2006

Polynomial time primality testing algorithm

Takeshi Aoyama

Department of Computer Science
Rochester Institute of Technology
Master's Project Report

Polynomial Time Primality Testing Algorithm

Takeshi Aoyama

May 19, 2003

Committee

Chairman: Stanisław P. Radziszowski
Reader: Edith Hemaspaandra
Observer: Peter G. Anderson

Abstract

In August 2002, three Indian researchers, Manindra Agrawal and his students Neeraj Kayal and Nitin Saxena at the Indian Institute of Technology in Kanpur, presented a remarkable algorithm (the AKS algorithm) in their paper "PRIMES is in P." It is the deterministic polynomial-time primality testing algorithm that determines whether an input number is prime or composite. Until now, there was no known deterministic polynomial-time primality testing algorithm, thus it has fundamental meaning for complexity theory.

This project centered around the AKS algorithm from "PRIMES is in P" paper. The objectives are both the AKS algorithm itself and theorems and lemmas showing the correctness of the algorithm. The first task of the project is to provide easy-to-follow explanations of the paper for average readers. The second task is to analyze the AKS algorithm in detail. Ideas and concepts in the algorithm are studied and possibilities of improvement of the algorithm are explored. Some simple programs are made for these experiments according to need.

Contents

Chapter 1

Overview of the project

Primality testing is a decision problem because if we are asked whether the given positive integer is prime or not, the answer must be "yes" or "no." The AKS algorithm proved primality testing is in the class P. That is why the title of the paper is "PRIMES is in P." PRIMES represents primality testing in their terminology. The class P problem can be solved in polynomial time in the size of the input of the problem. A method using the bit length of the given number n ($\simeq \log_2 n$) as the size is reasonable in PRIMES. Whether PRIMES is in the class P or not had been one of the most important problems in the theoretical computer science field for a long time.[27] Here is the brief history of the primality testing.[14]

- Eratosthenes, 276 BC-194 BC: the Eratosthenes Sieve
- Pratt '75: in NP
- Miller '76: $O((\log_2 n)^4)$-time solvable if the Extended Riemann Hypothesis is true
- Solovay and Strassen '77; Rabin '80: in coRP, still the choice in applications
- Adleman, Pomerance, and Rumely '83: deterministic $O((\log_2 n)^{\log_2 \log_2 \log_2 n})$-time algorithm
- Goldwasser and Kilian, '86: "Almost all" primes can be proven to be prime in $O((\log_2 n)^{12})$ time
- Adleman and Huang '87: in RP
- Fellows and Koblitz '92: in UP
- Agrawal, Kayal and Saxena 2002: in P, $O((\log_2 n)^{12}\mathrm{poly}(\log_2 \log_2 n))$-time

---

NOTE: Complexity Classes [27],[28]

NP(Nondeterministic Polynomial-time) If we can magically get some extra input, we can check if the given number is prime or not in polynomial time

coRP(complementary Randomized Polynomial-time) Solvable in polynomial time by randomized algorithms (The answer "prime" is probably right, "composite" is certainly right)

RP (Randomized Polynomial-time) Solvable in polynomial time by randomized algorithms (The answer "composite" is probably right, "prime" is certainly right)

UP(Unambiguous Polynomial time) The class of sets that are accepted by non-deterministic polynomial-time Turing machines that on no input have more than one accepting computation path [16]

P(polynomial-time) Solvable on a deterministic sequential machine in an amount of time that is polynomial in the size of the input

---

Many researchers have been interested in the primality testing algorithm from a theoretical standpoint. But some primality testing algorithm are also important from a practical standpoint because some cryptosystems such as RSA use this kind of algorithm for their key generation. RSA needs two large prime numbers for their key generation and a primality testing algorithm is used in this part of the system.[17]

Consequently, many researchers have provided various algorithms. There are some very efficient algorithms such as the Solovay-Strassen or Miller-Rabin algorithm. They are termed probabilistic polynomial time primality testing algorithms. (It is same as the class coRP.) The term "probabilistic" implies that they can make mistakes sometimes. That is to say there is a possibility that the algorithm may claim the given number is prime when in fact it is composite. But this error possibility can be reduced to an arbitrarily small percentage by repeating the test enough times. The algorithm is basically very fast, so the algorithm is practical enough in real cryptosystems.

Despite efforts of many researchers, no one could find the algorithm in the class P up to the AKS algorithm. Therefore, people believed that very complex mathematical techniques were necessary to solve this problem. However, Agrawal et al. accomplished their important achievement by using only a relatively basic number theory and group theory. That further surprised many researchers.

This Project and Report Chapter 2 explains the AKS algorithm and proofs in detail. We also have implemented the algorithm and conducted some experiments. Their results show some characteristics of the algorithm and help us to understand it better. They are shown in chapter 3.

Chapter 2

Detailed Commentary on "PRIMES is in P"

In this chapter a detailed explanation of "PRIMES is in P" is given. The theorems and lemmas in this chapter are quoted from the original paper. Other quotations are indented for clarity. A page number of a reference part of the original paper is indicated at each time.

2.1 Preparations

[2 Basic Idea and Approach] (page 2)

[Identity] (page 2) *Suppose that a is coprime to p. Then p is prime if and only if*

$$(x-a)^p \equiv (x^p - a) \pmod{p} \tag{2.1}$$

We do not care what x is specifically in this congruence. What is required in order to know whether this congruence is true or not is to compare each coefficients on both sides by mod p.

Proof. **Identity**

$$LHS = (x-a)^p = x^p + \sum_{i=1}^{p-1} (-1)^i {}_pC_i a^{p-i} x^i + (-1)^p a^p \tag{2.2}$$

If p is prime: $a^p \equiv a \pmod{p}$ by Fermat's Little Theorem. And if p is odd prime $(-1)^p a \equiv -a \pmod{p}$ and if $p = 2$, also $(-1)^2 a \equiv a \equiv -a \pmod{2}$, thus

$$LHS = (x-a)^p \equiv x^p + \sum_{i=1}^{p-1} (-1)^i {}_pC_i a^{p-i} x^i - a \pmod{p}. \tag{2.3}$$

Here

$${}_pC_i = \frac{p!}{(p-i)!i!} \qquad (1 \leq i \leq p-1). \tag{2.4}$$

Since $p - i < p$ and $i < p$, p is not a factor of the denominator, then ${}_pC_i$ is a multiple of p for $1 \leq i \leq p-1$. As a result,

$$ {}_pC_i \equiv 0 \pmod{p} \qquad (1 \leq i \leq p-1). \tag{2.5}$$

Therefore

$$ (2.3) \Rightarrow LHS \equiv x^p - a \pmod{p} \tag{2.6}$$
$$ \equiv RHS \pmod{p}. \tag{2.7}$$

If p is composite: Let $p = mq^k$, (q is a prime factor of p, $q \nmid m$, $k \geq 1$)

$$ {}_pC_q = \frac{p(p-1)\ldots\{p-(q-2)\}\{p-(q-1)\}}{q!} \tag{2.8}$$
$$ = \frac{mq^k(mq^k-1)\ldots\{mq^k-(q-2)\}\{mq^k-(q-1)\}}{q!} \tag{2.9}$$
$$ = \frac{mq^{k-1}(mq^k-1)\ldots\{mq^k-(q-2)\}\{mq^k-(q-1)\}}{(q-1)!} \tag{2.10}$$

Since $q \nmid (mq^k-1), \ldots, \{mq^k-(q-2)\}, \{mq^k-(q-1)\}$,

$$ q^k \nmid {}_pC_q \tag{2.11}$$
$$ \Rightarrow p \nmid {}_pC_q. \tag{2.12}$$

And a is coprime to p, then for the coefficient of x^q in LHS

$$ (-1)^q {}_pC_q a^{p-q} \not\equiv 0 \pmod{p} \tag{2.13}$$

On the other hand, the coefficient of x^q in RHS is zero $\pmod{p}$, hence

$$ (x-a)^p \not\equiv (x^p - a) \pmod{p}. \tag{2.14}$$

□

[3 Notation and Preliminaries] (pages 2-3)

[Lemma 3.1.] (page 2) *Let p and r be prime numbers, $p \neq r$ in fact 1 to fact 4.*

[Lemma 3.1.(fact 1)] (page 3) *The multiplicative group of any field F_{p^t} for $t > 0$, denoted by $F^*_{p^t}$ is cyclic.*

Proof. See [12]. □

[Lemma 3.1.(fact 2)] (page 3) *Let f(x) be a polynomial with integral coefficients. Then*

$$ f(x)^p \equiv f(x^p) \pmod{p} \tag{2.15}$$

Example: Let $f(x) = a_0 + a_1x + a_2x^2$ and $p = 3$, then

$$f(x)^3 = (a_0 + a_1x + a_2x^2)(a_0 + a_1x + a_2x^2)(a_0 + a_1x + a_2x^2). \tag{2.16}$$

Let $i_0 + i_1 + i_2 = 3$ (=p or the number of clauses above).

i_0	i_1	i_2	$i = 0i_0 + 1i_1 + 2i_2$	$a_0^{i_0}a_1^{i_1}a_2^{i_1}$	$\frac{p!}{i_0!i_1!i_2!}$	coefficient of x^i	x^i	mod 3
3	0	0	0	a_0^3	1	a_0^3	x^0	a_0
2	1	0	1	$a_0^2a_1$	3	$3a_0^2a_1$	x^1	0
1	2	0	2	$a_0a_1^2$	3	$3a_0a_1^2 + 3a_0^2a_2$	x^2	0
2	0	1		$a_0^2a_2$	3			
0	3	0	3	a_1^3	1	$a_1^3 + 6a_0a_1a_2$	x^3	a_1
1	1	1		$a_0a_1a_2$	6			
0	2	1	4	$a_1^2a_2$	3	$3a_1^2a_2 + 3a_0a_2^2$	x^4	0
1	0	2		$a_0a_2^2$	3			
0	1	2	5	$a_1a_2^2$	3	$3a_1a_2^2$	x^5	0
0	0	3	6	a_2^3	1	a_2^3	x^6	a_2

From the table above, we get $f(x)^3 = a_0 + a_1x^3 + a_2x^6 \equiv f(x^3) \pmod 3$.

For example, the term of x^2 can be found in two ways. Use a_0 from one of the three clauses in (2.16) and a_1x from the other two of the three clauses in (2.16), and multiple them, or use a_0 from two of the three clauses in (2.16) and a_2x^2 from the other one of the three clauses in (2.16) and multiple them. These two ways correspond to $i_0 = 1,\ i_1 = 2, i_2 = 0$ and $i_0 = 2,\ i_1 = 0, i_2 = 1$.

Proof. **Lemma 3.1.(fact 2)**
Let $f(x) = a_0 + a_1x + \cdots + a_dx^d$. The coefficient of x^i in $f(x)^p$ is

$$\sum_{\substack{i_0+\cdots+i_d=p \\ i_1+2i_2+\cdots+d_id=i}} a_0^{i_0} \ldots a_d^{i_d} \frac{p!}{i_0! \ldots i_d!} \tag{2.17}$$

That is because

$$i_0 + \cdots + i_d = p \text{ (= the number of multiplied terms.)} \tag{2.18}$$
$$i_0 \times 0 + i_1 \times 1 + i_2 \times 2 + \cdots + i_d \times d \tag{2.19}$$
$$= i_1 + 2i_2 + \cdots + d_id = i \text{(= the degree of the term.)} \tag{2.20}$$

Then if we fix the values of $i_0, \ldots,\ i_d$, then the number of ways picking up each term from p clauses is

$$\frac{p!}{i_0! \ldots i_d!}.$$

As for the coefficients of $f(x)^p$, there are two cases below. (Because $i_0 + \ldots + i_d = p$.)

(case 1) All i_k's are less than p. $(i_0, \ldots, i_d < p)$ In this case

$$\frac{p!}{i_0! \ldots i_d!} \equiv 0 \pmod{p}.$$

Because p is not a factor of the numerator.

(case 2) $i_j = p$ for $\exists 1 j$, and $i_k = 0$ for all k $(k \neq j)$ *i.e.* using the same terms, $a_j x^j$, from all clauses. In this case, from (2.20),

$$i = i_j \times j \tag{2.21}$$
$$= pj. \tag{2.22}$$

Since there is only one way to chose terms from the p clauses in this case, the coefficient of $x^i (= x^{pj})$ is

$$a_j^p \equiv a_j \pmod{p} \text{ [By Fermat's little theorem]}$$

From case 1 and 2, we get

$$f(x)^p \equiv a_0 x^{p \times 0} + a_1 x^{p \times 1} + \ldots + a_d x^{pd} \pmod{p} \tag{2.23}$$
$$\equiv f(x^p) \pmod{p} \tag{2.24}$$

□

[Lemma 3.1.(fact 3)] (page 3) *Let $h(x)$ be any factor of $x^r - 1$. Let $m \equiv m_r \pmod{r}$. Then*

$$x^m \equiv x^{m_r} \pmod{h(x)} \tag{2.25}$$

Proof. **Lemma 3.1.(fact 3)**
Let $m = kr + m_r$. Now

$$x^r \equiv 1 \pmod{x^r - 1} \tag{2.26}$$

Raise both sides to the power k

$$\Rightarrow x^{kr} \equiv 1^k \pmod{x^r - 1} \tag{2.27}$$
$$\Rightarrow x^{kr} \equiv 1 \pmod{x^r - 1} \tag{2.28}$$

Multiply both sides by x^{m_r}

$$\Rightarrow x^{kr + m_r} \equiv x^{m_r} \pmod{x^r - 1} \tag{2.29}$$
$$\Rightarrow x^m \equiv x^{m_r} \pmod{x^r - 1} \quad [\text{By } m = kr + m_r] \tag{2.30}$$

Since $h(x)$ is a factor of $x^r - 1$

$$\Rightarrow x^m \equiv x^{m_r} \pmod{h(x)} \tag{2.31}$$

□

[Lemma 3.1.(fact 4)] (page 3) *Let $o_r(p)$ be the order of p modulo r. Then in F_p, $\frac{x^r-1}{x-1}$ factorises into irreducible polynomials each of degree $o_r(p)$.*

Proof. **Lemma 3.1.(fact 4)**
Let $d = o_r(p)$ and $Q_r(x) = \frac{x^r-1}{x-1}$. Let $h(x)$ of degree k be an irreducible factor of $Q_r(x)$ in F_p. We prove $k = d$ by showing $k|d$ and $d|k$.

(a)Show $k|d$ Now $F_p[x]/h(x)$ forms a field of size p^k, F_{p^k}, and the multiplicative subgroup of $F_p[x]/h(x)$, $F^*_{p^k}$, is cyclic by Lemma 3.1 (fact 1) with a generator, say $g(x)$. Then, by Lemma 3.1 (fact 2),

$$g(x)^p \equiv g(x^p) \pmod{p} \tag{2.32}$$
$$\Rightarrow g(x)^p \equiv g(x^p) \pmod{h(x),\ p} \tag{2.33}$$
$$\Rightarrow g(x)^{p^d} \equiv g(x^{p^d}) \pmod{h(x),\ p} \tag{2.34}$$

Proof of the congruence (2.34) by induction.

The base case is the congruence (2.33).
The inductive hypothesis:

$$g(x)^{p^k} \equiv g(x^{p^k}) \pmod{h(x),\ p} \quad (k \geq 1) \tag{2.35}$$

The induction step:

$$g(x)^{p^{k+1}} \equiv g(x)^{p^k p} \tag{2.36}$$
$$\equiv (g(x)^{p^k})^p \tag{2.37}$$
$$\equiv (g(x^{p^k}))^p \quad \text{[By the induction hypothesis]} \tag{2.38}$$
$$\equiv g((x^p)^{p^k}) \quad \text{[By lemma 3.1.(fact 2)]} \tag{2.39}$$
$$\equiv g(x^{p^{k+1}}) \tag{2.40}$$

Therefore we get the congruence (2.34).

Since

$$p^d \equiv 1 \pmod{r} \quad \text{[By } d = o_r(p).\text{]} \tag{2.41}$$
$$h(x)|x^r - 1 \text{ in } F_p \quad \text{[By } h(x)|Q_r(x) \text{ and } Q_r(x)|x^r - 1 \text{ in } F_p.\text{]} \tag{2.42}$$

by lemma 3.1.(fact 3), we get

$$x^{p^d} \equiv x^1 \pmod{h(x),\ p} \tag{2.43}$$
$$\Rightarrow x^{p^d} \equiv x \pmod{h(x),\ p} \tag{2.44}$$
$$\Rightarrow g(x^{p^d}) \equiv g(x) \pmod{h(x),\ p}. \tag{2.45}$$

From the congruence (2.34) and (2.45),

$$g(x)^{p^d} \equiv g(x) \pmod{h(x),\ p} \tag{2.46}$$

$$\Rightarrow g(x)^{p^d-1} \equiv 1 \pmod{h(x),\ p}. \tag{2.47}$$

The order of $g(x)$ is $p^k - 1$ because $g(x)$ is a generator of $F^*_{p^k}$. That means $g(x)^{p^k-1} \equiv 1 \pmod{h(x),\ p}$, and $p^k - 1$ is the smallest such exponent. Thus we get

$$p^k - 1 | p^d - 1$$

which implies that

$$k|d. \tag{2.48}$$

Proof. $p^k - 1|p^d - 1 \Rightarrow k|d$.

Let $d = nk + a$ $(n \geq 1,\ 0 \leq a < k)$.

$$p^d - 1 = p^{nk+a} - 1 \tag{2.49}$$

$$= p^{nk}p^a - 1 \tag{2.50}$$

$$= (p^{nk} - 1)p^a + p^a - 1 \tag{2.51}$$

$$= (p^k - 1)(p^{k(n-1)} + p^{k(n-2)} + \ldots + p^k + 1)p^a + p^a - 1 \tag{2.52}$$

Therefore,

$$p^k - 1|p^d - 1 \Rightarrow p^a - 1 = 0 \tag{2.53}$$

$$\Rightarrow a = 0 \tag{2.54}$$

$$\Rightarrow k|d. \tag{2.55}$$

(b) Show $d|k$. Because $h(x)$ is a factor of $x^r - 1$, we have

$$x^r \equiv 1 \pmod{h(x),\ p} \tag{2.56}$$

Thus the order of x in $F^*_{p^k}$ is r. (since r is prime and $x \not\equiv 1$.) Recall that the order of an element in a cyclic group is a factor of the order of the cyclic group. The order of $F^*_{p^k}$ is $p^k - 1$. Therefore $r|p^k - 1$, *i.e.*

$$p^k - 1 \equiv 0 \pmod{r} \tag{2.57}$$

$$\Rightarrow p^k \equiv 1 \pmod{r} \tag{2.58}$$

$$\Rightarrow d|k \qquad [\text{by } d = O_r(p)] \tag{2.59}$$

From (a) and (b), we get $k = d$. □

[Lemma 3.2.] (page 3) *[6], [3] Let $P(n)$ denote the greatest prime divisor of n. There exist constants $c > 0$ and n_0 such that, for all $x > n_0$*

$$|\{p|p \text{ is prime, } p \leq x \text{ and } P(p-1) > x^{\frac{2}{3}}\}| \geq c\frac{x}{\log_2 x} \tag{2.60}$$

[Lemma 3.3.] (page 3) *[2] Let $\pi(n)$ be the number of primes $\leq n$. Then for $n \geq 1$*

$$\frac{n}{6\log_2 n} \leq \pi(n) \leq \frac{8n}{\log_2 n} \tag{2.61}$$

Proof. From the original book [2].

$$\pi(n) \geq \frac{n}{6\log_e n} \tag{2.62}$$

$$\geq \frac{n}{6\log_2 n} \tag{2.63}$$

And also from the book,

$$\pi(n) \leq \frac{n}{\log_e n}\left(6\log_e 2 + \frac{3}{e}\right) \tag{2.64}$$

$$= \frac{n\log_2 e}{\log_2 n}\left(\frac{6}{\log_2 e} + \frac{3}{e}\right) \tag{2.65}$$

$$= \frac{n}{\log_2 n}\left(6 + \frac{3\log_2 e}{e}\right) \tag{2.66}$$

$$\leq \frac{n}{\log_2 n}\left(6 + \frac{3 \times 1.45}{2.7}\right) \tag{2.67}$$

$$\leq \frac{8n}{\log_2 n} \tag{2.68}$$

□

2.2 The Algorithm

[4 The Algorithm] (pages 4-7)

The AKS algorithm (page 4)

Input: integer $n > 1$

1. if (n is of the form $a^b, b > 1$) output COMPOSITE;
2. $r = 2$;
3. while($r < n$) {
4. if ($\gcd(n, r) \neq 1$) output COMPOSITE;
5. if (r is prime) {
6. let q be the largest prime factor of $r - 1$;
7. if ($q \geq 4\sqrt{r}\log_2 n$) and ($n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$)
8. break; }
9. $r \leftarrow r + 1$;
10. }
11. for $a = 1$ to $2\sqrt{r}\log_2 n$
12. if ($(x - a)^n \not\equiv (x^n - a) \pmod{x^r - 1, n}$) output COMPOSITE;
13. output PRIME;

2.3 Terminology

We introduce the following terms for the arguments that are presented later.

