 2. For
 108 example, we can simplify as follows:

$$109 \quad 3^{11} \bmod R = (3^{2^3} \bmod R) * (3^{2^1} \bmod R) * (3^{2^0} \bmod R) \bmod R.$$

110 Therefore, after finding the array of powers of two, we implement modular exponentiation through
 111 repeated squaring to find $3^{\frac{R-1}{2}} \bmod R$ as desired.

112 For sufficiently large R , this process becomes prohibitively slow. When searching for large primes of the
 113 form $p2^n + 1$, many values of n and p must be tested. It is therefore necessary to quickly rule out multiple
 114 values of n for any given p . To do this, we implement trial division by the first two million primes for each
 115 $p2^n + 1$ before implementing the main test, as many composites are found and ruled out by this method.

116 The two expensive operations are repeated squaring and the modular multiplication of 3 to the powers of
 117 two. To account for this, multithreading is utilized ([Intel \(n. d.\)](#)). In the process of calculating $3^{2^a} \bmod R$,
 118 where a is the highest power of two returned in *findpowers*($\frac{R-1}{2}$), we save all of the $3^{2^z} \bmod R$ that have
 119 z as a value returned from the *findpowers* method. Because of multi-threading, we can concurrently
 120 calculate $3^{2^a} \bmod R$ and multiply each $3^{2^z} \bmod R$ we save together. Since there will always be a greater
 121 than or equal amount of modular multiplications in calculating $3^{2^a} \bmod R$ relative to multiplying each
 122 $3^{2^z} \bmod R$ together, the whole process will complete in the time it takes to do the one repeated modular
 123 squaring. In the most extreme situation, when calculating $3^M \bmod R$, where M is a Mersenne number
 124 (one less than a power of two), there are $a + 1$ powers of two in the binary representation of M and thus a
 125 modular multiplications of 3 to the powers of 2 returned by the *findpowers* method, the same amount of
 126 multiplications as calculating $3^{2^a} \bmod R$ via repeated modular squaring. In this case, if multithreading
 127 was not used, the time of completion would be roughly doubled. Regardless, in all cases, multithreading
 128 saves a significant amount of time that positively correlates with the number of powers of two in the
 129 binary representation of the input.

130 We implement similar methodology to check Remark 2 for all numbers that pass the original test. It is left
 131 up to the user to determine whether Remark 1 is satisfied, if they choose to do so.

132 RESULTS

133 The theory and methodology culminated in the package referenced below. Its code repository can be
 134 found in [Table 2](#).

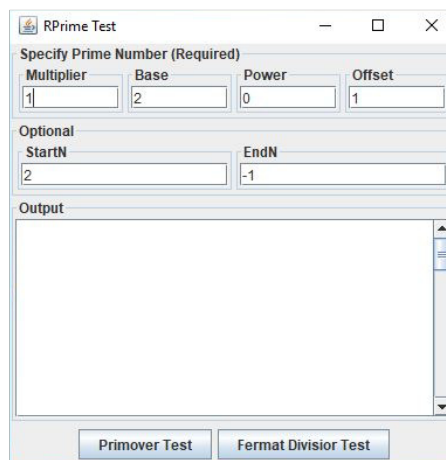


Figure 1. User interface for primality testing software package.

135 The first button supplies the primary test: those that pass the test are either prime or primover and divide a
 136 specific base 3 generalized Fermat number. To test for numbers of the form $p2^n + 1$, one must first specify

the original prime p . Since these values are often too large for the integer class, they must be specified with an integer multiplier, base, pow, and offset in the following format:

$$multiplier * base^{pow} + offset.$$

This is readily accomplished for most of the largest known primes, such as Proth primes and Mersenne primes.

The next row allows for specification of the range of n one would like to test. If values are left unspecified, then the test will range from 2 and go on to the maximum integer value Java can store. Additionally, one can specify whether the program should stop after finding the first primover number when clicking the primover test button. After running, the values of n in the specified range that make $p2^n + 1$ primover will be returned, as well as the number of digits in each primover $p2^n + 1$.

The second button specifies whether $p2^n + 1$ is guaranteed prime or primover and divides $GF(3, n - 1)$, given that the number passed the primary test. One inputs the original prime p as before, as well as a value of n returned from the primary test. The software will return *true* if it divides $GF(3, n - 1)$ and *false* if it does not and is therefore guaranteed prime provided it passes the primary test.

In a preliminary test of the first 1,000,000 primes p , 696,281 of the first 1,000,000 primes had some $p2^n + 1$, $2^n < p$ that was prime. The other 303,719 are possible Sierpinski numbers of the second kind. Some of them, such as 10223, are proven not to be. Furthermore, all of these primover numbers were shown not to divide $GF(3, n - 1)$, meaning they are all prime by Remark 2 and not Fermat Divisors base 3. The list of these primes is provided in the supplemental information section.

