An Efficient Deterministic Primality Test

Joseph M. Shunia

November 2023

Abstract

A deterministic primality test with a polynomial time complexity of $O(\log(n)^3 \log\log(n))$ is presented. The test posits that an integer n satisfying the conditions of the main theorem is prime. Combining elements of number theory and combinatorics, the proof operates on the basis of simultaneous modular congruences relating to binomial transforms of powers of two.

1 Introduction

Primality testing has seen remarkable advancements over the past few decades. A significant breakthrough in this field was the AKS primality test, introduced by Agrawal, Kayal, and Saxena (2002) [1]. The AKS test was the first to offer determinism and polynomial-time complexity, a monumental achievement that resolved a longstanding open question in computational number theory [2]. However, despite its theoretical importance, the AKS test has practical limitations due to its relatively high polynomial time complexity, rendering it inefficient for most applications. Agrawal, Kayal, and Saxena gave a time complexity of $O(\log(n)^{12})$ for the AKS test [1]. This bound was lowered significantly by Lenstra and Pomerance (2011) to $O(\log^6(n))$ [3]. Despite this reduction, AKS remains impractical and is mostly unused.

In the field of cryptography, the unique properties of prime numbers are widely exploited to create cryptographic primitives. It is often the case that many large primes must be generated in rapid succession [4]. To make these cryptographic operations practical, fast probabilistic primality tests such as the Baille-PSW primality test (BPSW) [5] or Miller-Rabin (MR) [6] [7] are used instead of AKS when searching for large primes. Probabilistic primality tests are by definition non-deterministic and may erroneously report a composite integer as being prime. Composite integers which pass a probabilistic primality test are relatively rare and are known as psuedoprimes (PSPs) for the respective test [8]. When generating primes for cryptographic purposes, probabilistic primality tests are often combined or repeated with different parameters in order to achieve an acceptable error-bound that makes it almost certain that no composite integer will pass. However, reducing the error-bound requires additional compute and increases running-time, creating a trade-off.

We present a new deterministic primality test that operates in polynomial time with a time complexity of $O(\log(n)^3 \log\log(n))$. This efficiency gain opens new avenues for practical applications, particularly in cryptography, where fast and reliable primality testing is desirable [9]. Our main theorem posits a condition for an odd integer n to be prime, based on specific modular congruences related to the binomial transforms of powers of 2. The basis for our test is the following main theorem: Let n be an odd integer satisfying $2^{n-1} \equiv 1 \pmod{n}$. Denote D as the least integer strictly greater than 2 and less than n which does not divide $n-1$. Then, n is prime if and only if a set of simultaneous modular congruences involving D, n, and binomial coefficients hold.

This paper is structured as follows: We begin by presenting the main theorem and its proof, substantiated by two critical lemmas. The first lemma demonstrates the test's validity for odd prime numbers, while the second confirms its failure for odd composite numbers. Through these lemmas, we establish the deterministic

nature of our test. We then describe the algorithm used to compute our test and analyze its computational complexity. We also give a pseudocode implementation for our test to show how it can be implemented.

2 Main Theorem

Theorem 1. Let n be an odd integer $n > 3$ satisfying $2^{n-1} \equiv 1 \pmod{n}$. Denote D as the least integer greater than 2 and less than n which does not divide $n-1$. Then, n is prime if and only if the following congruence holds:

$$1 + 2^{\lfloor \frac{n-1}{D} \rfloor} \equiv \left(1 + 2^{\lfloor \frac{n-1}{D} \rfloor}\right)^n \equiv \sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n} \tag{1}$$

2.1 Supporting Lemmas

2.1.1 Lemma for odd prime n

Lemma 1. If n is an odd prime integer > 3, it passes our test unconditionally.

Proof. To show that an odd prime $n > 3$ passes the test, we need to verify the requisite congruences under the assumption that n is an odd prime.

Note. We require $n > 3$ due to the nature of our test, which defines D as the least integer greater than 2 and less than n which does not divide $n-1$. In the case of $n = 3$, no such D exists.

Step 1: Show that $2^{n-1} \equiv 1 \pmod{n}$.

Let $\varphi(n)$ denote Euler's totient function. Euler's totient theorem [10] states that if n and a are coprime positive integers, then:

$$a^{\varphi(n)} \equiv 1 \pmod{n} \tag{2}$$

In our case, we have n as an odd prime and $a = 2$, which is even. These are obviously coprime. Also, since n is prime, we have $\varphi(n) = n - 1$. Applying Euler's totient theorem, we see that our initial criterion $2^{n-1} \equiv 1 \pmod{n}$ holds, as $2^{\varphi(n)} = 2^{n-1}$ and $2^{\varphi(n)} \equiv 1 \pmod{n}$.