$\boldsymbol{P(n)}$ The largest prime factor of n. (n is a composite number.)

$\boldsymbol{q}$ $= P(r - 1)$. (r is a prime.)

$\boldsymbol{o_r(n)}$ The order of n modulo r.

COND 1 $q \geq 4\sqrt{r}\log_2 n$ $(n > 1)$.

COND 2 $n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$.

COND 3 $q | o_r(n)$.

Useful Prime [14] r is a useful prime when it satisfies COND 1 and COND 2 (or COND 1 and COND 3 $\star$) for a given integer ($> n$).

$\star$ **NOTE** : COND 1 and COND 2 $\Leftrightarrow$ COND 1 and COND 3.

Proof. Assume COND 1. Then we show COND 3 $\Leftrightarrow$ COND 2.

(a) $q|o_r(n) \Rightarrow n^{\frac{r-1}{q}} \not\equiv 1 \pmod r$ (COND 3 $\Rightarrow$ COND 2) By COND 1

$$q > \sqrt{r-1} \tag{2.69}$$
$$\Rightarrow q^2 > r-1 \tag{2.70}$$
$$\Rightarrow q > \frac{r-1}{q} \tag{2.71}$$
$$\Rightarrow o_r(n) > \frac{r-1}{q} \quad [\text{By } q|o_r(n)] \tag{2.72}$$
$$\Rightarrow n^{\frac{r-1}{q}} \not\equiv 1 \pmod r \tag{2.73}$$

(b) $n^{\frac{r-1}{q}} \not\equiv 1 \pmod r \Rightarrow q|o_r(n)$ (COND 2 $\Rightarrow$ COND 3) We prove the contraposition, $q \not| o_r(n) \Rightarrow n^{\frac{r-1}{q}} \equiv 1 \pmod r$. Since $q \not| o_r(n)$ and $o_r(n)|r-1$,

$$o_r(n)|\frac{r-1}{q} \tag{2.74}$$
$$\Rightarrow n^{\frac{r-1}{q}} \equiv 1 \pmod r. \tag{2.75}$$

□

2.4 An Overview of the AKS Algorithm

If we consider the AKS algorithm as a filter for composite numbers, it is easy to understand how the algorithm works. Only prime numbers can reach the last line 13. Composite numbers are detected by filters at line 1, line 4, or line 12.

The Role of the While-loop The role of the while-loop is to find the smallest useful prime r for a given number n. If n is large enough ($n > n_0, n_0$ is a constant.), Lemma 4.2. assures that a useful prime r exists in the interval $[c_1(\log_2 n)^6, c_2(\log_2 n)^6]$. ($c_1$ and c_2 are constants.) This means that the while-loop does not repeat until the upper bound $r = n-1$, and breaks at line 8 in no more than $c_2(\log_2 n)^6$ iterations. Thus even if a primitive algorithm like the Brute Force Method is applied for the GCD check at line 4 or the prime check at line 5, the algorithm can work in polynomial time.

The Filter at Line 1 It detects special form composite numbers such that $a^b, b > 1$.

The Filter at Line 4 It detects composite numbers that have a smaller prime factor than their useful primes. A composite n that is not detected here satisfies the next condition.

COND 6: "Every prime factor of n" $> r$

The Filter at Line 12 This is the "main" filter. All other composites not detected at line 1 or line 4 are detected here.

The Behavior of the AKS Algorithm for Different Kind of Numbers

1. n is composite of the form $a^b, b > 1$.
   It returns COMPOSITE at line 1.

2. n is prime, or composite not of the form $a^b, b > 1$.

   (a) A useful prime ($< n$) does not exist.

      i. n is composite.
      It returns COMPOSITE at line 4. (It works like the Brute Force Method.)

      ii. n is prime.
      The while-loop iterates until $r = n - 1$.
      The for-loop iterates until $a = 2\sqrt{r}\log_2 n$.
      It returns PRIME at line 13.

   (b) A useful prime ($< n$) exists.

      i. n is composite.

         A. n has a prime factor ($<$ the smallest useful prime for n.)
         It returns COMPOSITE at line 4.

         B. n does not have a prime factor ($<$ the smallest useful prime for n.)
         The while-loop finds a useful prime r ($< n$) and breaks at line 8.
         It returns COMPOSITE at line 12 in the for-loop.

      ii. n is prime. The while-loop finds a useful prime r ($< n$) and breaks at line 8.
      The for-loop iterates until $a = 2\sqrt{r}\log_2 n$.
      It returns PRIME at line 13.

2.5 A Proof of Correctness of the AKS Algorithm

[Theorem 4.1.] (page 4) *The algorithm above returns PRIME if and only if n is prime.*

This theorem claims that the AKS algorithm always returns the correct answer. Whether the algorithm can work in polynomial time or not is not mentioned here. That is showed in the section 5 in "PRIMES is in P".

Proof. This is proved through Lemma 4.2. to Lemma 4.7. □

2.5.1 Existence of Useful Primes

[Lemma 4.2.] (page 4) *There exist positive constants c_1 and c_2 for which there is a prime r in the interval $[c_1(\log_2 n)^6, c_2(\log_2 n)^6]$ such that $r - 1$ has a prime factor $q \geq 4\sqrt{r}\log_2 n$ and $q | o_r(n)$.*

This lemma means that the while-loop does not iterate until its upper bound $n-1$, and before that a useful prime r that is at most the size of about $(\log_2 n)^6$ can be found, and then breaks at line 8.

NOTE This lemma assumes that $n > n_0$ (n_0 is a constant). Otherwise a useful prime r for the given n could not be found, even though composite numbers can be detected by the check on line 4 of the algorithm. In this case the algorithm works like the Brute Force Method not in polynomial time but in exponential time. Thus, formally, Lemma 4.2. is not needed for a proof of Theorem 4.1. However this lemma is necessary for assuring the algorithm working in polynomial time.

Proof. **Lemma 4.2.**
The lemma says r meets COND 1 and COND 3 (or COND 2).
Now consider following sets. Let

$$S = \{p \mid p \text{ is prime, } P(p-1) > p^{\frac{2}{3}}, p \leq c_2(\log_2 n)^6\} \tag{2.76}$$
$$R = \{p \mid p \text{ is prime, } P(p-1) > p^{\frac{2}{3}}, c_1(\log_2 n)^6 \leq p \leq c_2(\log_2 n)^6\} \tag{2.77}$$
$$T = \{p \mid p \text{ is prime, } p \leq c_1(\log_2 n)^6\}. \tag{2.78}$$

Let $c_1 \geq 4^6$. For $r \in R$,

$$r \geq c_1(\log_2 n)^6 \tag{2.79}$$
$$\geq 4^6(\log_2 n)^6 \tag{2.80}$$
$$\Rightarrow r^{\frac{1}{6}} \geq 4\log_2 n \tag{2.81}$$

While

$$P(r-1) \geq r^{\frac{2}{3}} \tag{2.82}$$
$$= r^{(\frac{1}{2}+\frac{1}{6})} \tag{2.83}$$
$$= \sqrt{r} \cdot r^{\frac{1}{6}} \tag{2.84}$$
$$\geq 4\sqrt{r}\log_2 n \quad \text{[By (2.81)]}. \tag{2.85}$$

Thereby $r \in R$ satisfies COND 1. Consider the bound.

$$|R| \geq |S| - |Q| \tag{2.86}$$

We get the minimum number of $|S|$ by substituting $c_2(\log_2 n)^6$ for x of Lemma 3.2.

$$|S| \geq c\frac{c_2(\log_2 n)^6}{\log_2 c_2(\log_2 n)^6} \tag{2.87}$$
$$= \frac{cc_2(\log_2 n)^6}{\log_2 c_2 + 6\log_2\log_2 n}. \tag{2.88}$$

Since $c_2 \leq \log_2 n$, where n is large enough, then by substituting $\log_2 n$ for c_2 in the denominator, we get

$$|S| \geq \frac{cc_2(\log_2 n)^6}{\log_2 \log_2 n + 6\log_2 \log_2 n} \tag{2.89}$$

$$= \frac{cc_2(\log_2 n)^6}{7\log_2 \log_2 n} \tag{2.90}$$

While the maximum of $|Q|$ is obtained by Lemma 3.3.

$$|Q| \leq \frac{8c_1(\log_2 n)^6}{\log_2 c_1(\log_2 n)^6} \tag{2.91}$$

$$= \frac{8c_1(\log_2 n)^6}{\log_2 c_1 + 6\log_2 \log_2 n} \tag{2.92}$$

$$< \frac{8c_1(\log_2 n)^6}{6\log_2 \log_2 n} \tag{2.93}$$

From (2.86), (2.90) and (2.93),

$$|R| > \frac{cc_2(\log_2 n)^6}{7\log_2 \log_2 n} - \frac{8c_1(\log_2 n)^6}{6\log_2 \log_2 n} \tag{2.94}$$

$$= \frac{(\log_2 n)^6}{\log_2 \log_2 n}\left(\frac{cc_2}{7} - \frac{8c_1}{6}\right) \tag{2.95}$$

Choose constants c_2 so that the quantity in braces is a positive constant, say c_3.

$$|R| > \frac{c_3(\log_2 n)^6}{\log_2 \log_2 n} \tag{2.96}$$

The remaining part needs to show that at least one prime satisfying COND 3 (or COND 2) exists in R. Consider a prime t such that

$$t \in T = \{t \mid t \in R,\ t \text{ does not satisfy COND 2}\} \subseteq R \tag{2.97}$$

Since t does not satisfy COND 2,

$$n^{\frac{t-1}{P(t-1)}} \equiv 1 \pmod{t}. \tag{2.98}$$

Since $t \in R$,

$$P(t-1) \geq t^{\frac{2}{3}}. \tag{2.99}$$

By this

$$\frac{t-1}{P(t-1)} < \frac{t}{t^{\frac{2}{3}}} \tag{2.100}$$

$$= t^{\frac{1}{3}} \tag{2.101}$$

$$\leq x^{\frac{1}{3}} \quad [\text{Here } x = c_2(\log_2 n)^6] \tag{2.102}$$

Thus, from (2.98), at least one of the following congruence holds.

$$n^1 \equiv 1 \pmod{t}, n^2 \equiv 1 \pmod{t}, \ldots, n^{x^{\frac{1}{3}}} \equiv 1 \pmod{t} \tag{2.103}$$

In other words, t is a divisor of one of the following.

$$n^1 - 1, n^2 - 1, \ldots, n^{x^{\frac{1}{3}}} - 1 \tag{2.104}$$

Hence $|T|$ can be bounded by the number of factors of the following product.

$$\prod = (n-1)(n^2-1)\ldots(n^{x^{\frac{1}{3}}} - 1). \tag{2.105}$$

Thus

$$\begin{aligned}
|T| &\leq \log_2 \prod && (2.106)\\
&= \sum_{i=1}^{x^{\frac{1}{3}}} \log_2(n^i - 1) && (2.107)\\
&< \sum_{i=1}^{x^{\frac{1}{3}}} \log_2 n^i && (2.108)\\
&= \log_2 n \sum_{i=1}^{x^{\frac{1}{3}}} i && (2.109)\\
&= \frac{x^{\frac{1}{3}}(x^{\frac{1}{3}}+1)}{2} \log_2 n && (2.110)\\
&< x^{\frac{2}{3}} \log_2 n && (2.111)\\
&= c_2(\log_2 n)^5 \qquad [\text{By } x = c_2(\log_2 n)^6]. && (2.112)
\end{aligned}$$

To complete the proof, we need to show $|R| - |T| \geq 1$. From (2.95) and (2.112)

$$\begin{aligned}
&|R| - |T| && (2.113)\\
\geq &\frac{(\log_2 n)^6}{\log_2 \log_2 n} \left(\frac{cc_2}{7} - \frac{8c_1}{6} \right) - c_2(\log_2 n)^5 && (2.114)\\
= &\frac{(\log_2 n)^6}{\log_2 \log_2 n} \left\{ c_2 \left(\frac{c}{7} - \frac{\log_2 \log_2 n}{\log_2 n} \right) - \frac{8c_1}{6} \right\} && (2.115)\\
& && (2.116)
\end{aligned}$$

For n, which is large enough

$$\frac{c}{7} - \frac{\log_2 \log_2 n}{\log_2 n} > 0. \tag{2.117}$$

Thus we can make $|R| - |T|$ greater than or equal to 1 by choosing a c_2, that is large enough. □

2.5.2 If n is Prime

[Lemma 4.3.] (page 5) *If n is prime, the algorithm returns PRIME.*

Proof. **Lemma 4.3.**
Let n be a prime. The while-loop cannot return COMPOSITE since $\gcd(n, r) = 1$ for all $r < n$.
Concerning the for-loop: By Lemma 3.1. (fact 2) for all a's $(1 \leq a \leq 2\sqrt{r}\log_2 n)$

$$(x-a)^n \equiv (x^n - a) \pmod{n} \tag{2.118}$$
$$\Rightarrow (x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}. \tag{2.119}$$

Thus the for-loop also cannot return COMPOSITE. Thereby, the algorithm returns PRIME. □

2.5.3 If n is Composite

Now we consider the case that the given number n is composite. If n is a type of number that can be detected at line 1 or line 4, it is obvious that the algorithm works correctly. Then what we need to prove here is another type of composite number that can be detected as composite by the check at line 12 in the for-loop. For the reminder of the argument, we reconfirm the conditions about the useful prime r and the given number n in such case.

As for r:

COND 1: $q \geq 4\sqrt{r}\log_2 n$ (or $q \geq 2\ell, \ell = 2\sqrt{r}\log_2 n$) [By the check at line 7]

COND 2: $n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$ [By the check at line 7]

COND 3 : $q | o_r(n)$ [By the check at line 7]

(NOTE: COND 1 and COND 2 $\Leftrightarrow$ COND 1 and COND 3)

As for n:

COND 4: n is composite [assumption]

COND 5: n is not of the form $a^b, (b > 1)$ [By the check at line 1]

COND 6: "Every prime factor of n" $> r$ [By the check at line 4]

[Statement after Lemma 4.3.] (page 5)

> Now let us turn our attention to the case where a composite n is input to our algorithm. The significance of the r found by the while-loop arises when n is composite with say $p_i, 1 \leq i \leq k$, as its prime factors. In this case $o_r(n) | \text{lcm}\{o_r(p_i)\}$.

Let $\lambda = \text{lcm}\{o_r(p_i)\}$. We can write $\lambda = o_r(p_i)m_i$ by choosing the appropriate m_i for each $o_r(p_i)$, and $n = \prod p_i^{j_i}$ $(1 \leq i \leq k)$. By using them

$$n^\lambda = (\prod_{i=1}^{k} {p_i}^{j_i})^\lambda \tag{2.120}$$

$$= \prod_{i=1}^{k} {p_i}^{\lambda j_i} \tag{2.121}$$

$$= \prod_{i=1}^{k} ({p_i}^{o_r(p_i)})^{m_i j_i} \tag{2.122}$$

$$= \prod_{i=1}^{k} 1^{m_i j_i} \tag{2.123}$$

$$= 1 \tag{2.124}$$

Thus $o_r(n)|\text{lcm}\{o_r(p_i)\}$.

And hence there exists a prime factor p of n such that $q|o_r(p)$, where q is the largest prime factor of $r-1$.

From $o_r(n)|\text{lcm}\{o_r(p_i)\}$ and $q|o_r(n)$ (COND 3), we get $q|\text{lcm}\{o_r(p_i)\}$. Since every prime factor of $\text{lcm}\{o_r(p_i)\}$ is a prime factor of at least one of the $o_r(p_i)$, we can say "there exists a prime factor p of n such that $q|o_r(p)$" mentioned above.

For the remainder of the argument, let p be such a prime factor of n.

COND 3A : $q|o_r(p)$

The second loop of the algorithm uses the value of r obtained to do polynomial computations on $\ell = 2\sqrt{r}\log_2 n$ binomials: $(x-a)$ for $1 \leq a \leq \ell$. By Lemma 3.1 (fact 4), we have a polynomial $h(x)$ (factor of $x^r - 1$) of degree $d = o_r(p)$ irreducible in F_p. Note that

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n} \tag{2.125}$$

implies that

$$(x-a)^n \equiv (x^n - a) \pmod{h(x), p} \tag{2.126}$$

So the identities on each binomial hold in the field $F_p[x]/(h(x))$. The set of ℓ binomials form a large cyclic group in this field:

For a polynomial $f(x)$

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n} \tag{2.127}$$

$$\Rightarrow (x-a)^n - (x^n - a) \equiv (x^r - 1)f(x) \pmod{n} \tag{2.128}$$

$$\Rightarrow (x-a)^n - (x^n - a) \equiv (x^r - 1)f(x) \pmod{p} \quad [\text{By } p|n] \tag{2.129}$$

By Lemma 3.1 (fact 4), for a polynomial $g(x)$

$$\Rightarrow (x-a)^n - (x^n - a) \equiv h(x)g(x)f(x) \pmod{p} \tag{2.130}$$

Thus

$$\Rightarrow (x-a)^n \equiv (x^n - a) \pmod{h(x), p} \tag{2.131}$$

[Lemma 4.4.] (page 5) *In the field $F_p[x]/(h(x))$, the group generated by the ℓ binomials: $(x-a), 1 \leq a \leq \ell$ i.e,*

$$G = \{ \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} | \alpha_a \geq 0, \forall 1 \leq a \leq \ell \}. \tag{2.132}$$

is cyclic and of size $> \left(\frac{d}{\ell}\right)^{\ell}$.

Proof. **Lemma 4.4.**
G is closed under multiplication in $F_p[x]/(h(x))$ since

$$\prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} \times \prod_{1 \leq a \leq \ell} (x-a)^{\beta_a} = \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a + \beta_a} \in G \tag{2.133}$$

$$\Rightarrow \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} \times \prod_{1 \leq a \leq \ell} (x-a)^{\beta_a} \equiv \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a + \beta_a} \in G \pmod{h(x), p}. \tag{2.134}$$

Thereby G is a cyclic group since it is a subgroup of the cyclic group $(F_p[x]/(h(x)))^*$.
Now consider the set

$$S = \{ \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} | \sum_{1 \leq a \leq \ell} \alpha_a \leq d-1, \alpha_a \geq 0, \forall 1 \leq a \leq \ell \}, \tag{2.135}$$

which is a subset of G. The following argument shows that all the elements of S are distinct in $F_p[x]/(h(x))$.
$r > q$ since q is a prime factor of $r-1$, and $q \geq 2\ell$ by COND 1. Hence

$$r > \ell. \tag{2.136}$$

While $p > r$ by COND 6, thus

$$p > \ell. \tag{2.137}$$

Since $1 \leq a \leq \ell$, none of the a's are congruent modulo p.
For two elements of S, $\prod (x-a)^{\alpha_a}$ and $\prod (x-a)^{\beta_a}$, suppose

$$\prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} \equiv \prod_{1 \leq a \leq \ell} (x-a)^{\beta_a} \pmod{h(x), p}. \tag{2.138}$$

Since the degree of any element of S is less than d, which is the degree of $h(x)$, mod $h(x)$ can be taken away.

$$(2.138) \Rightarrow \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} \equiv \prod_{1 \leq a \leq \ell} (x-a)^{\beta_a} \pmod{p} \tag{2.139}$$

Suppose $\alpha_k > \beta_k$ for a k $(1 \le k \le \ell)$. Divide the congruence (2.139) by $(x-k)^{\beta_k}$. If k is substituted for x, LHS $\equiv 0 \pmod{p}$ because LHS has $(x-k)^{\alpha_k - \beta_k}$ and $\alpha_k - \beta_k > 0$. While RHS $\not\equiv 0 \pmod{p}$ because RHS does not have the term of $(x-k)$ and $k - a \not\equiv 0 \pmod{p}$ for all $a \neq k$. Contradiction. Thus

$$\alpha_a = \beta_a \text{ (for all } 1 \le a \le \ell). \tag{2.140}$$

This implies that all elements of S are distinct in $F_p[x]/(h(x))$.