Additionally, in the first week of use, a preliminary result was achieved: four primes above one thousand digits were found. Note that, once again, all primes that passed the primary test were guaranteed prime by the second.

Original Prime p	Exponent n	# of Digits
$(305136484659)2^{11471} + 2^{72} + 1$	5659	5169
$(305136484659)2^{11399} + 1$	72	3465
$(699549860111847)2^{4244} + 11$	2745	2119
$(699549860111847)2^{4244} + 7$	1030	1603

Table 1. Some primes of the form $p2^n + 1$, $2^n < p$.

158

Software Specifications

The summary of the main characteristics, availability, and requirements is given in the following table.

	RPrimes Package
Language	Java
Operating system	Platform independent; requires Java distribution
Dependencies	NONE
Software location	DOI: 10.5281/zenodo.1560703
Code repository	https://github.com/tejasrao42/RPrimes
License	MIT

Table 2. Software characteristics, requirements and availability for RPrimes package.

DISCUSSION

Due to the similarity of this test to Proth's test, the computational complexity of this test is $\tilde{O}(\log^2(N))$. More importantly, only one modular exponentiation is required. This efficiency makes it as efficient as

164 Proth's theorem for testing for the largest primes.

165 This inclusion bridges the gap between Sophie-Germain and Proth primality tests. Utilizing all three tests,
166 one can now determine the first n for which $p2^n + 1$ is prime for previously prohibitively large primes p ,
167 which can greatly improve sequences such as (Sloane, N. J. A. (n. d.)). Additionally, if no n are found
168 that make $p2^n + 1$ prime, one has found a large prime p that is under suspicion of being a Sierpinski
169 number of the second kind. Usually, small numbers such as 10223 were computationally thought to be
170 Sierpinski numbers of the second kind partly because for all $2^n < p$, the numbers were small enough
171 to check for primality using trial division, and no primes were found. Recall that for numbers $2^n > p$,
172 Proth's test applies. For large p , one cannot use trial division to check primality when $2^n < p$, so our test
173 is of unique significance in those circumstances. Combined with the aforementioned tests, all $p2^n + 1$
174 may be checked for primality, and large primes p can now be computationally put under suspicion of
175 being Sierpinski numbers of the second kind.

176 Furthermore, the first million primes p have no $p2^n + 1$ that pass the primary test but factor a Generalized
177 Fermat number base 3. This additionally held true for the four prime numbers found above 1000 digits.
178 The result hints at the overall efficiency of the primary test: it is likely passed predominantly by primes
179 numbers and only rarely by composite numbers. For each n , there can only be a small, finite amount
180 of $p2^n + 1$ that pass the primary test, but not the second, because they all must factor $GF(3, n - 1)$.
181 Furthermore, these numbers (if they exist) may be identified through online resources through finding a
182 factorization of $GF(3, n - 1)$. For these n , the primary test becomes not only a test for primover numbers,
183 but moreover a primality test independent of having to compute the second test for all p where $p2^n + 1$ is
184 not found in the factorization of $GF(3, n - 1)$.

185 ACKNOWLEDGMENTS

186 I would like to thank the editor and the reviewers for valuable comments contributing to the overall
187 improvement of the manuscript.

188 REFERENCES

- 189 Agrawal, M. and Kayal, N. and Saxena, N. 2004. [Primes Is in P](#). *Annals of Mathematics* **160**:781-793
190 Baillie, R. and Cormack, G. and Williams, H. C. 1981. [The Problem of Sierpinski Concerning](#). *Math.*
191 *Comp.* **37**:229-231
192 Dubner, H. 1996. [Large Sophie Germain primes](#). *Math. Comp.* **65**:393-396
193 Ferguson, N. and Schneier, B. 1994. [Practical Cryptography](#). New York, Dover Publications
194 LCCN:64013458
195 Grantham, J. 2001. [Frobenius pseudoprimes](#). *Math. Comp.* **70**:873-891
196 Intel. n. d. [Intel Hyper-Threading Technology, Technical User's Guide](#). Intel
197 Lemmermeyer, F. 2000. [Reciprocity Laws: from Euler to Eisenstein](#). Springer ISBN-13:9783642086281
198 Matthew, G. and Williams, H. C. 1977. [Some new primes of the form](#). *Math. Comp.* **31**:797-798
199 Oracle. n. d. [An improved BigInteger class which uses efficient algorithms, including Schönhage-Strassen](#).
200 *Oracle*
201 Shevelev, V. et al. 2012. [Overpseudoprimes, and Mersenne and Fermat Numbers as Primover Numbers](#).
202 *Journal of Integer Sequences* **15**
203 Sloane, N. J. A. n. d. [Smallest m such that \$\(2n - 1\)2^m + 1\$ is prime, or \$-1\$ if no such value exists..](#) *The*
204 *On-Line Encyclopedia of Integer Sequences* **A046067**