Step 2: Verify that $1 + 2^{\lfloor \frac{n-1}{D} \rfloor} \equiv \left(1 + 2^{\lfloor \frac{n-1}{D} \rfloor}\right)^n \pmod{n}$.

Define $f(x) = 2^{\lfloor \frac{x-1}{D} \rfloor}$. Taking the binomial expansion of $(1 + f(n))^n$, we can see:

$$(1 + f(n))^n = \sum_{k=0}^{n} \binom{n}{k} 1^{n-k} \cdot f(n)^k = \sum_{k=0}^{n} \binom{n}{k} f(n)^k \tag{3}$$

Since n is prime, $\binom{n}{k}$ is divisible by n for all k in $0 < k < n$. This is because $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ has n in the numerator and, as k and $n-k$ are less than our n (which is prime), neither $k!$ nor $(n-k)!$ can have n as a factor. Thus, all terms of the sum, except for $k = 0$ and $k = n$, are divisible by n. Further, as $\binom{n}{0} = \binom{n}{n} = 1$, our congruence simplifies as follows:

$$(1 + f(n))^n \equiv \binom{n}{0} f(n)^0 + \binom{n}{n} f(n)^n \pmod{n} \tag{4}$$

$$\equiv 1 \cdot f(n)^0 + 1 \cdot f(n)^n \pmod{n} \tag{5}$$

$$\equiv 1 + f(n)^n \pmod{n} \tag{6}$$

Fermat's little theorem tells us that when n is prime and a is an integer coprime to n, we have:

$$a^n \equiv a \pmod{n} \tag{7}$$

In our case, n is odd and $f(n)^n$ is an even power of 2, therefore these are always coprime and we have $f(n)^n \equiv f(n) \pmod{n}$. Now, we can use Fermat's little theorem to simplify further:

$$(1 + f(n))^n \equiv 1 + f(n)^n \equiv 1 + f(n) \pmod{n} \tag{8}$$

Step 3: Confirm that $1 + 2^{\lfloor \frac{n-1}{D} \rfloor} \equiv \sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n}$.

Define $f(x) = 2^{\lfloor \frac{x-1}{D} \rfloor}$. By substitution, we have:

$$1 + f(n) \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \pmod{n} \tag{9}$$

In Step 2, we noted how when n is prime, $\binom{n}{k} \equiv 0 \pmod{n}$ for every $0 < k < n$. Therefore, all terms of the sum, except for $k = 0$ and $k = n$, are divisible by n. Further, as $\binom{n}{0} = \binom{n}{n} = 1$ and $f(1) = 2^{\lfloor \frac{1-1}{D} \rfloor} = 2^{\lfloor \frac{0}{D} \rfloor} = 2^0 = 1$, our congruence simplifies as follows:

$$\sum_{k=0}^{n} \binom{n}{k} f(k+1) \pmod{n} \equiv \binom{n}{0} f(1) + \binom{n}{n} f(n+1) \pmod{n} \tag{10}$$

$$\equiv 1 \cdot f(1) + 1 \cdot f(n+1) \pmod{n} \tag{11}$$

$$\equiv f(1) + f(n+1) \pmod{n} \tag{12}$$

$$\equiv 1 + f(n+1) \pmod{n} \tag{13}$$

Recall that $f(x) = 2^{\lfloor \frac{x-1}{D} \rfloor}$. Examine the exponent $\lfloor \frac{x-1}{D} \rfloor$. We have defined D to be an integer greater than 2 which does not divide $n-1$. Consequently, by the nature of the floor operation, we have $\left\lfloor \frac{(n+1)-1}{D} \right\rfloor = \lfloor \frac{n-1}{D} \rfloor$. This implies $f(n+1) = f(n)$, and thus:

$$\sum_{k=0}^{n} \binom{n}{k} f(k+1) \equiv 1 + f(n) \pmod{n} \tag{14}$$

Conclusion:

The congruences share a common term, leading to the test's defining congruence:

$$1 + 2^{\lfloor \frac{n-1}{D} \rfloor} \equiv \left(1 + 2^{\lfloor \frac{n-1}{D} \rfloor}\right)^n \equiv \sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n} \tag{15}$$

These steps establish that for an odd prime n, the initial criteria and all conditions of our test are satisfied. Hence, odd primes > 3 pass our test unconditionally. □

2.1.2 Lemma for odd composite n

Lemma 2. If n is an odd composite integer, it fails our test unconditionally.