The cardinality of S corresponds to the number of different combinations of α_a's such that $\sum_{(1 \le a \le \ell)} \alpha_a \le d-1, \alpha_a \ge 0$. That can be obtained as follows.

Number of @: $d-1$
Number of | (partition): ℓ

e.g. @@@@ | @@@@@ | @@@ | ... | @@@@@@@@ | @@@@@@

Let the number of @'s in the left segment of k-th | be α_k. Then the cardinality of S is equal to the number of ways of such partitioning. That is

$$\binom{\ell + d - 1}{\ell} = \frac{(\ell + d - 1)(\ell + d - 2)\ldots(d)}{\ell!} \tag{2.141}$$

$$> \left(\frac{d}{\ell}\right)^{\ell}. \tag{2.142}$$

Since S is just a subset of G we get the result. □

[Statement after the proof of Lemma 4.4.] (page 5)

Since $d \ge 2\ell$, size of G is $> 2^{\ell} = n^{2\sqrt{r}}$.

Comment for the statement above:

$$d = o_r(p) \ge q \qquad \text{[By } q | o_r(p) \text{ (COND 3A)]} \tag{2.143}$$

$$\ge 4\sqrt{r}\log_2 n = 2\ell \qquad \text{[By COND 1]} \tag{2.144}$$

$$\Rightarrow \frac{d}{\ell} \ge 2. \tag{2.145}$$

Hence,

$$|G| \ge |S| \ge \left(\frac{d}{\ell}\right)^{\ell} \ge 2^{\ell} = 2^{2\sqrt{r}\log_2 n} = n^{2\sqrt{r}} \tag{2.146}$$

$$\Rightarrow |G| \ge n^{2\sqrt{r}}. \tag{2.147}$$

[Statement after the proof of Lemma 4.4. (continue)] (page 5)

Let $g(x)$ be a generator of G. Clearly, the order of $g(x)$ in $F_p[x]/(h(x))$ is $> n^{2\sqrt{r}}$. We now define a set related to $g(x)$ which will play an important role in the remaining arguments. Let

$$I_{g(x)} = \{m | g(x)^m \equiv g(x^m) \pmod{x^r - 1, p}\}. \tag{2.148}$$

Here is a nice property of $I_{g(x)}$:

[Lemma 4.5.] (page 6) *The set $I_{g(x)}$ is closed under multiplication.*

Proof. Let $m_1, m_2 \in I_{g(x)}$. Then

$$g(x)^{m_1} \equiv g(x^{m_1}) \pmod{x^r - 1, p}, \tag{2.149}$$

and

$$g(x)^{m_2} \equiv g(x^{m_2}) \pmod{x^r - 1, p}. \tag{2.150}$$

Also we have, by substituting x^{m_1} in place of x in the second congruence:

$$g(x^{m_1})^{m_2} \equiv g(x^{m_1 m_2}) \pmod{x^{m_1 r} - 1, p} \tag{2.151}$$
$$\Rightarrow g(x^{m_1})^{m_2} \equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p} \tag{2.152}$$
$$[\text{By } x^{m_1 r} - 1 = (x^r - 1)(x^{(m_1-1)r} + x^{(m_1-2)r} + \cdots + x + 1).] \tag{2.153}$$

From these, we get

$$g(x)^{m_1 m_2} \equiv (g(x)^{m_1})^{m_2} \pmod{x^r - 1, p} \tag{2.154}$$
$$\equiv g(x^{m_1})^{m_2} \pmod{x^r - 1, p} \quad [\text{By (2.149)}] \tag{2.155}$$
$$\equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p} \quad [\text{By (2.152)}] \tag{2.156}$$
$$\tag{2.157}$$

Hence $m_1 m_2 \in I_{g(x)}$. □

Now we prove a property of $I_{g(x)}$ that plays a crucial role in our proof.

[Lemma 4.6.] (page 6) *Let the order of $g(x)$ in $F_p[x]/(h(x))$ be o_g. Let $m_1, m_2 \in I_{g(x)}$. Then $m_1 \equiv m_2 \pmod{r}$ implies that $m_1 \equiv m_2 \pmod{o_g}$.*

Proof. Since $m_1 \equiv m_2 \pmod{r}, m_2 = m_1 + kr$ for some $k \geq 0$. Since $m_2 \in I_{g(x)}$: (in what follows, the congruence are in the field $F_p[x]/(h(x))$ unless indicated otherwise.)

$$g(x)^{m_2} \equiv g(x^{m_2}) \pmod{x^r - 1, p} \tag{2.158}$$
$$\Rightarrow g(x)^{m_2} \equiv g(x^{m_2}) \pmod{h(x), p} \tag{2.159}$$
$$\Rightarrow g(x)^{m_1 + kr} \equiv g(x^{m_2}) \pmod{h(x), p} \tag{2.160}$$

By Lemma 3.1 (fact 3),

$$m_1 \equiv m_2 \pmod{r} \tag{2.161}$$
$$\Rightarrow x^{m_1} \equiv x^{m_2} \pmod{h(x)} \tag{2.162}$$

By this,

$$(2.160) \Rightarrow g(x)^{m_1+kr} \equiv g(x^{m_1}) \pmod{h(x), p} \tag{2.163}$$
$$\Rightarrow g(x)^{m_1} g(x)^{kr} \equiv g(x^{m_1}) \pmod{h(x), p} \tag{2.164}$$
$$\Rightarrow g(x)^{m_1} g(x)^{kr} \equiv g(x)^{m_1} \pmod{h(x), p} \quad [\text{ By } m_1 \in I_{g(x)}] \tag{2.165}$$

Now $g(x) \not\equiv 0$ implies $g(x)^{m_1} \not\equiv 0$ and hence we can cancel $g(x)^{m_1}$ from both sides leaving us with

$$g(x)^{kr} \equiv 1 \pmod{h(x), p}. \tag{2.166}$$

Since $g(x)$ is a generator of G, $o_g | kr$. Therefore,

$$kr \equiv 0 \pmod{o_g} \tag{2.167}$$
$$\Rightarrow m_2 \equiv m_1 \pmod{o_g} \tag{2.168}$$

□

[Statement after the proof of Lemma 4.6.] (page 6)

> The above property implies that there are "very few" ($\leq r$) numbers in $I_{g(x)}$ that are less than o_g. Now we are ready to prove the most important property of our algorithm.

Comment for the statement above: The contraposition of Lemma 4.6. is:

> Let $m_1, m_2 \in I_{g(x)}$. Then $m_1 \not\equiv m_2 \pmod{o_g}$ implies that $m_1 \not\equiv m_2 \pmod{r}$.

Consider the positive integers less than o_g in I_g. Since they are distinct from each other by modulo o_g, they must be also distinct by modulo r from the contraposition above. Then the number of such integers is at most r. This means there are only at most r elements of $I_{g(x)}$ in the range of less than o_g. The next proof of Lemma 4.7. is based on this idea.

[Lemma 4.7.] (page 6) *If n is composite, the algorithm returns COMPOSITE.*

Proof. Suppose that the algorithm returns PRIME instead. Thus, the for-loop ensures that for all $1 \leq a \leq 2\sqrt{r}\log_2 n$,

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, p}. \tag{2.169}$$

Notice that $g(x)$ is just a product of powers of ℓ binomials $(x-a), (1 \leq a \leq \ell)$ all of which satisfy congruence (2.169). Let

$$g(x) = \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a}. \tag{2.170}$$

Then

$$\begin{aligned} g(x)^n &= \prod_{1 \leq a \leq \ell} (x-a)^{n\alpha_a} && (2.171)\\ &\equiv \prod_{1 \leq a \leq \ell} (x^n - a)^{\alpha_a} \pmod{x^r - 1, p} \quad [\text{By } (2.169)] && (2.172)\\ &\equiv g(x^n) \pmod{x^r - 1, p} && (2.173) \end{aligned}$$

Therefore, $n \in I_{g(x)}$.
On the other hand, by Lemma 3.1. (fact 2)

$$\begin{aligned} & g(x)^p = g(x^p) \pmod{p} && (2.174)\\ \Rightarrow & g(x)^p = g(x^p) \pmod{x^r - 1, p} && (2.175)\\ \Rightarrow & p \in I_{g(x)}. && (2.176) \end{aligned}$$

Trivially $1 \in I_{g(x)}$. We will now show that the set $I_g(x)$ has "many" numbers less than o_g contradicting Lemma 4.6.
Consider the set

$$E = \{n^i p^j | 0 \leq i, j \leq \lfloor \sqrt{r} \rfloor\}. \tag{2.177}$$

Since "$|E| = (1 + \lfloor \sqrt{r} \rfloor)^2 > r$" in the original paper is not precise, we change the proof a little.
By Lemma 4.5, $E \subseteq I_{g(x)}$. Since, for all $\leq i, j \leq \lfloor \sqrt{r} \rfloor$,

$$1 \leq n^i p^j < n^{\sqrt{r}} n^{\sqrt{r}} < o_g. \tag{2.178}$$

Thus all the elements of E are distinct modulo o_g. By the contraposition of Lemma 4.6. they are also distinct modulo r. Hence $|E| \leq r$. Since "the number of pairs (i,j)" $= (1+\lfloor \sqrt{r} \rfloor)^2 > r$, there are two elements $n^{i_1} p^{j_1}$ and $n^{i_2} p^{j_2}$ in E with $i_1 \neq i_2$ or $j_1 \neq j_2$, $i_1 \geq i_2$ such that $n^{i_1} p^{j_1} = n^{i_2} p^{j_2}$ by pigeon-hole principle.

$$\Rightarrow n^{i_1 - i_2} p^{j_1} = p^{j_2}. \tag{2.179}$$

If $i_1 = i_2$, $p^{j_1} = p^{j_2}$, then $j_1 = j_2$: a contradiction. Thus $i_1 > i_2$ $(i_1 - i_2 > 0)$. Since p is prime, this equality implies $n = p^k$ for some $k \geq 1$. However, in step 1 of the algorithm, composite numbers of the form p^k for $k \geq 2$ are already detected. Therefore, $n = p$: a contradiction. □

This completes the proof of theorem.

2.6 Time Complexity Analysis

[5 Time Complexity Analysis] (pages 7-8)

It is straightforward to calculate the time complexity of the algorithm.

[Theorem 5.1.] (page 7) *The asymptotic time complexity of the algorithm is* $\tilde{O}((\log_2 n)^{12})$.

Proof. Hereafter, we check the algorithm line by line to estimate the time complexity. We use the symbol $\tilde{O}(t(n))$ for $O((t(n)\mathrm{poly}(\log_2 t(n)))$, where $t(n)$ is some function of n.

Line 1: if (n is of the form $a^b, b > 1$) output COMPOSITE;
Suppose

$$n = a^b \tag{2.180}$$
$$\Rightarrow \log_a n = b \tag{2.181}$$
$$\Rightarrow \log_2 n \geq b \tag{2.182}$$

Since b is integer, b is $2, 3, \ldots,$ or $\lfloor \log_2 n \rfloor$. And $a = n^{\frac{1}{b}}$ is also integer. Thus line 1 is achieved by checking if there is an integer in the following sequence.

$$n^{\frac{1}{2}}, n^{\frac{1}{3}}, \ldots, n^{\frac{1}{\lfloor \log_2 n \rfloor}}, \tag{2.183}$$

Thereby line 1 takes asymptotic time: $O((\log_2 n)^3)$.

Line 2: $r = 2$ constant time

Line 3 to 10: while$(r < n)$ { ... }

By Lemma 4.2. r can be found in $[c_1(\log_2 n)^6, c_2(\log_2 n)^6]$. Hence the number of iterations of the while-loop is

$$O((\log_2 n)^6). \tag{2.184}$$

Let us now measure the work done in one iteration of the while-loop.

Line 4: if ($\gcd(n, r) \neq 1$) output COMPOSITE;

$$\mathrm{poly}(\log_2 n) \tag{2.185}$$

Line 5: if (r is prime)
Line 6: let q be the largest prime factor of $r - 1$;

Line 5 and 6 would take at most

$$r^{\frac{1}{2}}\mathrm{poly}(\log_2 n) \tag{2.186}$$

time in the brute-force implementation.

Line 7: if $(q \geq 4\sqrt{r}\log_2 n)$ and $(n^{\frac{r-1}{q}} \not\equiv 1 \pmod r)$
Line 8: break;
Line 9: $r \leftarrow r+1$;

Line 7,8 and 9 take at most

$$\text{poly}(\log_2 n). \tag{2.187}$$

From (2.184), (2.185), (2.186), and (2.187), total asymptotic time taken by the while-loop is

$$\begin{aligned} & O((\log_2 n)^6 r^{\frac{1}{2}} \text{poly}(\log_2 n)) && (2.188)\\ =& O((\log_2 n)^9 \text{poly}(\log_2 n)) \quad [\text{ By } r = O((\log_2 n)^6)] && (2.189)\\ & && (2.190) \end{aligned}$$

Line 11: for $a = 1$ to $2\sqrt{r}\log_2 n$
Line 12: if ($(x-a)^n \not\equiv (x^n - a) \pmod{x^r - 1, n}$) output COMPOSITE;

The for-loop does modular computation over polynomials. If repeated-squaring and Fast-Fourier Multiplication is used then one iteration of this for-loop takes

$$\begin{aligned} & \tilde{O}(\log_2 n \cdot r \log_2 n) && (2.191)\\ =& O(r(\log_2 n)^2 \text{poly}(\log_2(r(\log_2 n)^2))) && (2.192) \end{aligned}$$

steps. Thus, the for-loop takes asymptotic time

$$\begin{aligned} & O(r(\log_2 n)^2 \text{poly}(\log_2(r(\log_2 n)^2) \times r^{\frac{1}{2}} \log_2 n) && (2.193)\\ =& O(r^{\frac{3}{2}}(\log_2 n)^3 \text{poly}(\log_2(r(\log_2 n)^2)) && (2.194)\\ =& O((\log_2 n)^{12} \text{poly}(\log_2(\log_2 n)^8)) \quad [\text{ By } r = O((\log_2 n)^6)] && (2.195)\\ =& O((\log_2 n)^{12} \text{poly}(8 \log_2 \log_2 n)) && (2.196)\\ =& O((\log_2 n)^{12} \text{poly}(12 \log_2 \log_2 n)) && (2.197)\\ =& O((\log_2 n)^{12} \text{poly}(\log_2(\log_2 n)^{12})) && (2.198)\\ =& \tilde{O}((\log_2 n)^{12}) && (2.199) \end{aligned}$$

□

[Statement after the proof of Theorem 5.1.] (page 7)

> In practice, however, our algorithm is likely to work much faster. The reason is that even though we only know that there are "many" primes r such that $P(r-1) > r^{\frac{2}{3}}$, a stronger property is believed to be true. In fact it is believed that for many primes r, $P(r-1) = \frac{r-1}{2}$. (Such primes are called *Sophie Germain primes.*)

[**Definition 5.2.**] **(page 7)** If both r and $\frac{r-1}{2}$ are primes, then $\frac{r-1}{2}$ is a Sophie Germain prime. We will call such r's as co-Sophie Germain primes.

The following conjecture gives the density of Sophie Germain primes. This conjecture has been verified for $r \leq 10^{10}$:

Conjecture 5. (page 7) [HL22]*The number of co-Sophie Germain primes is asymptotic to* $\frac{Dx}{(\log_e x)^2}$ *, where D is the twin prime constant (estimated by Wrench and others to be approximately 0.6601618...).*

If this conjecture is true, then the while-loop exits with a "suitable" r of size $O((\log_2 n)^2)$:

Lemma 5.3. *Assuming the conjecture 5, there exists "suitable" r in the range* $64(\log_2 n)^2$ *to* $c_2(\log_2 n)^2$ *for all* $n > n_0$*, where* n_0 *and* c_2 *are positive constants.*

Proof. Note: "suitable" r's are the prime numbers that meet COND 1 and COND 3.

$$\text{COND 1: } q \geq 4\sqrt{r}\log_2 n \tag{2.200}$$
$$\text{COND 3: } q | o_r(n) \tag{2.201}$$

Let r is a co-Sophie Germain prime and $q = \frac{r-1}{2}$ that is a Sophie Germain prime. The minimum value of q is 2 when r is 5. Since $o_r(n)|r-1$ and $r-1 = 2q$, $o_r(n)|2q$. That is

$$o_r(n) \in \{1, 2, q, 2q\}. \tag{2.202}$$

Note that $o_r(n) = 1$ or 2 can not meet COND 1 and COND 3.
Now consider the number of r's such that $o_r(n) = 1$ or 2. In this case,

$$n \equiv 1 \pmod{r} \text{ or } n^2 \equiv 1 \pmod{r} \tag{2.203}$$
$$\Rightarrow n-1 \equiv 0 \pmod{r} \text{ or } n^2 - 1 = (n-1)(n+1) \equiv 0 \pmod{r} \tag{2.204}$$

Hence r must be a prime factor of $n^2 - 1$. Thus the number of such r's is at most

$$\log_2(n^2-1) \tag{2.205}$$
$$\leq \log_2 n^2 \tag{2.206}$$
$$= 2\log_2 n \tag{2.207}$$

Let us leave aside these prime factors of $n^2 - 1$ and consider the other co-Sophie Germain primes r for which $o_r(n) = q = \frac{r-1}{2}$ or $o_r(n) = 2q = r - 1$. They meet COND 3. And in

order to meet COND 1, the sufficient condition is

$$\frac{r-1}{2} \geq 4\sqrt{r}\log_2 n \tag{2.208}$$
$$\Rightarrow r-1 \geq 8\sqrt{r}\log_2 n \tag{2.209}$$
$$\Rightarrow r \geq 8\sqrt{r}\log_2 n \tag{2.210}$$
$$\Rightarrow \sqrt{r} \geq 8\log_2 n \tag{2.211}$$
$$\Rightarrow r \geq 64(\log_2 n)^2 \tag{2.212}$$
$$\tag{2.213}$$

Hence we consider the range $64(\log_2 n)^2$ to $c_2(\log_n 2)^2$ and we show that by making c_2 large enough, we find at least one desired r in this range.
By the conjecture 5, The number of co-Sophie Germain primes:

$$\text{less than } c_2(\log_2 n)^2 = \frac{Dc_2(\log_2 n)^2}{(\log_e c_2(\log_2 n)^2)^2} \tag{2.214}$$
$$\text{less than } 64(\log_2 n)^2 = \frac{D64(\log_2 n)^2}{(\log_e 64(\log_2 n)^2)^2} \tag{2.215}$$
$$\tag{2.216}$$

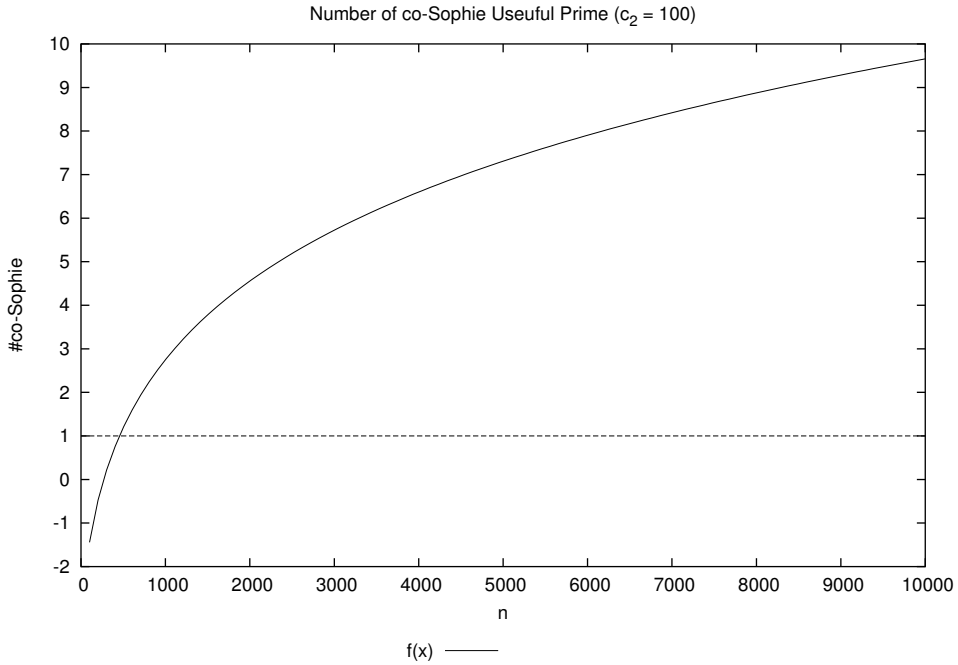
By (2.207), the number of co-Sophie Germain primes such that $o_r(n) = 1$ or 2:

$$\leq 2\log_2 n \tag{2.217}$$

Thus we will choose c_2 such that

$$(2.214) - (2.215) - (2.217) \geq 1 \tag{2.218}$$
$$\Leftrightarrow f(x) = \frac{Dc_2(\log_2 n)^2}{(\log_e c_2(\log_2 n)^2)^2} - \frac{D64(\log_2 n)^2}{(\log_e 64(\log_2 n)^2)^2} - 2\log_2 n \geq 1 \tag{2.219}$$
$$\tag{2.220}$$

The graph of $f(x)$ with $c_2 = 100$ is shown next.



Thus, if $c_2 \geq 100$, there exists "suitable" r in $[64(\log_2 n)^2, c_2(\log_2 n)^2]$ for all n that is large enough. □

The time complexity under the conjecture 5

The while-loop makes $O(\log_2 n)$ iterations. Hence total time of while-loop, (2.188), is changed to

$$
\begin{aligned}
& O((\log_2 n)^2 r^{\frac{1}{2}} \mathrm{poly}(\log_2 \log_2 n)) && (2.221)\\
=& O((\log_2 n)^3 \mathrm{poly}(\log_2 \log_2 n)) && (2.222)\\
=& O((\log_2 n)^3 \mathrm{poly}(3 \log_2 \log_2 n)) && (2.223)\\
=& O((\log_2 n)^3 \mathrm{poly}(\log_2 (\log_2 n)^3)) && (2.224)\\
=& \tilde{O}((\log_2 n)^3) && (2.225)
\end{aligned}
$$

In the same way, the time complexity of the for-loop, (2.194), is changed to

$$
\begin{aligned}
& O(r^{\frac{3}{2}} (\log_2 n)^3 \mathrm{poly}(\log_2 (r(\log_2 n)^2)) && (2.226)\\
=& O((\log_2 n)^6 \mathrm{poly}(\log_2 (\log_2 n)^4)) && (2.227)\\
=& O((\log_2 n)^6 \mathrm{poly}(\log_2 (\log_2 n)^6)) && (2.228)\\
=& \tilde{O}((\log_2 n)^6) && (2.229)
\end{aligned}
$$

Therefore the total time complexity for the algorithm is $\tilde{O}((\log_2 n)^6)$.

2.7 Future Work and Improvements

[6 Future Work and Improvements] (page 8)

> In our algorithm, the for-loop needs to run for $1 \leq a \leq 2\sqrt{r}\log_2 n$ in order to ensure that the size of the group G referred to in Lemma 4.4 is large enough ($> n^{2\sqrt{r}}$). The upper limit for a could be improved if we can show that a still smaller set of $(x-a)$'s generates a group of the required size. This seems very likely.

In later section 3.3, we see that some results of experiments support this claim.

> We can further improve the complexity to $\tilde{O}((\log_2 n)^3)$ if we can prove the following conjecture given in [4] and verified for $r \leq 100$ and $n \leq 10^{10}$ in [11]:
>
> **Conjecture 6. (page 8)** *If r does not divide n and if*
>
> $$(x-1)^n \equiv (x^n - 1) \pmod{x^r - 1, n}, \tag{2.230}$$
>
> *then either n is prime or $n^2 \equiv 1 \pmod{r}$.*
>
> If this conjecture is true, we can modify the algorithm slightly to first search for an r which does not divide $n^2 - 1$. Such an r can assuredly be found in the range $[2,\ 4\log_2 n]$. This is because the product of prime numbers less than x is at least e^x (see [2]).($\star$) Thereafter we can test whether the congruence (2.230) holds or not. Verifying congruence (2.230) takes time $\tilde{O}(r(\log_2 n)^2)$ using FFT for multiplication. This would give us a time complexity of $\tilde{O}(\log_2 n)^3$.

($\star$) This claim of the original paper is incorrect. The fact is that the product of prime numbers less than x is **at most** e^x. This product is called primorial. (See the pertinent section in [21].)

prime	primorial $p_k\# = p_{k-1}\# \times p_k$	e^x
$p_1 = 2$	$p_1\# = 2$	7
$p_2 = 3$	$p_2\# = 6$	20
$p_3 = 5$	$p_3\# = 30$	148
$p_4 = 7$	$p_4\# = 210$	1097
$p_5 = 11$	$p_5\# = 2310$	59874
$p_6 = 13$	$p_6\# = 30030$	442413
$p_7 = 17$	$p_7\# = 510510$	24154953
$p_8 = 19$	$p_8\# = 9699690$	178482301
$p_9 = 23$	$p_9\# = 223092870$	9744803446
$p_{10} = 29$	$p_{10}\# = 6469693230$	3931334297144
$p_{11} = 31$	$p_{11}\# = 200560490130$	29048849665247

Chapter 3

Analyzing the AKS algorithm

3.1 Finding Useful Primes

In this part of the project we actually try to find the (smallest) useful prime r for each given positive integer n. Some simple programs are made and run for this purpose.

If n is large enough, presence of a useful prime r for the n is certified by Lemma 4.2. In other words, if n is small, such a useful prime r may not be found. The real lower bound of n to have a useful prime is experimentally examined.

And then if n is large enough, Lemma 4.2. also assures that r exists in the range of less than $c_2(\log_2 n)^6$ (c_2 is a constant). However a useful prime may exist for every n in a much smaller range than this theoretically assured range. How small of a range can be expected? Consider COND 1 to see this.

$$\text{COND 1: } q \geq 4\sqrt{r}\log_2 n \tag{3.1}$$

$$\Rightarrow \frac{q}{4\sqrt{r}} \geq \log_2 n \tag{3.2}$$

This implies a larger q, which is the largest prime factor of $r-1$ ($= P(r-1)$), has more of an advantage. A possible maximum value of q is $(r-1)/2$ when $(r-1)/2$ is a Sophie Germain prime (or r is a co-Sophie Germain prime). When r is large, $(r-1)/2 \simeq r/2$. By substituting this for (3.2)

$$\frac{r/2}{4\sqrt{r}} \geq \log_2 n \tag{3.3}$$

$$\Rightarrow r \geq 64(\log_2 n)^2 \tag{3.4}$$

Thus $64(\log_2 n)^2$ is the smallest size of the useful prime that can be expected. More generally when $q = O(r)$ and the q also satisfies COND 2, by (3.2), the expected size of r is:

$$q \simeq c(\log_2 n)^2 \text{ [c is a constant]}. \tag{3.5}$$

3.1.1 The Algorithm

Direct implementation of some parts of original AKS algorithm is not an efficient way to calculate useful primes for a sequence of many n. To handle this situation, we consider the following inequality modified COND 1.

$$\text{COND 1: } q \geq 4\sqrt{r}\log_2 n \tag{3.6}$$

$$\Rightarrow \frac{q}{4\sqrt{r}} \geq \log_2 n \tag{3.7}$$

$$\Rightarrow 2^{q/4\sqrt{r}} \geq n \tag{3.8}$$

By denoting $e = q/4\sqrt{r}, max\_n = 2^e$

$$max\_n \geq n \tag{3.9}$$

Thereby max_n represents the upper bound of n which r can apply to. Hence we only need to check r's such that $max\_n \geq n$ in order to find a useful prime for the given n. By using this we can improve the performance of our process to find useful primes. The specific algorithm is outlined below.

1. **Create a Table**
   Starting from 2 (the smallest prime) consecutively calculate max_n of each prime r and create the table of the following format. However, if $max\_n \leq r$, it is not output to the table, because it is not appropriate for a useful prime. (NOTE: The table is in increasing order of r.)

r	max_n

2. **Find Useful Primes**
   While incrementing n from an initial value to an upper bound, try to find the smallest useful prime r for each n.

   (a) Skip lines of the table until $max\_n \geq n$. (Label the line number of the table at this point as k.)

   (b) Read the table from k until satisfying the following condition.

   if $max\_n \geq n$ and COND 2 ($n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$)
   then output r as the smallest useful prime for n
   (If no one satisfies the conditions until end of the table, output "Not Found".)

   (c) Increment n and go back (a). (Restart reading the table from the line k.)

3.1.2 Experiments and Results

Lower bound of n Firstly, see the result from relatively small n (Table 3.1). For $n \leq 11700$ and $11702 \leq n \leq 11808$, useful primes were not found. Useful primes were found for n greater than 11808. In later experiments useful primes were always found in the range of greater than 11808. This implies the lower bound of n is 11809 so that every n over it has a useful prime constantly.

n (Range) From	∼	To	Useful Prime for Range
0	∼	11700	not exist
11701	∼	11701	11699
11702	∼	11808	not exist
11809	∼	12256	11807
12257	∼	13803	12107
13804	∼	14334	12203
14335	∼	14469	12227
14470	∼	14675	12263
14676	∼	15164	12347
15165	∼	16262	12527
≀			≀

Table 3.1: Useful Primes for Small n

$11809 < n < 57070000000$ Secondly, we calculate useful primes for all n of $11809 < n < 57070000000$. ($\log_2 57070000000 \simeq 35.73$) Since we are interested in the size of useful primes relative to n, we compute and observe the value of $r/(\log_2 n)^2$. The following result was obtained.

$$64.0015 \leq \frac{r}{(\log_2 n)^2} \leq 64.0109 \tag{3.10}$$

The maximum size 64.0109 was obtained for $n = 11809$. Then the size of $r/(\log_2 n)^2$ tends to close in on 64 according to the increase in the size of n. This result means these useful primes are the smallest ones indicated by the formula (3.4). In fact it was also ascertained that all of them were co-Sophie Germain primes.

For Larger n It is difficult to find useful primes for n's larger than 57070000000 since it takes too much processing time. Accordingly we pick up just some n's by the rule below and find useful primes for them.

$$n_0 = 57070000000 \tag{3.11}$$

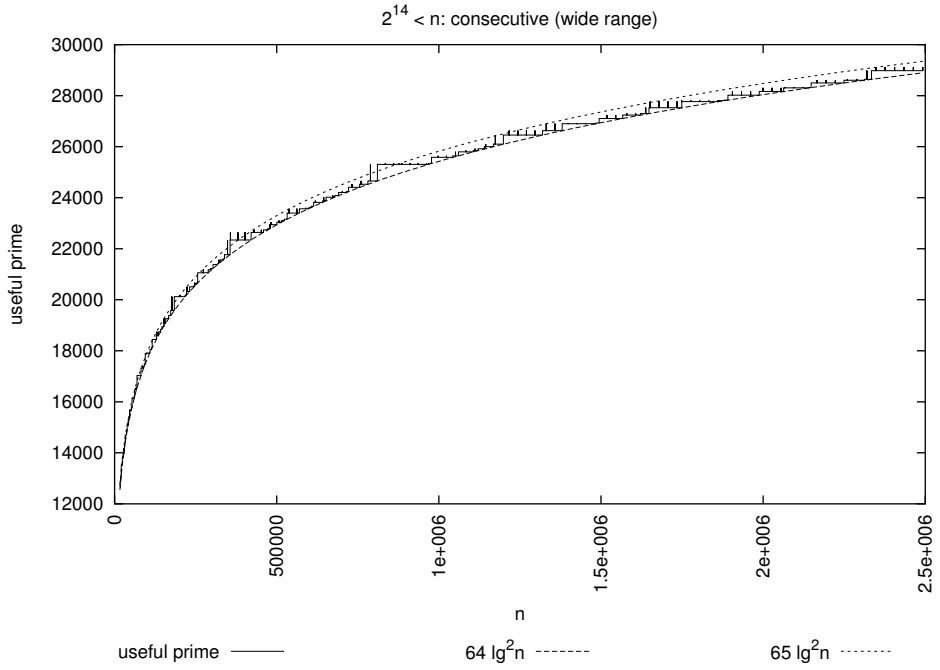
$$n_{k+1} = \lfloor n_k \times 1.001 \rfloor \quad (\text{while } n_k < 2^{63}) \tag{3.12}$$

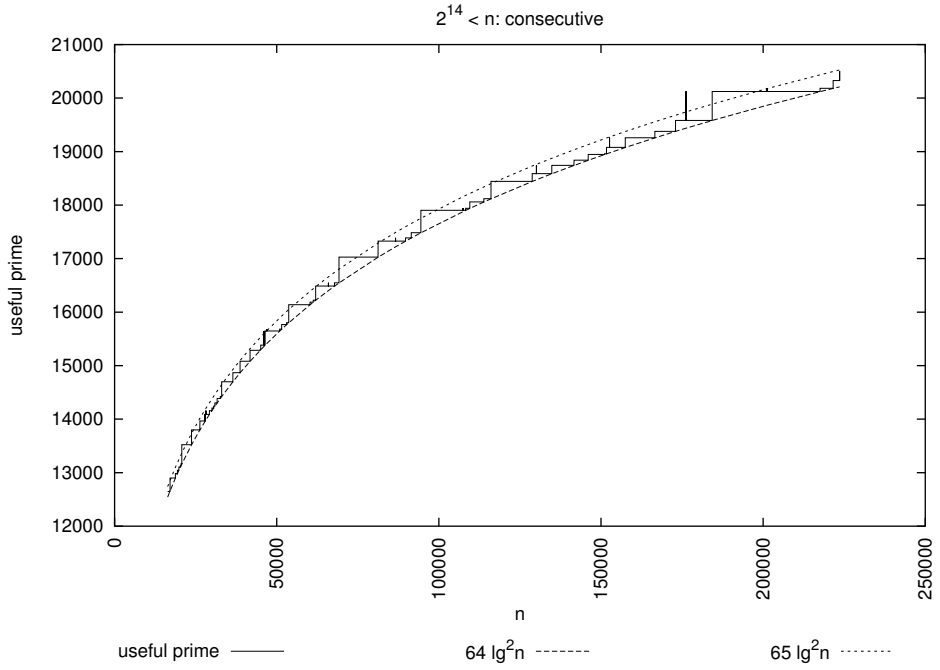
The following result was obtained for $0 \leq k \leq 18910$. ($n_{18910} = 9221742743576376320$, $\log_2 n_{18910} \simeq 63$)

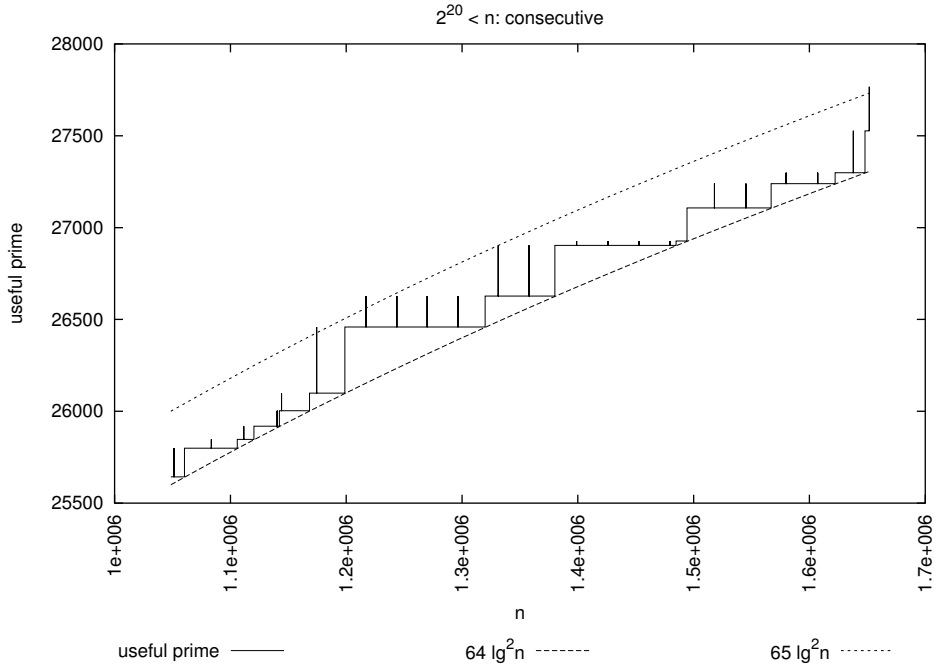
$$64.0005 \leq \frac{r}{(\log_2 n)^2} \leq 64.0016 \tag{3.13}$$

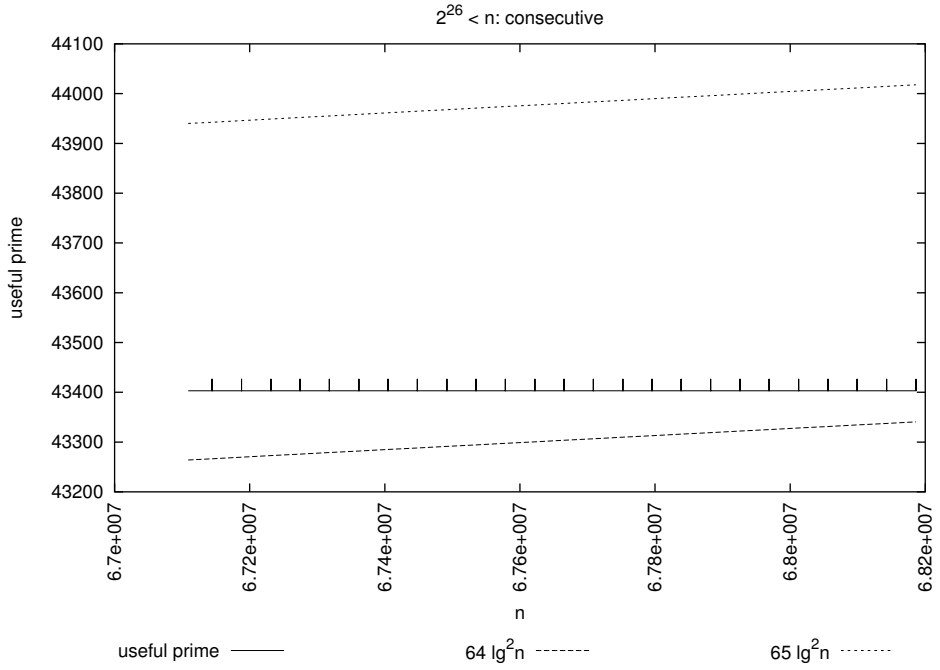
Again all useful primes were co-Sophie Germain primes.

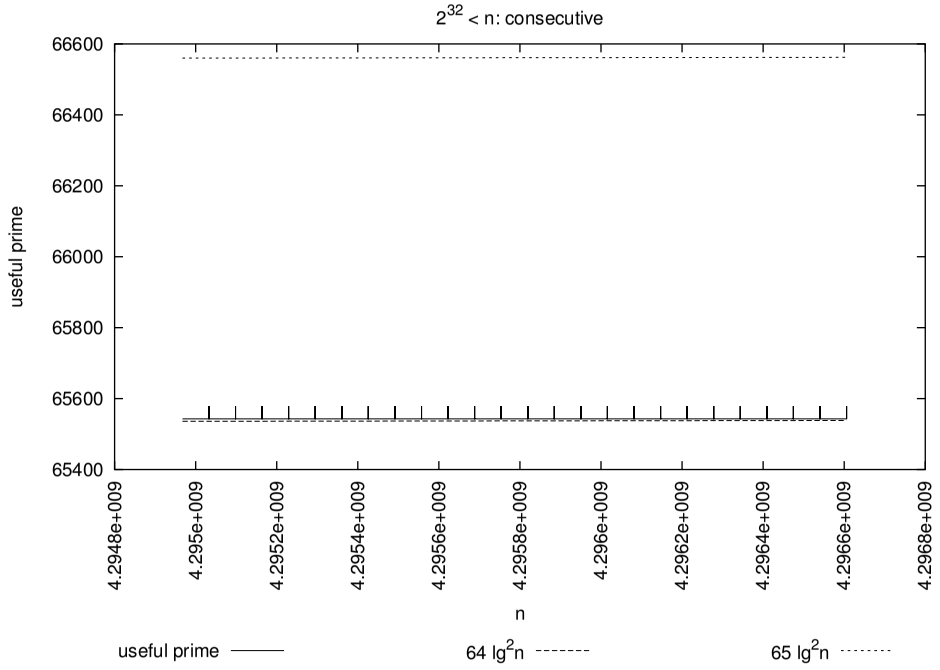
Graphs of Useful Primes Some of the results we have obtained so far are shown in the following graphs. Each graph corresponds to a different size of n and plots values of useful primes for some sequential n's. The curves of $64(\log_2 n)^2$ and $65(\log_2 n)^2$ are also drawn in each graph to measure the size of useful primes.