Proof. Assume n is an odd composite integer proposed to pass the given primality test. We aim to demonstrate that under these conditions, the necessary congruences of our test cannot hold simultaneously.

Define $f(x) = 2^{\left\lfloor \frac{x-1}{D} \right\rfloor}$ and D as the least integer greater than 2 that does not divide $n-1$. We begin with the congruence:

$$(1 + f(n))^n \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \pmod{n} \tag{16}$$

Using the binomial theorem, we expand the left-hand side:

$$\sum_{k=0}^{n} \binom{n}{k} f(n)^k \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \pmod{n}. \tag{17}$$

Rewriting this, we must have:

$$\sum_{k=0}^{n} \binom{n}{k} \left(f(n)^k - f(k+1) \right) \equiv 0 \pmod{n}. \tag{18}$$

As a necessary condition of our test, we also require:

$$\sum_{k=0}^{n} \binom{n}{k} f(n)^k \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \equiv 1 + f(n) \pmod{n} \tag{19}$$

For this to hold, in both $f(n)^k$ and $f(k+1)$, at indices $k = 0, n$, their evaluations taken modulo n must be equivalent to $1, f(n)$ respectively. In such case, the inner terms sum to a multiple of n. Therefore, it must be true that:

$$\sum_{k=1}^{n-1} \binom{n}{k} \left(f(n)^k - f(k+1) \right) \equiv 0 \pmod{n} \tag{20}$$

For composite n, the binomial coefficient $\binom{n}{k}$ is not divisible by n for k such that $\gcd(n, k) \neq 1$. For these values of k, the congruence $f(n)^k \equiv f(k+1) \pmod{n}$ is unlikely because $f(n)^k$ and $f(k+1)$ represent different powers of 2 modulo n. It would necessarily imply $2^{\left\lfloor \frac{n-1}{D} \right\rfloor k} \equiv 2^{\left\lfloor \frac{k}{D} \right\rfloor} \pmod{n}$. This can only occur at values k in the range $0 < k < n$ which are solutions to the modular congruence:

$$\left\lfloor \frac{n-1}{D} \right\rfloor k \equiv \left\lfloor \frac{k}{D} \right\rfloor \pmod{\operatorname{ord}_n(2)} \tag{21}$$

- **Cycling of $2^{\left\lfloor \frac{n-1}{D} \right\rfloor k}$ Modulo n:** This term cycles through all possible residues of powers of 2 modulo n, as k varies from 1 to $n-1$. This is because $\left\lfloor \frac{n-1}{D} \right\rfloor$ is fixed for a given n, and as k increases, the exponent $\left\lfloor \frac{n-1}{D} \right\rfloor k$ effectively cycles through various powers.
- **Behavior of $2^{\left\lfloor \frac{k}{D} \right\rfloor}$ Modulo n:** In contrast, this term does not cycle through all residues in the same way. The exponent $\left\lfloor \frac{k}{D} \right\rfloor$ changes more slowly with k, as it's the floor of a fraction. Therefore, for multiple consecutive values of k, the value of $2^{\left\lfloor \frac{k}{D} \right\rfloor}$ remains the same.

In the special case where $f(n) \equiv 1 \pmod{n}$, $f(n)^k$ simplifies to $1 \pmod{n}$ for all k. The congruence would require $f(k+1)$ to also simplify to $1 \pmod{n}$ for each k such that $\binom{n}{k} \not\equiv 0 \pmod{n}$. This demands each

$\left\lfloor \frac{k}{D} \right\rfloor$ to be a multiple of the order of 2 modulo n, or $\text{ord}_n(2)$. In this scenario, it is possible for the congruence to hold. However, if D does not divide $\text{ord}_n(2)$, the variability in $\left\lfloor \frac{k}{D} \right\rfloor$ disrupts any potential alignment with multiples of $\text{ord}_n(2)$. This variability would ensure that $f(k+1)$ does not uniformly reduce to 1 $(\text{mod } n)$ across all k.

Recall that we are given an odd composite n with $2^{n-1} \equiv 1 \pmod{n}$. By the properties of the order of an integer modulo composite n, the smallest k such that $2^k \equiv 1 \pmod{n}$, that is $k = \text{ord}_n(2)$, must be a divisor of $n-1$. That is, $\text{ord}_n(2) \mid (n-1)$. Since $\left\lfloor \frac{n-1}{D} \right\rfloor$ is strictly less than $n-1$ and D does not divide $n-1$, it follows that $f(n) \not\equiv 1 \pmod{n}$ and $D \nmid \text{ord}_n(2)$.