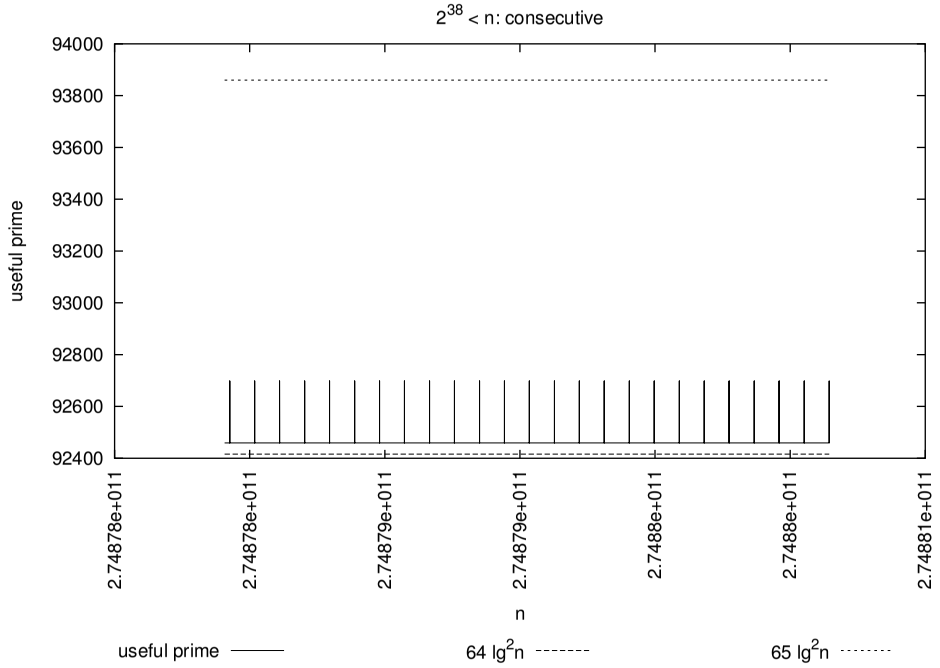


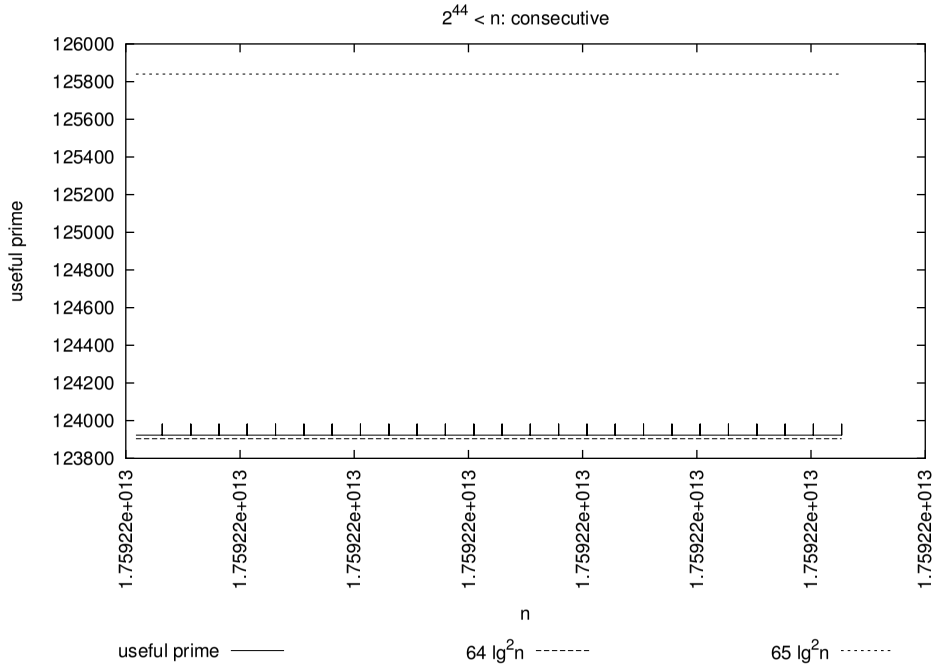


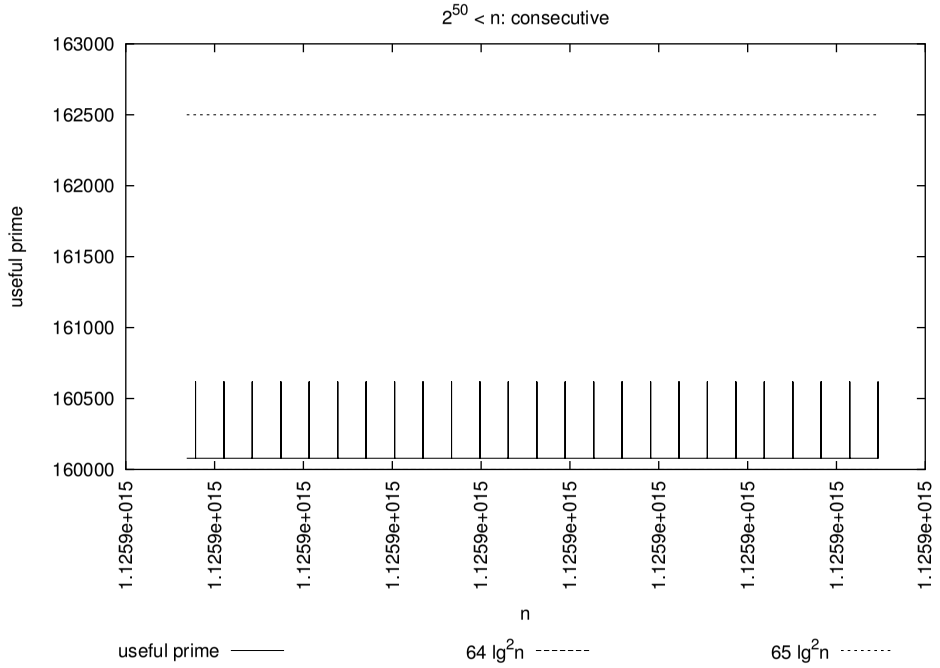

2^{20} < n: consecutive
useful prime
n
28000
27500
27000
26500
26000
25500
1e+006
1.1e+006
1.2e+006
1.3e+006
1.4e+006
1.5e+006
1.6e+006
1.7e+006
useful prime
64 lg^{2}n
65 lg^{2}n

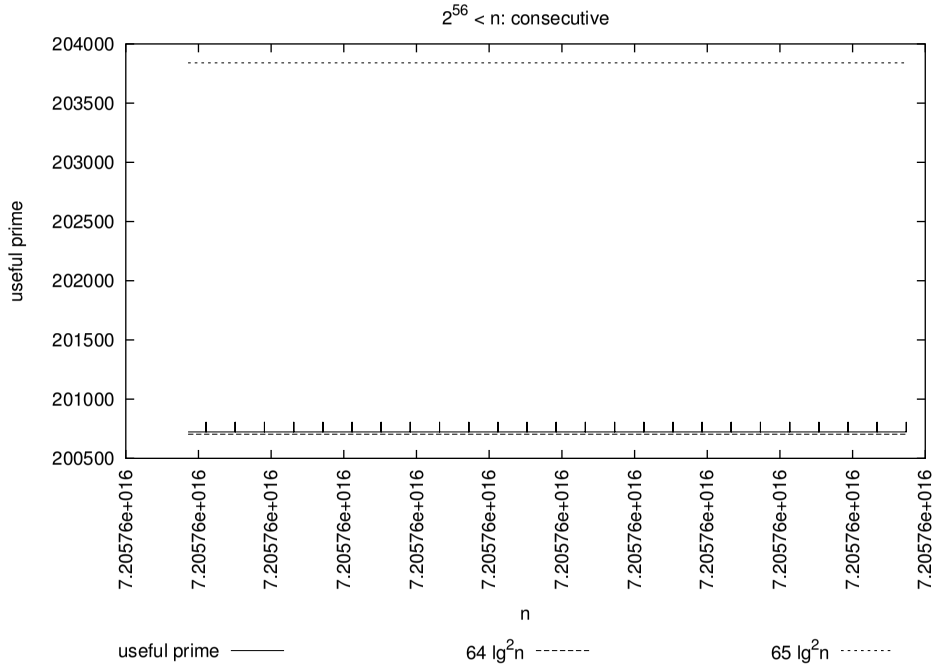

2^{26} < n: consecutive
useful prime
n
44100
44000
43900
43800
43700
43600
43500
43400
43300
43200
6.7e+007
6.72e+007
6.74e+007
6.76e+007
6.78e+007
6.8e+007
6.82e+007
useful prime
64 lg^{2}n
65 lg^{2}n

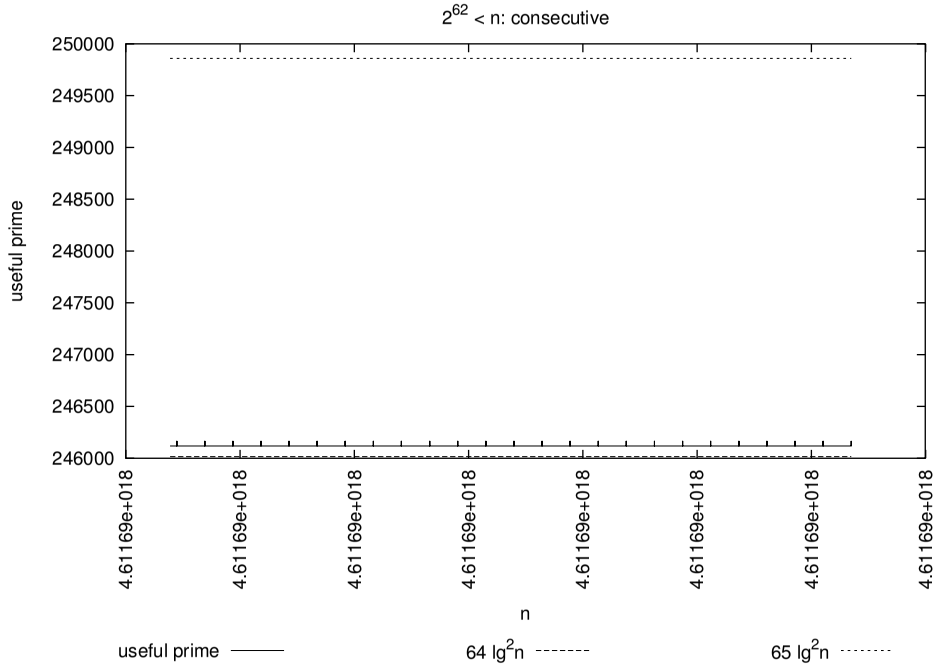

$2^{32} < n$: consecutive
66600
66400
66200
66000
65800
65600
65400
useful prime
4.2948e+009
4.295e+009
4.2952e+009
4.2954e+009
4.2956e+009
4.2958e+009
4.296e+009
4.2962e+009
4.2964e+009
4.2966e+009
4.2968e+009
n
useful prime
64 lg^{2}n
65 lg^{2}n


$2^{38} < n$: consecutive
94000
93800
93600
93400
93200
93000
92800
92600
92400
useful prime
2.74878e+011
2.74878e+011
2.74879e+011
2.74879e+011
2.7488e+011
2.7488e+011
2.74881e+011
n
useful prime
64 lg^{2}n
65 lg^{2}n


2^{44} < n: consecutive
126000
125800
125600
125400
125200
125000
124800
124600
124400
124200
124000
123800
useful prime
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
n
useful prime
64 lg^{2}n
65 lg^{2}n


2^{50} < n: consecutive
163000
162500
162000
161500
161000
160500
160000
useful prime
1.1259e+015
1.1259e+015
1.1259e+015
1.1259e+015
1.1259e+015
1.1259e+015
1.1259e+015
1.1259e+015
n
useful prime
64 lg^{2}n
65 lg^{2}n





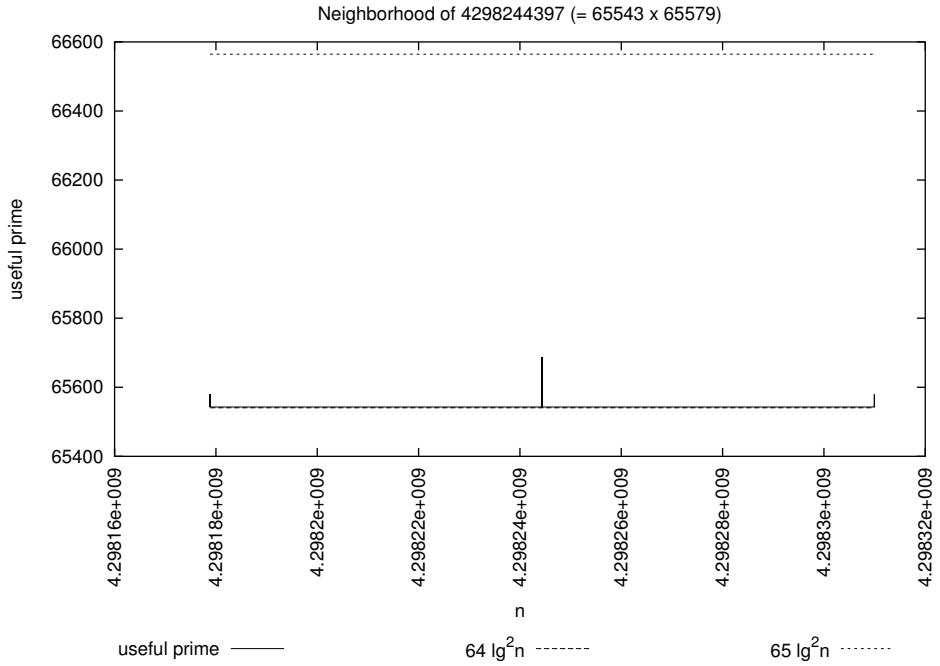
What is $\perp$? The shape of $\perp$ often appears in the previous graphs. What does it mean? The next table shows the detail of each value as it pertains to one of the graphs.

n			r	n	$\frac{65543-1}{q}$	$n^{\frac{r-1}{q}}$	$\perp$
From	To	To -From		(mod 65543)	$(q = P(65543-1) = 32771)$	(mod 65543)	
4295032792	4295098331	65539	65543	2	2	4	
4295098332	4295098332	0	65579	-1	2	1	$\perp$
4295098333	4295098333	0	65543	0	2	0	
4295098334	4295098334	0	65579	1	2	1	$\perp$
4295098335	4295163874	65539	65543	2	2	4	
4295163875	4295163875	0	65579	-1	2	1	$\perp$
4295163876	4295163876	0	65543	0	2	0	
4295163877	4295163877	0	65579	1	2	1	$\perp$
4295163878	4295229417	65539	65543	2	2	4	
4295229418	4295229418	0	65579	-1	2	1	$\perp$
4295229419	4295229419	0	65543	0	2	0	
4295229420	4295229420	0	65579	1	2	1	$\perp$
4295229421	4295294960	65539	65543	2	2	4	

The rows with $\perp$ indicate the values of n, when $\perp$ appears in the graph. In those rows $n = 65543k \pm 1$, then $n^{\frac{65543-1}{q}} = n^2 \equiv 1 \pmod{65543}$. Thus co-Sophie Germain prime 65543 is not adopted as a useful prime. Then the co-Sophie Germain prime next to it, 65579, is adopted. From this observation we can expect that $n = k \times 65543 \times 65579 \pm 1 = k4298244397 \pm 1$ have neither 65543 nor 65579 as a useful prime. The result of the experiment when $k = 1$ is the following:

	n			r	n	n	$\perp$	
	From	To	To -From		(mod 65543)	(mod 65579)		
	4298178847	4298178852	5	65543	65536	34		
	4298178853	4298178853	0	65579	-1	35	$\perp$	
	4298178854	4298178854	0	65543	0	36		
	4298178855	4298178855	0	65579	1	37	$\perp$	
	4298178856	4298244395	65539	65543	2	65577		
$65543 \times 65579 - 1 =$	4298244396	4298244396	0	65687	-1	-1	$\perp$	$\star$
$65543 \times 65579 =$	4298244397	4298244397	0	65543	0	0		
$65543 \times 65579 + 1 =$	4298244398	4298244398	0	65687	1	1	$\perp$	$\star$
	4298244399	4298309938	65539	65543	2	65541		
	4298309939	4298309939	0	65579	-1	65542	$\perp$	

When $n = 65543 \times 65579 \pm 1$, the third co-Sophie Germain prime 65687 is chosen as a useful prime other than 65543 and 65579. The next graph plots the data of the table above.



If co-Sophie Germain primes are not used All useful primes we have found in previous experiments were co-Sophie Germain primes. Next we observe what kind of useful primes are found when co-Sophie Germain primes are not adopted as useful primes. The next condition is added to the program.

$$r \neq 2q + 1 \tag{3.14}$$

We searched useful primes for some different order of n. ($n \approx 2^{16}, 2^{20}, 2^{26}, 2^{32}, 2^{38}, 2^{44}, 2^{50}, 2^{56}, 2^{62}$) Firstly, the lower bound of n to have a useful prime was changed by the additional condition. The revised lower bound is $n = 65791$ with $r = 65789$. Here $r - 1 = 65788 = 4 \times 16447$ (16447 is prime), then r is not co-Sophie Germain prime. Secondly the following property about size of useful primes was observed for all cases.

$$r \simeq 256(\log_2 n)^2 \tag{3.15}$$

This indicates $r = 4q + 1$, because, by replacing $r/2$ in formula (3.3) with $r/4$, we got:

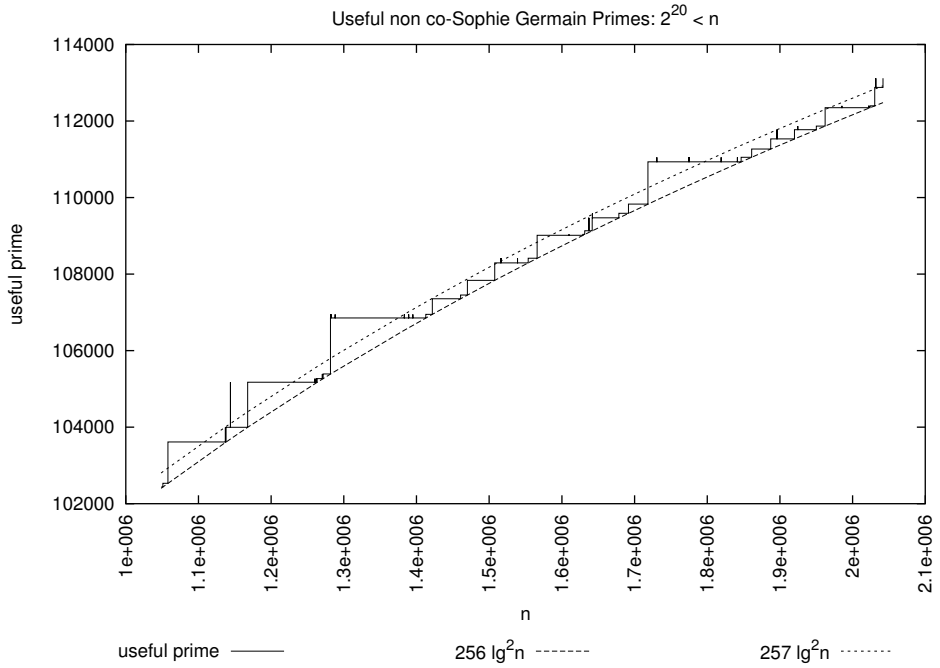
$$\frac{r/4}{4\sqrt{r}} \geq \log_2 n \tag{3.16}$$

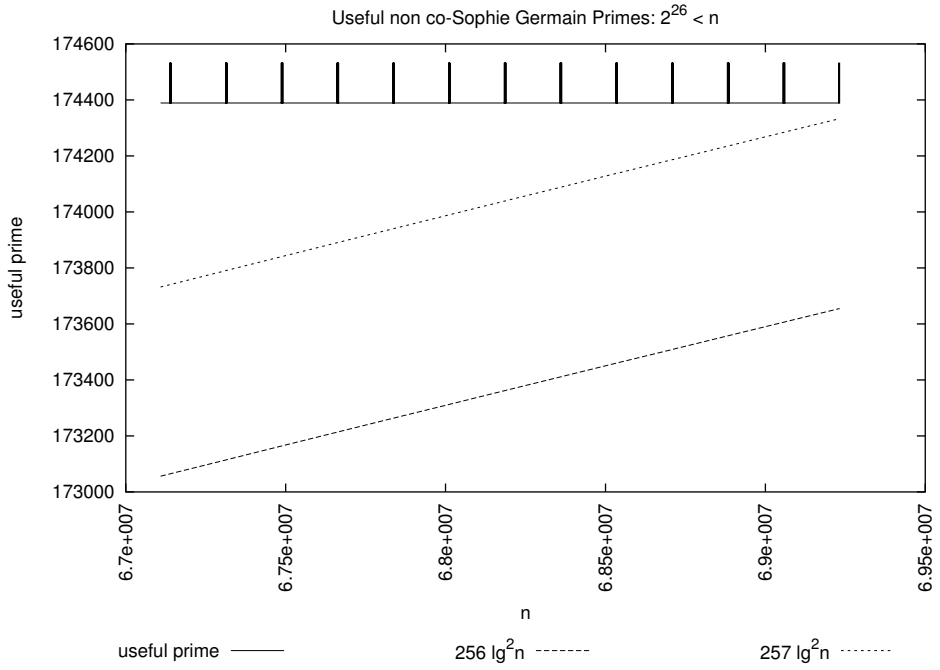
$$\Rightarrow \frac{\sqrt{r}}{16} \geq \log_2 n \tag{3.17}$$

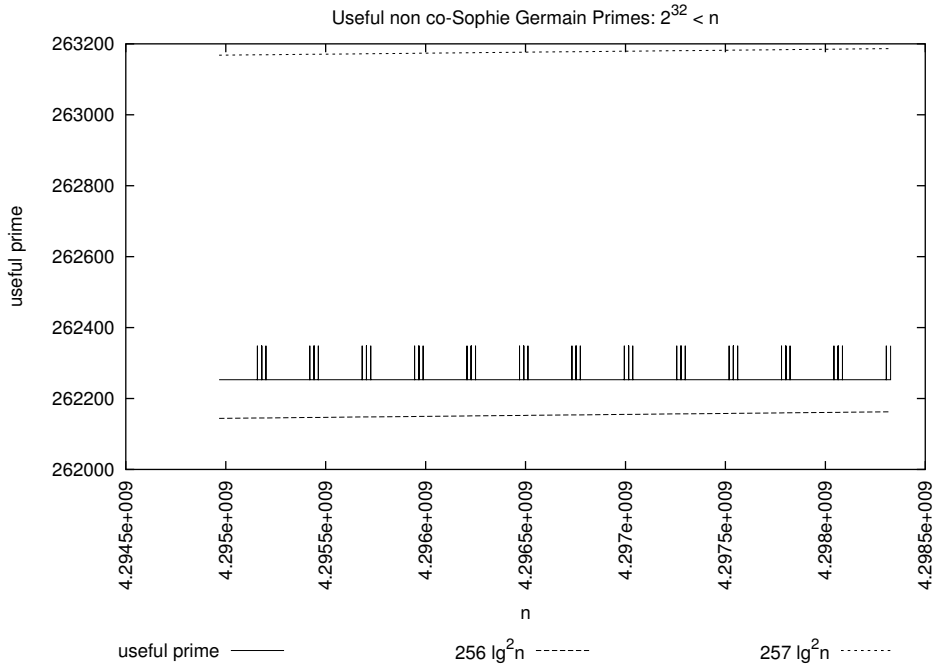
$$\Rightarrow r > 256(\log_2 n)^2. \tag{3.18}$$

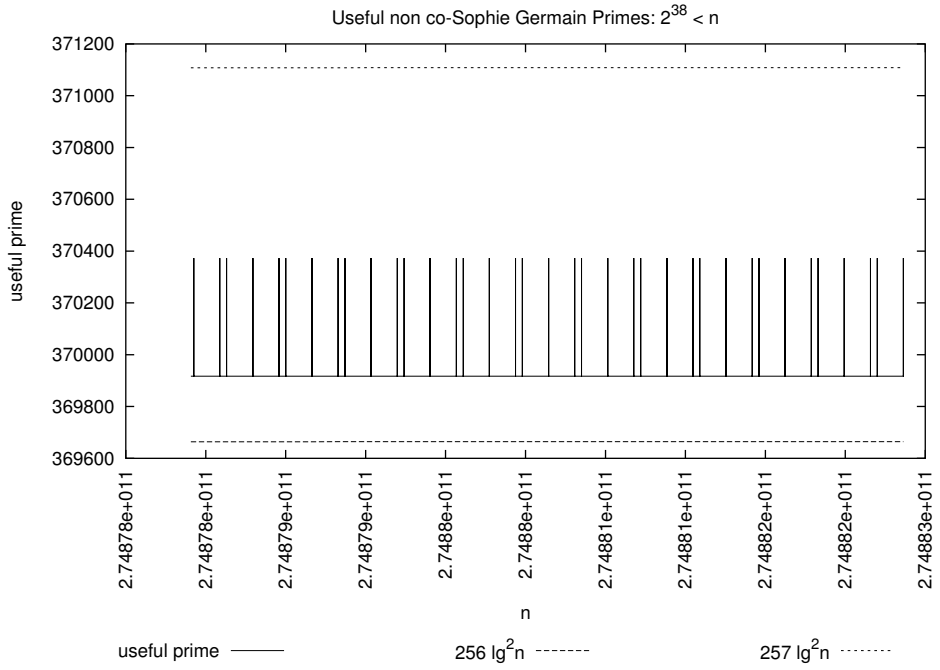
Why is it not $r = 3q + 1$? Since $r - 1$ is even, it must have 2 as a factor. Thus $r = 4q + 1$ is the second-best type other than co-Sophie Germain prime. The results are shown in the following graphs:

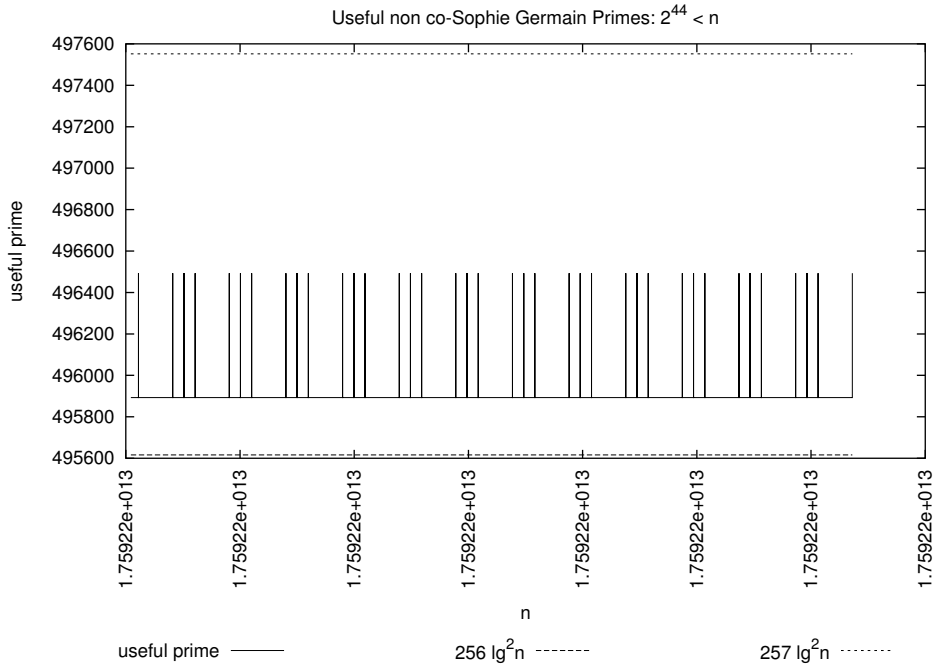

Useful non co-Sophie Germain Primes: $2^{16} < n$
useful prime
n
90000
85000
80000
75000
70000
65000
50000
100000
150000
200000
250000
300000
350000
useful prime
256 lg^{2}n
257 lg^{2}n

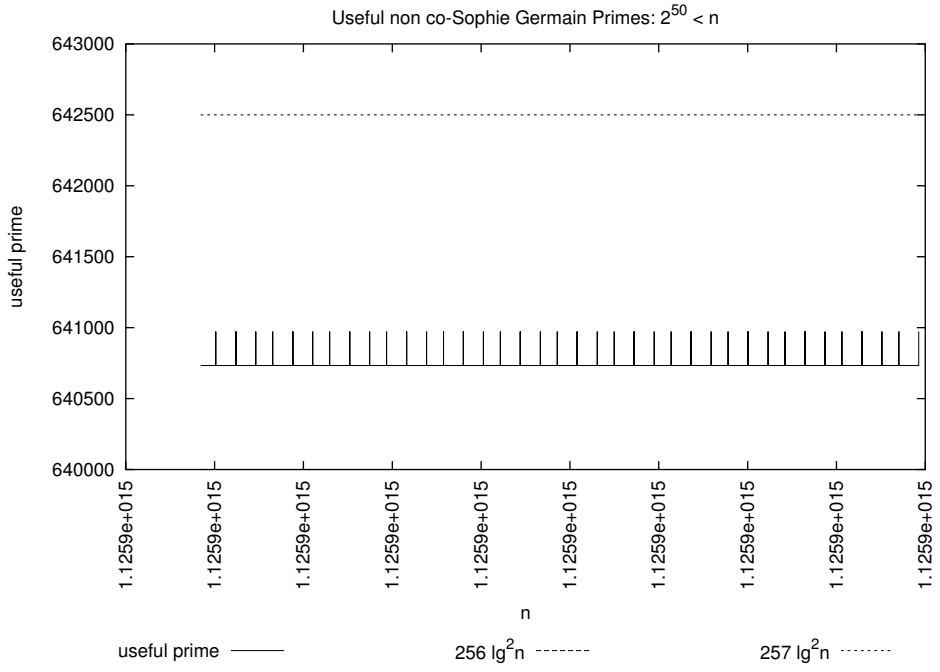

Useful non co-Sophie Germain Primes: $2^{20} < n$
useful prime
n
114000
112000
110000
108000
106000
104000
102000
1e+006
1.1e+006
1.2e+006
1.3e+006
1.4e+006
1.5e+006
1.6e+006
1.7e+006
1.8e+006
1.9e+006
2e+006
2.1e+006
useful prime
256 lg^{2}n
257 lg^{2}n

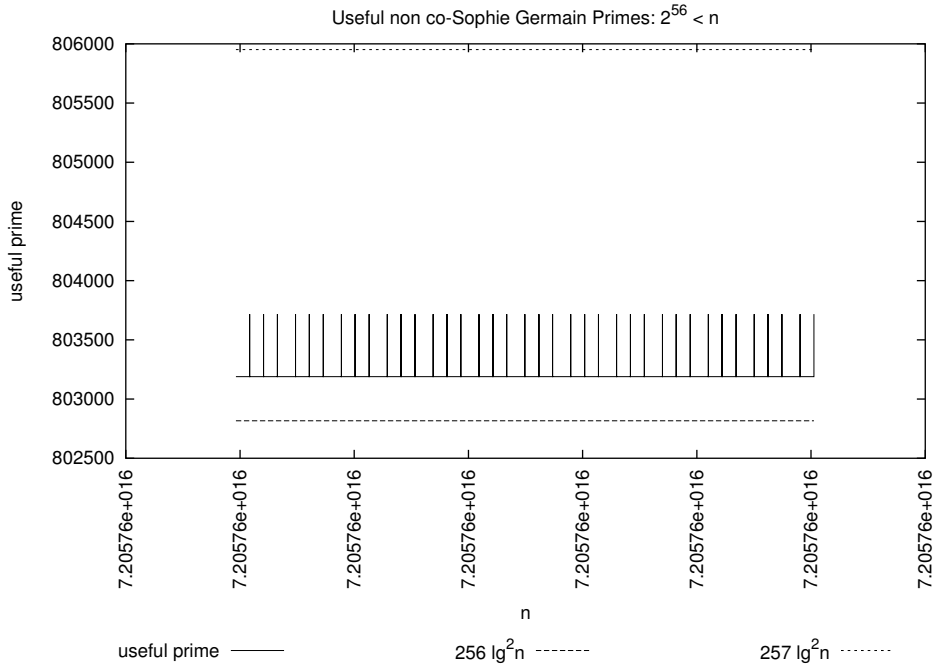

Useful non co-Sophie Germain Primes: $2^{26} < n$
useful prime
174600
174400
174200
174000
173800
173600
173400
173200
173000
6.7e+007
6.75e+007
6.8e+007
6.85e+007
6.9e+007
6.95e+007
n
useful prime
256 lg^{2}n
257 lg^{2}n

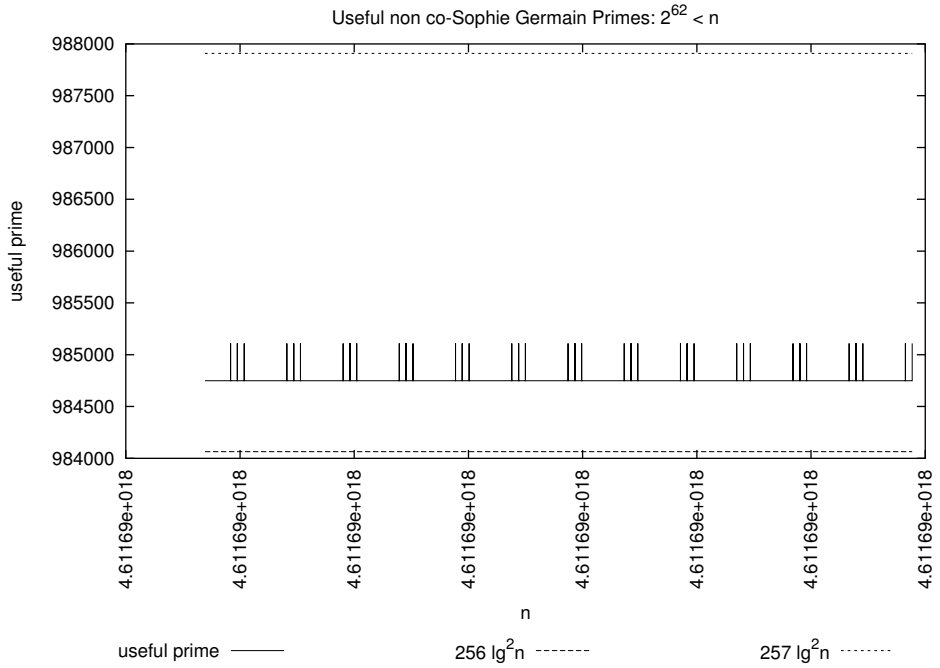

Useful non co-Sophie Germain Primes: $2^{32} < n$
useful prime
263200
263000
262800
262600
262400
262200
262000
4.2945e+009
4.295e+009
4.2955e+009
4.296e+009
4.2965e+009
4.297e+009
4.2975e+009
4.298e+009
4.2985e+009
n
useful prime
256 lg^{2}n
257 lg^{2}n


Useful non co-Sophie Germain Primes: $2^{38} < n$
useful prime
371200
371000
370800
370600
370400
370200
370000
369800
369600
2.74878e+011
2.74878e+011
2.74879e+011
2.74879e+011
2.7488e+011
2.7488e+011
2.74881e+011
2.74881e+011
2.74882e+011
2.74882e+011
2.74883e+011
n
useful prime
256 lg^{2}n
257 lg^{2}n


Useful non co-Sophie Germain Primes: $2^{44} < n$
useful prime
497600
497400
497200
497000
496800
496600
496400
496200
496000
495800
495600
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
1.75922e+013
n
useful prime
256 lg^{2}n
257 lg^{2}n


Useful non co-Sophie Germain Primes: $2^{50} < n$
useful prime
643000
642500
642000
641500
641000
640500
640000
1.1259e+015
n
useful prime
256 lg^{2}n
257 lg^{2}n


Useful non co-Sophie Germain Primes: $2^{56} < n$
useful prime
806000
805500
805000
804500
804000
803500
803000
802500
7.20576e+016
n
useful prime
256 lg^{2}n
257 lg^{2}n



The Lower Bound on Composite n Detected with a Useful Prime From former experiments, the lower bound of n to have a useful prime always is 11809 with the useful prime 11807. (See Table 3.1.) 11809 ($= 7 \times 7 \times 241$) is incidentally composite. However, 11807 is not actually used to detect 11809 as composite in the running of the AKS algorithm, because there is the check of $\gcd(n, r) \neq 1$ in the while-loop of the AKS algorithm. 11809 is detected by this check before the useful prime 11807 is found, since 11809 has a factor of $7 (< 11807)$. Thus, if 11807 is applied for a composite n, the n must be greater than $139405249 (= 11807^2)$, since all prime factors of such n must be greater than 11807. Then, we try to find the lower bound of composite n in this context. The algorithm, a little changed (adding the GCD check etc.) from the previous one, is used for this purpose. It is reasonable to expect that the composite number that gives such lower bound is of the form p^2 (p is prime). Then, we checked about this type of composite numbers and obtained the following result.

$n (= \text{prime}^2)$	r (Useful Prime)
$4296933601 (= 65551^2)$	65543
$4300867561 (= 65581^2)$	65579
$4326613729 (= 65777^2)$	65687
$4405375129 (= 66373^2)$	65867
$4541546881 (= 67379^2)$	66047

From the table above, the lower bound is 4296933601. This is 65551 squared and 65551 is the smallest prime factor of 4296933601. 65551 is the smallest prime larger than 65543 that is a useful prime of 4296933601.

But again, is this really lower bound of n that is detected by a useful prime? We must recall the a^b check at Line 1 of the AKS algorithm. p^2 type numbers are detected by this check. Thus we had to check the $n = p_1 p_2$ type instead of p^2. (p_1 and p_2 are successive prime numbers.) The revised result is shown in the table below.

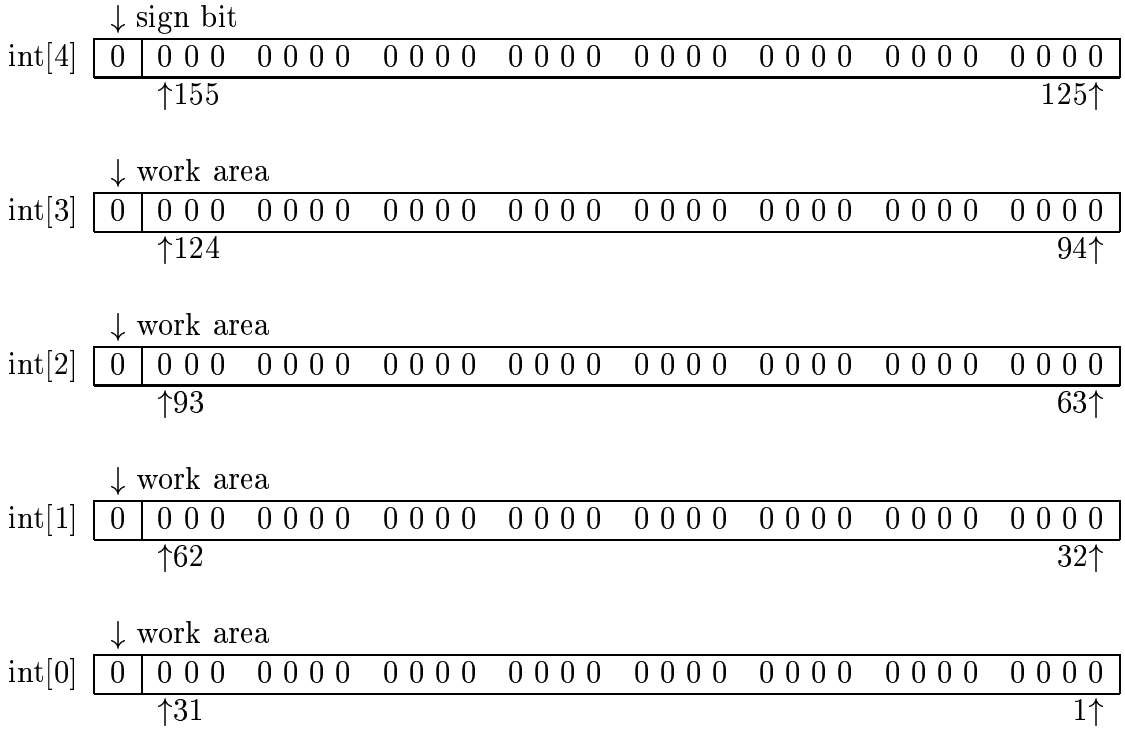
n					r (Useful Prime)
120337528573	=	346891	×	346903	346877
120488128771	=	347099	×	347129	347069
121339362019	=	348323	×	348353	347717
124268235253	=	352511	×	352523	348053
125808542989	=	354689	×	354701	348149
126256566251	=	355321	×	355331	349109
130783489591	=	361637	×	361643	349709
133687490653	=	365627	×	365639	350093
135580076719	=	368197	×	368227	350213
136174284299	=	369013	×	369023	350237

Finally we obtained the lower composite number 120337528573 (37 bits) with the useful prime 346877 that is smaller than 346898 that is the smallest prime factor of 120337528573.