Since $f(n) \not\equiv 1 \pmod{n}$, applying the Pigeonhole Principle [11], there must exist at least one value of k such that $\binom{n}{k} \left(f(n)^k - f(k+1)\right) \not\equiv 0 \pmod{n}$. This non-zero term in the sum, when multiplied by the non-zero binomial coefficient (which shares a common factor with n), ensures that the sum cannot be congruent to zero modulo n. This gives a contradiction in the equivalence of the two sums modulo n, demonstrating that the test's conditions cannot be simultaneously satisfied for odd composite integers. Therefore, such integers will fail the test unconditionally. □

2.2 Proof of the Main Theorem

Proof of Theorem 1. The correctness of the theorem follows from Lemma 1 and Lemma 2. The lemmas together cover all possible cases of odd integers n. Therefore, the theorem is proven. □

3 Algorithm

INPUT: An integer $n > 1$.

1. If $n \equiv 0 \mod 2$:
   (a) If n equals 2, output PRIME.
   (b) Otherwise, output COMPOSITE.
2. If n equals 3, output PRIME.
3. If $2^{n-1} \not\equiv 1 \pmod{n}$, output COMPOSITE.
4. Find the least integer D that is greater than 2 and less than n which does not divide $n-1$.
5. Set $A = 2^{\left\lfloor \frac{n-1}{D} \right\rfloor} \bmod n$.
6. Set $B = (1+A)^n \bmod n$.
7. If $B \not\equiv 1 + A \pmod{n}$, output COMPOSITE.
8. Set $C = \sum_{k=0}^{n} \binom{n}{k} 2^{\left\lfloor \frac{k}{D} \right\rfloor} \bmod n$.
9. If $C \not\equiv 1 + A \pmod{n}$, output COMPOSITE.
10. Output PRIME;

3.1 Time Complexity Analysis

3.1.1 Algorithm Overview

The given algorithm is a primality test that involves several computational steps, including modular arithmetic and polynomial exponentiation in the ring $\mathbb{Z}/n\mathbb{Z}$.

3.1.2 Analysis of Individual Operations

1. **Check for Even** n:

   This step involves calculating n mod 2 and has a time complexity of $O(1)$.

2. **Modular Exponentiation** $2^{n-1} \mod n$:

   This step requires modular exponentiation with a $\log(n)$-digit base and a $\log(n)$-digit exponent. The time complexity of modular exponentiation is $O(\log(n)M(n))$.

3. **Finding** D:

   Finding the least integer $D > 2$ that does not divide $n - 1$ takes at most $O(\log(n))$ steps, with each step requiring $O(1)$ time for the mod operation. Hence, the overall complexity is $O(\log(n))$.

4. **Computing** A **and** B:

   Each of these steps involves modular exponentiation similar to Step 2, and thus each has a time complexity of $O(\log(n)M(n))$.

5. **Computing** C:

   Computing C involves exponentiating a polynomial in the ring $\mathbb{Z}/n\mathbb{Z}$ with $O(\log(n))$ terms and summing the coefficients.

   (a) Summing the polynomial coefficients can be done in $O(\log(n)^2)$ time.

   (b) Exponentiation using repeated squaring takes $O(\log(n))$ steps, and each step requires $O(M(n)\log(n))$ time due to the multiplication of polynomials of size $O(\log(n))$.

   Therefore, the overall complexity for computing C is $O(\log(n)^2 M(n))$.

6. **Comparisons**:

   The final steps involve comparisons which are $O(1)$ operations.

3.1.3 Overall Time Complexity

The dominant time complexity in the algorithm comes from computing C. Therefore, the overall time complexity of the algorithm is $T(n) = O(\log(n)^2 M(n))$.

Harvey and van Der Heoven (2021) have given an algorithm for integer multiplication which has a time complexity $M(n) = O(\log(n)\log\log(n))$ [12]. This would give our algorithm an overall time complexity of:

$$T(n) = O(\log(n)^2 M(n)) = O(\log(n)^2 \log(n)\log\log(n)) = O(\log(n)^3 \log\log(n)) \tag{22}$$

3.1.4 Conclusion

The overall complexity is polynomial in the size of n when expressed in terms of bit operations, making the algorithm efficient for large values of n.

3.2 Psuedocode Implementation

To demonstrate how our test may be implemented, we offer a pseudocode implementation of the key functions involved.