3.2 Finding Useful Primes with Multiple Precision Integers

For Much Larger n The size of n we have dealt with so far was at most 2^{63}. Regular data types of C or Java cannot deal with sizes larger than this. Java has a BigInteger class that can handle arbitrary-precision integers. We once tried to use that class, but its performance was too bad. Then we decided to implement some C functions to be able to handle integers larger than 2^{63}.

The Expression of multi-precision integers A multi-precision integer is expressed with an array of k integers. Suppose the size of integer is 32 bits. The format of a multi-precision integer in case of $k = 5$ is shown below:



Negative integers are expressed by two's complement notation. It can hold integers between -2^{155} to $2^{155}-1$.

Basic Functions The following basic functions are implemented to deal with multiple precision integers.
MPI (multiple precision integer): X, Y

function	input	return
set: integer → MPI	int[0]∼int[4], sign(-/+)	X set from integers
left shift	X	X: left shifted with 1 bit
right shift	X	X: right shifted with 1 bit
count #bits of MPI	X	number of bits of X
addition	X, Y	X+Y
multiplication	X, Y	$X \times Y$
minus	X	$-X$
compare X to Y	X, Y	if $X < Y$: -1
		if $X = Y$: 0
		if $X > Y$: 1
modulo	X, Y	X (mod Y)
power	X, n (integer)	X^n
display MPI with binary system	X	
display MPI with decimal system	X	

The Program to Find Useful Primes The program to find useful primes is implemented by combining the basic functions above. Multi-precision integer is used for n in the program.

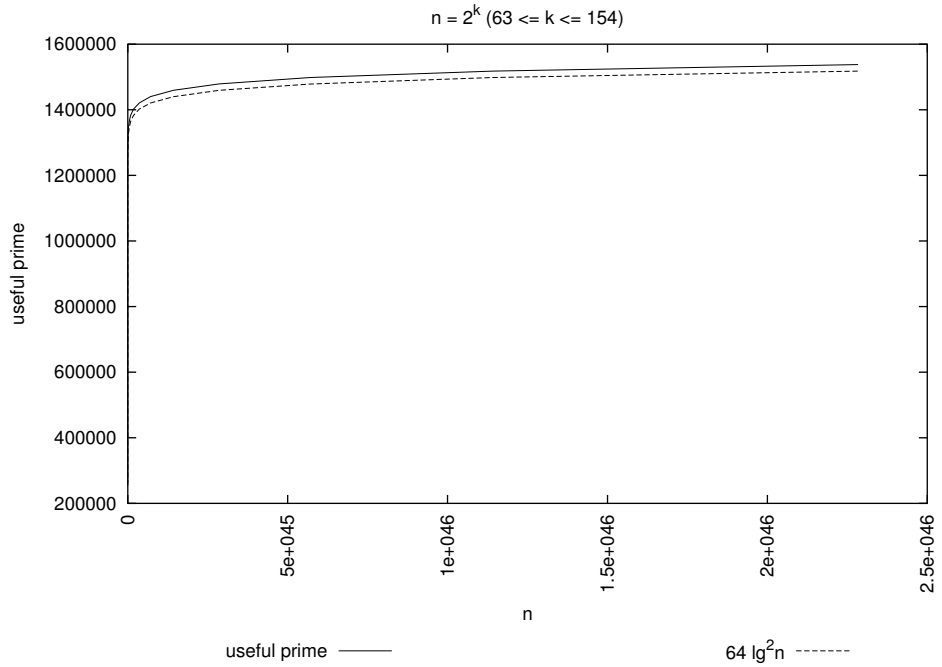
On the other hand, regular integer type is used for useful primes, since its expected size is much smaller than n. ($\simeq (\log_2 n)^2$) To avoid an implementation of logarithm calculation, the program adopts condition (3.20) instead of the original condition (3.19) of the AKS algorithm.

$$q \geq 4\sqrt{r}\log_2 n \tag{3.19}$$

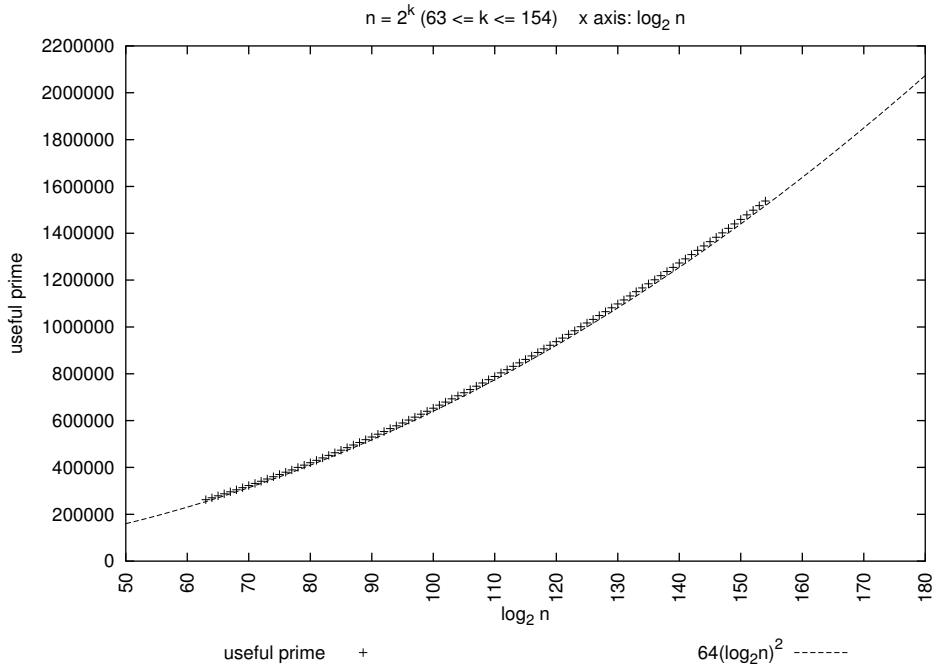
$$q \geq 4\sqrt{r}\lfloor \log_2 n + 1 \rfloor \tag{3.20}$$

Although condition (3.20) is a little more strict than condition (3.19), it is easy to implement since $\lfloor \log_2 n + 1 \rfloor$ is just the same as the number of bits of n.

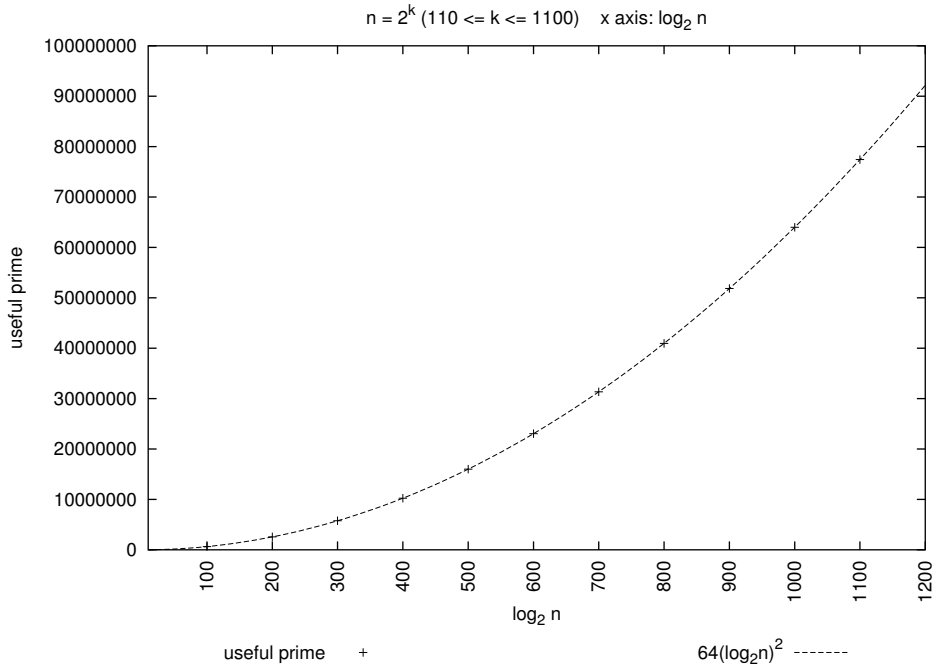
Useful Primes for Large n Useful primes were searched for $n = 2^k$ ($63 \leq k \leq 154$) by the program described above. The a^b check and the GCD check are excluded in this experiment. The results are shown in the following graph. The data table is shown in Appendix A.



Again, all of useful primes in this result are co-Sophie Germain prime and they are close to $64(\log_2 n)^2$. The graph whose x axis is $\log_2 n$ with same data is shown below:



Useful primes for much larger $n = 2^k$ $(100 \leq k \leq 1100)$ are shown in the next graph. Its data is also shown in Appendix A.



3.3 Implementation of the Congruence in the for-loop

We implement the congruence in the for-loop and conduct the primality test for some integers by using this program in order to check the correctness of the AKS algorithm. Relatively simple ways are used for the implementation instead of some sophisticated methods such as Fast Fourier Transform.

3.3.1 The Algorithm

- Congruence: $(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}$
- Input Parameters: n, a, r
- $\ell = \lfloor \log_2 n \rfloor$, $n = \sum_{i=0}^{\ell} b_i 2^i$
- LHS[r + 1]: coefficients of the left hand side of the congruence
- LxL[$2r$ - 1]: coefficients of LHS $\times$ LHS
- xa[2]: coefficients of $x - a$
  xa[0] $\leftarrow -a$
  xa[1] $\leftarrow 1$
- degLHS: degree of the left hand side of the congruence
- degLxL: degree of LxL
- degRHS: degree of the right hand side of the congruence

The algorithm consists of three parts.

1. Calculate LHS
2. Calculate RHS
3. Compare LHS and RHS
   if LHS $\not\equiv$ RHS return COMPOSITE
   otherwise return UNDECIDED

The details of the algorithm are as follows:

1. Calculate LHS by the square-and-multiply method.

```
LHS[j] ← 0 (for all j)
degLHS ← 0

for i = 0 to ℓ {

   LxL[j] ← 0 (for all j)

   /* LxL ← LHS × LHS */
   for k = 0 to degLHS
      for j = 0 to degLHS
         LxL[k+j] ← (LxL[k+j] + LHS[k] × LHS[j])  (mod n)
   degLxL ← degLHS × 2

   /* LxL  (mod x^r − 1) */
   if(degLxL ≥ r)
      for k = degLxL to r by -1
         LxL[k - r] ← (LxL[k - r] + LxL[k])  (mod n)
         LxL[k] ← 0
      degLxL ← r - 1;

   LHS[k] ← 0 (for all k)

   if(b_i = 1)
      /* LHS ← LxL ×(x − a) */
      for k = 0 to degLxL
         for int j = 0 to 1
            LHS[k+j] ← (LHS[k+j] + LxL[k] * xa[j])  (mod n)
      degLHS ← degLxL + 1;
   else
      /* LHS ← LxL */
      for k = 0 to degLxL
         LHS[k] ← LxL[k]
      degLHS ← degLxL;

   /* LHS  (mod x^r − 1) */
   if(degLHS ≥ r)
      for k = degLHS to r by -1
         LHS[k - r] ← (LHS[k - r] + LHS[k])  (mod n)
         LHS[k] ← 0
      degLHS ← r - 1
}
```

2. Calculate degRHS
See

$$\begin{aligned} \text{RHS} &= x^n - a && (3.21)\\ &= (x^r - 1)(x^{n-r} + x^{n-2r} + \cdots + x^{n-kr}) + x^{n-kr} - a && (3.22)\\ &\equiv x^{n-kr} - a \pmod{x^r - 1, n}. && (3.23) \end{aligned}$$

Here k is the smallest integer such that $n - kr < r$. Let degRHS $= n - kr$. Knowing degRHS is enough to decide what RHS is. The algorithm to calculate degRHS is specified below.

```
if (r > n)
   k ← 0
else
   k ← ⌊n/r⌋
degRHS ← n − kr
```

3. Compare LHS and RHS

The coefficients of RHS are:
- If degRHS $= 0$
  - zeroth degree: $1 - a$
  - otherwise: 0
- If degRHS $\neq 0$
  - zeroth degree: $-a$
  - degRHS-th degree: 1
  - otherwise: 0

Check whether LHS is equal above or not. The specific algorithm is following.

```
if (degRHS = 0)
   if (LHS[0] ≢ 1 − a (mod n))
   return COMPOSITE
else
   if (LHS[0] ≢ −a (mod n))
   return COMPOSITE
for i = 1 to r − 1
   if (i = degRHS and LHS[i] ≠ 1) return COMPOSITE
   else if (LHS[i] ≠ 0) return COMPOSITE
return UNDECIDED
```

3.3.2 Experiments and Results

Firstly, in order to verify that the program performs correctly, we run the program with basic parameter values. In particular, $r = n + 1$. By doing this $\pmod{x^r - 1}$ can be taken away from the congruence.

$$(x-a)^n \equiv (x^n - a) \pmod{x^{n+1}, n} \tag{3.24}$$

$$\Leftrightarrow (x-a)^n \equiv (x^n - a) \pmod{n} \tag{3.25}$$

Suppose that a is coprime to n, then we know that the congruence holds true if and only if n is prime from **Identity**. Our program returns UNDECIDED when the congruence holds true otherwise it returns COMPOSITE. Thus if n is prime, the program must return UNDECIDED otherwise COMPOSITE

1),2),3),4) We run the program with $2 \leq n \leq 2549$, $r = n + 1$, and four different values of a. In all cases the program returned correctly the expected value. (It is UNDECIDED for prime numbers and COMPOSITES for composite numbers.

	parameters			
	n	r	a	result
1)	$2 \sim 2549$	$n+1$	1	Right
2)	$2 \sim 2549$	$n+1$	2	Right
3)	$2 \sim 2549$	$n+1$	3	Right
4)	$2 \sim 2549$	$n+1$	$n-1$	Right

When does the program misjudge a number? In the experiments above the program always judged numbers correctly. Although if n is not coprime to a ($\gcd(n, a) \neq 1$) the program can make a mistake. We consider this point more specifically.

1. If n is prime:
   $\gcd(n, a) \neq 1$ can happen only when $n|a$. In this case the congruence (3.25) holds true because $a \equiv 0 \pmod{n}$. Thus if n is prime, it is judged correctly even though $\gcd(n, a) \neq 1$.

2. If n is composite:
   Consider conditions when the congruence (3.25) holds despite n being composite. Let

   $$n = \prod_i^m q_i^{k_i}, \tag{3.26}$$

   The coefficient of x^{q_i} is:

   $$_nC_{q_i}(-a)^{q_i} \tag{3.27}$$

   Here $_nC_{q_i} \not\equiv 0 \pmod{n}$, since it does not have q_i as a factor. (Recall the proof of [**Identity**].) In order that the all coefficients of x^{q_i} are equivalent to 0 mod n,

   $$a = w \prod_i^m q_i^{\ell_i} \quad (\ell_i \times q_i \geq k_i,\ q_i \not| w,\ w > 0) \tag{3.28}$$

Then execute the program with some sample parameters based on (3.28).

A. If $a \geq n$:

	parameters			
	n	r	a	result
5)	$2 \sim 2549$	$n+1$	n	UNDECIDED for all
6)	$2 \sim 2549$	$n+1$	$3n$	UNDECIDED for all
7)	$2 \sim 2549$	$n+1$	n^2	UNDECIDED for all

We ran the program with $a = n, 3n, n^2$. Then the program returned UNDECIDED for all numbers. That is all composite numbers were misjudged. This result is logical since the coefficients of the both sides of the congruence other than those of x^n must be zero modulo n.

B. If $a < n$: We make a sample parameters based on (3.28) and ran the program with the parameters.

	parameters			
	n	r	a	result
8)	$2^5 \cdot 3^4$	$n+1$	$2^3 \cdot 3^2$	COMPOSITE (right judge)

This time the program correctly judged n $(= 2^5 \cdot 3^4)$ as composite. This means at least one pair of the coefficients of both sides is different. We checked that and found that the constant terms were different. In fact if a is such that (3.28),

$$a^n \equiv 0 \pmod{n} \tag{3.29}$$

$$\Rightarrow a^n \not\equiv a \pmod{n}. \tag{3.30}$$

Consequently we obtained a variant of [**Identity**].

[**Identity'**] If $a < n$,

$$(x-a)^n \equiv (x^n - a) \pmod{n} \tag{3.31}$$

if n is prime: TRUE
if n is composite: FALSE

Consider prime numbers with $(\mathbf{mod}\ x^r - 1)$ From [**Identity'**] above, if n is prime,

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}\ (1 \leq r \leq n). \tag{3.32}$$

By using this, we checked the correctness of the program logic to calculate modulo $x^r - 1$. The program correctly returned UNDECIDED for all prime numbers in the following three cases.

	parameters			
	n	r	a	Judge
9)	$2 \sim 99973$ only prime	1	3	UNDECIDED for all
10)	$2 \sim 3253$ only prime	$\lceil n/2 \rceil$	12	UNDECIDED for all
11)	$2 \sim 2741$ only prime	n	4	UNDECIDED for all

Consider composite numbers with $(\mathbf{mod}\ x^r - 1)$ The experiments we have done so far were somewhat trivial or logically obvious cases. They rather intended to check the correctness of the program logic itself. The more important case is when the program tests composite numbers with relatively small r.

Then we test composite numbers by the following program with relatively small r's, and find the smallest number that is not detected as composite.

```
for n = 4 to large enough number
   if (n is composite)
      flag ← 1;
      for a = 1 to 2√r log2 n
         if ( (x − a)^n ≢ (x^n − a) (mod x^r − 1, n) )
            flag ← 0;
            break;
      if (flag = 1) output n and stop;
   n ← n + 1;
```

Result:

r	output	$\log_2$ (output)
1	561	9
2	561	9
3	1729	11
5	252601	18
7	1152271	20
11	1615681	21
13	2508013	21

This result shows that the program can work correctly with small r's until it reaches a comparatively big composite number. In other words it is difficult to find a "bad composite number" that cannot be detected as composite, if r is just larger than 13 or so. Since it was difficult to conduct further various experiments for consecutive n, we tested some random generated composite numbers. However we did not find such a "bad composite number" with r that is larger than 13.

p^k type composite The original paper claims that the only case that the for-loop in the AKS algorithm can not detect n as composite with a useful prime r is that n is of the form p^k ($k > 1$, p is prime). We accordingly test this type of numbers in order to find the "bad composite number."

$n = p^k$

p (prime)	$2 \le k \le \lfloor 62 \log_p 2 \rfloor$
2	$2 \sim 62$
3	$2 \sim 39$
5	$2 \sim 22$
11	$2 \sim 23$
...	...
4225457	2

All numbers in the table above were tested with $r = 1, 2, 13$. Again no "bad composite number" was found. These results suggest the possibility that the necessary size of useful primes can be smaller than the theoretically assured size, $\approx (\log_2 n)^6$ or $\approx (\log_2 n)^2$ (when it is co-Sophie Germain prime).

Upper bound of for-loop The upper bound of the for-loop of the AKS algorithm is $2\sqrt{r}\log_2 n$. Then we experimentally observe how many iterations the program actually needs to detect composite numbers. The following algorithm is used.

```
for n = 4 to large enough number
   if (n is composite)
      flag ← 1;
      for a = 1 to 2√r log2 n
         if ( (x − a)^n ≢ (x^n − a) (mod x^r − 1, n) )
            output n and a
            flag ← 0;
            break;
      if (flag = 1) output n and ``error'';
   n ← n + 1;
```

The value of a in an output indicates the number of repetition used to detect a given composite number.