3.2.1 IsPrime Function

Require: An integer $n > 1$
  function IsPrime(n)
    if $n \bmod 2 = 0$ **then**
      if $n = 2$ **then**
        return true ▷ n is prime
      else
        return false ▷ n is composite
      end if
    end if
    if $n = 3$ **then**
      return true ▷ n is prime
    end if
    $fermat \leftarrow \text{Pow}(2, n-1, n)$ ▷ Fermat pseudoprime test to base 2
    if $fermat \neq 1$ **then**
      return false ▷ n is composite
    end if
    $D \leftarrow 2$
    $log \leftarrow \text{Log2}(n)$
    for $i \leftarrow 3$ **to** $\text{Max}(log, 3)$ **do**
      $D \leftarrow i$
      $m \leftarrow (n-1) \bmod D$
      if $m \neq 0$ **then**
        break
      end if
    end for
    $A \leftarrow \text{Pow}(2, \lfloor (n-1)/D \rfloor, n)$
    $expectedValue \leftarrow (A+1) \bmod n$
    $B \leftarrow \text{Pow}(A+1, n, n)$
    if $B \neq expectedValue$ **then**
      return false ▷ n is composite
    end if
    $polyDegree \leftarrow D - 1$ ▷ Subtract 1 to account for zero indexing in arrays
    $poly \leftarrow \text{PolyPow}([1,1], n, n, polyDegree, [2])$
    $C \leftarrow \text{PolyEval}(poly, 1) \bmod n$ ▷ Evaluate at x=1 to sum coefficients
    if $C \neq expectedValue$ **then**
      return false ▷ n is composite
    end if
    return true ▷ n is prime
  end function

3.2.2 PolyPow Function

  function PolyPow($poly_A$, k, n, d, $polyMapping$)
    $polyB \leftarrow [1]$ ▷ Initialize polyB as a polynomial with constant term 1
    while $k > 0$ **do**
      if $k \bmod 2 = 1$ **then**
        $polyB \leftarrow \text{PolyMul}(polyA, polyB, n)$ ▷ Multiply polynomials modulo n
        $polyB \leftarrow \text{PolyReduce}(polyB, d, polyMapping, n)$ ▷ Reduce degree of polyB
        if $k = 1$ **then**
          break

```
            end if
        end if
        polyA ← PolyMul(polyA, polyA, n)                    ▷ Square polyA modulo n
        polyA ← PolyReduce(polyA, d, polyMapping, n)        ▷ Reduce degree of polyA
        k ← k/2
    end while
    return polyB
end function
```

3.2.3 PolyReduce Function

```
function PolyReduce(polyA, d, mappingPoly, n)
    if Length(polyA) ≤ d then
        return polyA
    end if
    polyB ← Clone(polyA)                                    ▷ Copy the polynomial to a new array
    polyDegree ← Degree(mappingPoly)                        ▷ Find degree of the mapping polynomial
    degreeDelta ← d − polyDegree
    for i ← Length(polyB) − 1 downto d + 1 do
        if polyB[i] = 0 then
            continue
        end if
        for j ← i − 1 − degreeDelta, k ← polyDegree downto 0 do
            polyB[j] ← polyB[j] + polyB[i] × mappingPoly[k]     ▷ Degree reduction step
        end for
        polyB[i] ← 0
    end for
    polyC ← new array of size d + 1
    for i ← 0 to min(Length(polyC), Length(polyB)) − 1 do     ▷ Take coefficients modulo n
        polyC[i] ← polyB[i]
        if n ≠ 0 then
            polyC[i] ← polyC[i] mod n
        end if
    end for
    return polyC
end function
```

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, pages 781–793, 2002.

[2] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[3] Hendrik W Lenstra and Carl Pomerance. Primality testing with gaussian periods. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1951): 3376–3390, 2011.

[4] Hendrik W Lenstra. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[5] Robert Baillie and Samuel S Jr Wagstaff. Lucas pseudoprimes. *Mathematics of Computation*, 35(152): 1391–1417, 1980.

[6] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1): 128–138, 1980.

[7] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[8] Samuel S Jr Wagstaff. Pseudoprimes and a generalization of artin's conjecture. *Acta Arithmetica*, 41 (2):141–150, 1983.

[9] Carl Pomerance. The use of elliptic curves in cryptography. *Advances in Cryptology*, pages 203–208, 1984.

[10] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 6 edition, 2008.

[11] Kenneth H Rosen. *Discrete Mathematics and its Applications*. McGraw-Hill Education, 7 edition, 2012.

[12] Joris van Der Hoeven David Harvey. Integer multiplication in time o(n log n). *Annals of Mathematics*, 2021. doi: 10.4007/annals.2021.193.2.4.hal-02070778v2.