Case: $n = 4 \sim 500003$ (sequential)

$r = 17$
$n = 4 \sim 500003$ (sequential)

n	a
$289 = 17^2$	3
$4913 = 17^3$	3
$83521 = 17^4$	3
otherwise	1

$r = 19$
$n = 4 \sim 500003$ (sequential)

n	a
$361 = 19^2$	3
$6859 = 19^3$	3
$130321 = 19^4$	3
otherwise	1

Case: $n = p^k, 2 \leq k \leq \lfloor 62 \log_p 2 \rfloor$

$r = 13$
$p = 2 \sim 218131$

p	k	a
13	2	3
13	3	3
13	4	3
13	5	3
13	6	3
13	7	3
13	8	3
otherwise		1

$r = 17$
$p = 2 \sim 287117$

p	k	a
17	2	3
17	3	3
17	4	3
17	5	3
17	6	3
17	7	3
otherwise		1

$r = 41$
$p = 2 \sim 59359$

p	k	a
41	2	3
41	3	3
41	4	3
41	5	3
otherwise		1

$r = 101$
$p = 101 \sim 1669$

p	k	a
101	2	3
101	3	3
101	4	3
otherwise		1

$r = 211$
$p = 211 \sim 463$

p	k	a
211	2	3
211	3	3
211	4	3
otherwise		1

$r = 3001$
$p = 3001 \sim 3119$

p	k	a
3001	2	3
otherwise		1

As far as the results we obtained, the only case that $a \neq 1$ was when n was of the form r^k, and then a took only the value of 3. This implies that less iterations of the for-loop might be sufficient to detect composite numbers than the theoretically required $2\sqrt{r}\log_2 n$.

3.4 Full Implementation and Measurement of CPU time

In this section, the AKS algorithm is implemented by combining previous programs. Some simple program components are also added for missing parts. Then CPU time of the program is measured for several types of numbers.

3.4.1 Implementation

The line numbers below indicate the line number of the AKS algorithm in Section 2.2.

Line 1 if (n is of the form $a^b, b > 1$)

Pseudo-code

```
for b = 2 to ⌊log2 n⌋
   a = n^(1/b);
   if(a^b = n) return TRUE;
return FALSE;
```

Line 4 if ($\gcd(n, r) \neq 1$)

Pseudo-code

```
if(r|n)
   return TRUE;
else
   return FALSE;
```

Line 5 if (r is prime)

Pseudo-code

```
for i = 2 to ⌊√n⌋
   if(i|n) return FALSE;
return TRUE;
```

Line 6 $q \leftarrow$ the largest factor of $r - 1$

Pseudo-code

```
i ← 2;
k ← ⌊√n⌋;
while(i ≤ k)
   if(i|n)
      n = ⌊n/i⌋;
      k = ⌊√n⌋;
   else
      i ← i + 1;
```

Line 7 calculate $n^{\frac{r-1}{q}} \pmod{r}$

Pseudo-code

```
e ← (r−1)/q;
ñ ← n (mod r);
m ← 1;
for i = 1 to e
    m ← mñ (mod r);
return m;
```

Line 12 congruence

Use the algorithm in Section 3.3

3.4.2 Measurement of CPU time

The running time of the AKS algorithm is expected to be very different between when it deals with a prime number or a composite number. When the algorithm checks a prime number, it always iterates the for-loop until the upper bound $a = 2\sqrt{r}\log_2 n$. On the other hand, a composite number is detected far before the for-loop reaches $a = 2\sqrt{r}\log_2 n$ or even it can be detected by the a^b check at Line 1 or the GCD check at Line 4. Thus prime numbers are expected to be much more costly on processing time than composite numbers.

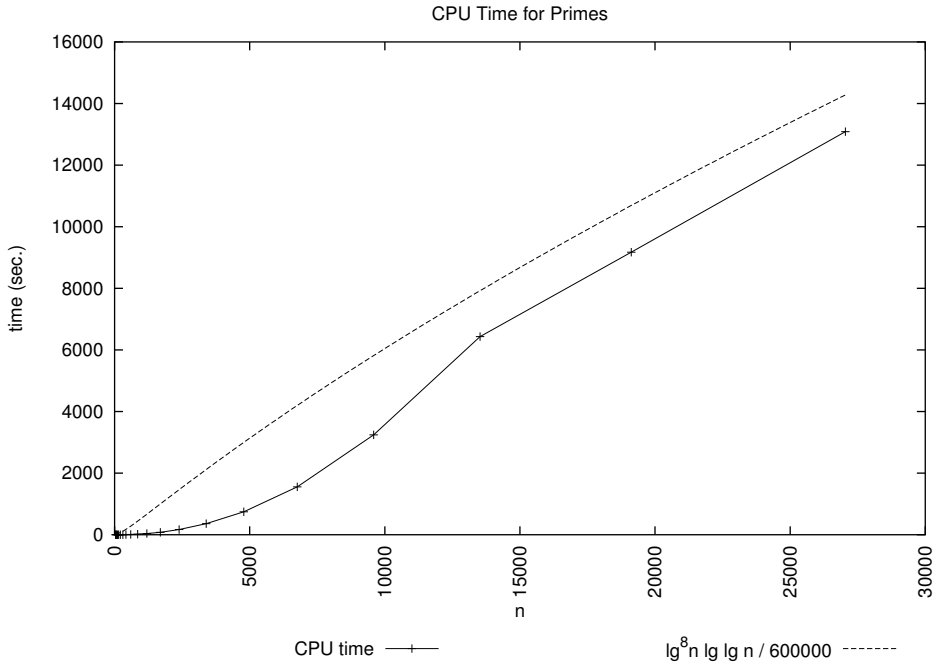
The processing CPU time for some prime numbers are shown in the following table.

n (prime)	bits of n	CPU time (sec.)
17	5	0.00
29	5	0.00
37	6	0.01
53	6	0.02
73	7	0.04
107	7	0.11
149	8	0.23
211	8	0.55
307	9	1.39
421	9	3.09
599	10	7.28
853	10	16.77
1193	11	36.90
1693	11	79.92
2389	12	169.10
3389	12	357.36
4783	13	743.19
6761	13	1555.16
9587	14	3243.78
13523	14	6433.08
19121	15	9172.17
27043	15	13088.34

The for-loop makes up most part of the CPU time in the whole process. The CPU time of the while-loop is very small and it can be ignored. The number of steps of the for-loop in our implementation is estimated as follows.

- the number of iteration of the for-loop: $O(r^{\frac{1}{2}} \log_2 n)$
- one iteration
  - the number of the inside iteration for $(x-a)^n$: $O(\log_2 n)$ (By using the square and multiply algorithm)
  - the number of multiplications of coefficients in one inside iteration: $O(r^2)$ (For LHS $\times$ LHS)
  - each multiplication of coefficients: $O(\log_2 n \log_2 \log_2 n)$ (Suppose using FFT for multiplication of integers)
- thus the whole step is: $O(r^{\frac{5}{2}} (\log_2 n)^3 \log_2 \log_2 n)$

If r is co-Sophie Germain prime, $r^{\frac{1}{2}} = \log_2 n$. Hence the whole step is $O((\log_2 n)^8 \log_2 \log_2 n)$.
The next graph plots the data of the previous table.
It also shows the curve of $O((\log_2 n)^8 \log_2 \log_2 n)/600000$ for comparison.



The following table shows the predicted CPU time when we suppose the CPU time can be approximated by $O((\log_2 n)^8 \log_2 \log_2 n)/600000$.

bits of n	CPU time (sec)	years
100	110730936496	3511
500	58370991436602100	1850931996
1000	16609640474436800000	526688244370
2000	4678734628122490000000	148361701805000

As already mentioned in section 3.1 any composite numbers in this range is detected by the a^b check or the GCD check. We measured CPU time for some composite numbers and observed these processing times were less than 0.01 sec. In most cases, CPU time for composite numbers is expected much shorter than that for prime numbers, because the for-loop terminates after smaller number of iterations.

Chapter 4

Conclusions

We found the following facts from the experiments we have conducted.

- All the smallest useful primes for each given number n we found are co-Sophie Germain primes.
- When co-Sophie Germain primes are excluded for useful primes, the useful primes we have found in the experiments are of the form $4q + 1$ (q is prime).
- The lower bound of n in order to have a useful prime constantly is 11809 with $r = 11807$.
- The smallest composite number that is actually detected by using a useful prime in the AKS algorithm is 120337528573=346891×346903. Here 346891 and 346903 are successive primes. And the useful prime is 346877.
- All composite numbers less than 63 bits were detected when $r \geq 17$ in our experiments. This implies smaller size of r of $\pmod{x^r - 1, n}$ might be sufficient to detect composite numbers rather than its theoretically required size $\simeq O((\log_2 n)^6)$ or $O((\log_2 n)^2)$ (when r is co-Sophie Germain prime).
- When we have checked composite numbers less than 63 bits, the needed number of iterations of the for-loop was at most 3. That happened several times when n is a multiple of its useful prime r. In all other cases, composite numbers are detected by just one iteration. This implies the necessary number of iteration of the for-loop might be much smaller than theoretically required $2\sqrt{r}\log_2 n$.

Appendix A

Tables of Useful Primes

k	$n = 2^k$	r	$64(\log_2 n)^2$
63	9223372036854775808	262643	254016
64	18446744073709551616	270443	262144
65	36893488147419103232	278819	270400
66	73786976294838206464	287747	278784
67	147573952589676412928	296027	287296
68	295147905179352825856	304763	295936
69	590295810358705651712	313883	304704
70	1180591620717411303424	322727	313600
71	2361183241434822606848	331883	322624
72	4722366482869645213696	341219	331776
73	9444732965739290427392	350699	341056
74	18889465931478580854784	360323	350464
75	37778931862957161709568	369827	360000
76	75557863725914323419136	379703	369664
77	151115727451828646838272	389447	379456
78	302231454903657293676544	400067	389376
79	604462909807314587353088	409967	399424
80	1208925819614629174706176	420263	409600
81	2417851639229258349412352	430499	419904
82	4835703278458516698824704	440939	430336
83	9671406556917033397649408	451679	440896
84	19342813113834066795298816	462719	451584
85	38685626227668133590597632	473987	462400
86	77371252455336267181195264	484487	473344
87	154742504910672534362390528	495707	484416
88	309485009821345068724781056	506963	495616

Table A.1: Useful Primes for $n = 2^k$ $(63 \leq k \leq 88)$

k	$n = 2^k$	r	$64(\log_2 n)^2$
89	618970019642690137449562112	518543	506944
90	1237940039285380274899124224	530183	518400
91	2475880078570760549798248448	541799	529984
92	4951760157141521099596496896	553643	541696
93	9903520314283042199192993792	565667	553536
94	19807040628566084398385987584	577667	565504
95	39614081257132168796771975168	589847	577600
96	79228162514264337593543950336	602639	589824
97	158456325028528675187087900672	614963	602176
98	316912650057057350374175801344	627479	614656
99	633825300114114700748351602688	640019	627264
100	1267650600228229401496703205376	653267	640000
101	2535301200456458802993406410752	666143	652864
102	5070602400912917605986812821504	679319	665856
103	10141204801825835211973625643008	692663	678976
104	20282409603651670423947251286016	705863	692224
105	40564819207303340847894502572032	719267	705600
106	81129638414606681695789005144064	732923	719104
107	162259276829213363391578010288128	746723	732736
108	324518553658426726783156020576256	760619	746496
109	649037107316853453566312041152512	775007	760384
110	1298074214633706907132624082305024	788819	774400
111	2596148429267413814265248164610048	803819	788544
112	5192296858534827628530496329220096	817319	802816
113	10384593717069655257060992658440192	831863	817216
114	20769187434139310514121985316880384	846563	831744
115	41538374868278621028243970633760768	861299	846400
116	83076749736557242056487941267521536	876263	861184
117	166153499473114484112975882535043072	891179	876096
118	332306998946228968225951765070086144	906539	891136
119	664613997892457936451903530140172288	921743	906304
120	1329227995784915872903807060280344576	937187	921600
121	2658455991569831745807614120560689152	952739	937024
122	5316911983139663491615228241121378304	968459	952576

Table A.2: Useful Primes for $n = 2^k$ $(89 \leq k \leq 122)$

k	$n = 2^k$	r	$64(\log_2 n)^2$
123	10633823966279326983230456482242756608	984119	968256
124	21267647932558653966460912964485513216	1000667	984064
125	42535295865117307932921825928971026432	1016663	1000000
126	85070591730234615865843651857942052864	1032419	1016064
127	170141183460469231731687303715884105728	1048703	1032256
128	340282366920938463463374607431768211456	1065047	1048576
129	680564733841876926926749214863536422912	1081979	1065024
130	1361129467683753853853498429727072845824	1098443	1081600
131	2722258935367507707706996859454145691648	1115267	1098304
132	5444517870735015415413993718908291383296	1132403	1115136
133	10889035741470030830827987437816582766592	1150739	1132096
134	21778071482940061661655974875633165533184	1166603	1149184
135	43556142965880123323311949751266331066368	1184123	1166400
136	87112285931760246646623899502532662132736	1201247	1183744
137	174224571863520493293247799005065324265472	1218923	1201216
138	348449143727040986586495598010130648530944	1236623	1218816
139	696898287454081973172991196020261297061888	1254503	1236544
140	1393796574908163946345982392040522594123776	1273127	1254400
141	2787593149816327892691964784081045188247552	1290659	1272384
142	5575186299632655785383929568162090376495104	1309079	1290496
143	11150372599265311570767859136324180752990208	1327199	1308736
144	22300745198530623141535718272648361505980416	1346039	1327104
145	44601490397061246283071436545296723011960832	1364303	1345600
146	89202980794122492566142873090593446023921664	1382999	1364224
147	178405961588244985132285746181186892047843328	1402019	1382976
148	356811923176489970264571492362373784095686656	1420883	1401856
149	713623846352979940529142984724747568191373312	1440107	1420864
150	1427247692705959881058285969449495136382746624	1459427	1440000
151	2854495385411919762116571938898990272765493248	1478663	1459264
152	5708990770823839524233143877797980545530986496	1498439	1478656
153	11417981541647679048466287755595961091061972992	1517927	1498176
154	22835963083295358096932575511191922182123945984	1537847	1517824

Table A.3: Useful Primes for $n = 2^k$ $(123 \leq k \leq 154)$

$n = 2^k$

k	r	$64(\log_2 n)^2$
100	653267	640000
200	2560367	2560000
300	5760659	5760000
400	10240067	10240000
500	16000463	16000000
600	23040167	23040000
700	31360067	31360000
800	40962683	40960000
900	51840407	51840000
1000	64001303	64000000
1100	77440043	77440000

Table A.4: Useful Primes for $n = 2^k$ $(100 \leq k \leq 1100)$

Appendix B

Source Code

Java	http://www.cs.rit.edu/ axt9690/ms_project/java/

File	Description
BigBinomial.java	An implementation of the for-loop of the AKS algorithm with BigInteger
Binomial.java	An implementation of the for-loop of the AKS algorithm
CoSophie.java	Output co-Sophie Germain primes
FileTool.java	Tools about files
nrTableGen.java	Find useful primes for sequential numbers
PascalTriangle.java	Output Pascal's triangle
Poly.java	Functions for polynomial calculation
pTable.java	Output prime numbers
rTable.java	Create $r - n$ table to find useful primes for sequential numbers
Tool.java	Basic functions and mathematical tools used for the AKS algorithm
UsefulPrime.java	Find a useful prime (a direct implementation)

C	http://www.cs.rit.edu/ axt9690/ms_project/c/

File	Description
mp.h	The definition of the multi-precision integer
mp_func.h	The definition of the multi-precision functions
tool.func.h	The definition of basic tool functions
AKS.c	A full implementation of the AKS algorithm
Binomial.c	An implementation of the for-loop of the AKS algorithm
cpu1.c	Measure CPU time
cpu2.c	Measure CPU time
mp.c	Basic multi-precision functions
mp_u_prime.c	Find useful primes with multi-precision integers
Tool.c	Basic functions and mathematical tools used for the AKS algorithm
UsefulPrime.c	Find a useful prime

Bibliography

[1] Manindra Agrawal, Neeraj Kayal and Nitin Saxena. PRIMES is in P. Available at http://www.cse.iitk.ac.in/news/primality.html.

[2] T. M. Apostol. *Introduction to Analytic Number Theory.* Springer-Verlag, 1976.

[3] R. C. Baker and G. Harman. The Brun-Titchmarsh Theorem on average. In *Proceedings of a conference in Honor of Heini Halberstam, Volume 1,* pages 39-103, 1996.

[4] Rajat Bhattacharjee and Prashant Pandey. Primality testing. Technical report, IIT Kanpur, 2001. Available at http://www.cse.iitk.ac.in/research/btp2001/primality.html.

[5] Chris Caldwell. *The Prime Pages* http://www.utm.edu/research/primes/

[6] E. Fouvry. Theoreme de Brun-Titchmarsh; application au theoreme de Fermat. *Invent. Math.,*79:383-407, 1985.

[7] Hisashi Fukagawa *Miller-Rabin ni yoru sosuu no kakuritsuteki hanteihou.* http://www.h-fukagawa.com/documents/miller-rabin.pdf

[8] Yves Gallot. *Implementation of the AKS algorithm.* http://perso.wanadoo.fr/yves.gallot/src/

[9] Eitan Gurari. *An Introduction to the Theory of Computation.* Ohio State University http://www.cis.ohio-state.edu/~gurari/theory-bk/theory-bk.html

[10] G. H. Hardy and J. E. Littlewood. Some problems of 'Partitio Numerorum' III: On the expres-sion of a number as a sum of primes. *Acta Mathematica*, 44:1-70, 1922.

[11] Neeraj Kayal and Nitin Saxena. Towards a deterministic polynomial-time test. Technical report, IIT Kanpur, 2002. Available at http://www.cse.iitk.ac.in/research/btp2002/primality.html.

[12] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1986.

[13] Hisanori Mishima. *Sugakusya no missitsu.* http://www.asahi-net.or.jp/~KC2H-MSM/index.htm

[14] Mitsunori Ogihara and Stanisław Radziszowski. Agrawal-Kayal-Saxena Algorithm for Testing Primality in Polynomial Time. Available at http://www.cs.rit.edu/ spr/PUBL/primes.html

[15] Kabocha pi. *Kabocha suuron.* http://www.hcn.zaq.ne.jp/caaid806/math/num_thr.html

[16] Jorg Rothe. *On Some Promise Classes in Structural Complexity Theory.* http://www.informatik.uni-trier.de/GI/FG-014/PhDs/JoergRothe.html

[17] Douglas R. Stinson. *Cryptography : theory and practice.* CRC press LLC, 1995.

[18] Hiranouchi Toshiro. *Sosu daisuki.* http://homepage2.nifty.com/hiranouchi/

[19] Ken Uehara. *Coding Theory.* Department of Information Science, Saga University. http://www.ma.is.saga-u.ac.jp/uehara/coding.html

[20] Tomohisa Wada *Galois tai no kantanna setsumei.* Department of Information Engineering University of the Ryukyus. http://bw-www.ie.u-ryukyu.ac.jp/~wada/vhdl/GaloisField.html

[21] Eric Weisstein. *World of Mathematics.* Wolfram Research. Dec.15 2002 http://mathworld.wolfram.com/

[22] Koichi Yamazaki. *Kakuritsu algorithm ni tsuite.* Department of Computer Science, Gunma University http://www.comp.cs.gunma-u.ac.jp/~koichi/RAND/rand/rand.html

[23] *Shoto seisuron.* Aozora Gakuen Sugakuka http://www80.sakura.ne.jp/~aozora/suuron/suuron.html

[24] *Kokaikagi ango RSA nyumon.* Yousei genjitsu. http://www.faireal.net/articles/5/24/#d20519

[25] *Seisuron.* Kiso Lab, Dept. of Information Engineering, Shinsyu university. http://markun.cs.shinshu-u.ac.jp/learn/integer/index.html

[26] *Math and Computing.* Nara University of Education. http://www.nara-edu.ac.jp/~asait/

[27] *The PRIMES is in P little FAQ.* http://crypto.cs.mcgill.ca/ stiglic/PRIMES_P_FAQ.html

[28] *Computational complexity theory.* Wikipedia. http://www.wikipedia.org/wiki/Complexity_theory_in_computation