INTEGER FACTORIZATION ALGORITHMS

by

NOR AZUANI BINTI HASAN

**Dissertation submitted in partial fulfillment
of the requirements for the degree
of Master of Science in Mathematics**

April 2010

LIST OF CONTENTS

ABSTRACT

Factoring integers is not an easy task. It is classified as a hard algorithm such that the security of the RSA cryptosystem is based upon. Many different methods for factoring integers have been developed. There are many set of classes of algorithms such as Trial Division, Fermat, Pollard Rho, Pollard p-1 and General Number Field Sieve(GNFS).

In this dissertation, we discussed four factoring algorithms such as Trial Division, Fermat, Pollard Rho and Pollard p-1. Some examples are given to illustrate the mathematical concepts in the integer factorization algorithms. A programming using software Mathematica version 7.0 were used to carry out the integer factorization algorithms. Results on integer factorization algorithms obtained were shown and discussed.

CHAPTER 1

INTRODUCTION

Cryptography is an important building block of e-commerce system. The used of cryptography include ATM cards, computer passwords, and electronic commerce. Cryptography is a word from Greek κρυπτός, *kryptos*, "hidden, secret"; and γράφω, *gráphō*, "I write", or -λογία, *-logia*, is the practice and study of hiding information.

In particular, public key cryptography can be used for ensuring the authenticity of information in an organization. To protect the sensitive information in an organization, encryption can be applied so that the encrypted data is completely meaningless except to the individuals with the correct decryption key.

Encryption is the process of converting ordinary information into cipher text. While decryption is the reverse, in other words, moving from the cipher text to ordinary information. A cipher is a pair of algorithms that create the encryption and the decryption. The detailed operation of a cipher is controlled both by the algorithm and by a key. This is a secret parameter for a specific message exchange context.

1.2 Objectives

This dissertation is to study the integer factorization and we are also going to discuss a few factoring algorithms related to cryptography and their findings by using software Mathematica version 7.0.

1.3 Dissertation Outline

There are 4 chapters in this dissertation. In chapter 1, we give an introduction on integer factorization. In chapter 2, we discuss briefly about integer factorization algorithms and the fundamental theorem of Arithmetic. In chapter 3, we discuss the four algorithms and their findings. In this chapter, we also are going to discuss about general number field sieve (GNFS). Lastly, in chapter 4, we have a future direction and conclusion which we summarized briefly on algorithms and the findings.

In 1750 Euler had an idea of integer factorization. He only looked at integer based on special forms. One of the method is integers can be written as $n= a^2 + Db^2$. He used the method to factor some large numbers for that time.

In 1974, John Pollard was developed the p-1 method and he targeted at a special class of integers which relies on the hope that there is at least one prime p in the factorization of n, such that p-1 is smooth. This means p-1 is the product of relatively small primes. Then, Brent optimized the p method in 1980 and Carl Pomerance introduced the quadratic sieve algorithm which added some digits to the numbers that could be factored resulting in a factorization of a 71 digit number in 1983.

Lenstra developed a new method by using elliptic curves method in 1987 which is specialized for composites with small factors. In 1988, John Pollard, Richard P. Brent, J. Brillhart, H. W. Lenstra, C. P. Schnorr and H. Suyama outlining an idea of factoring a special class of numbers by using algebraic number fields. Shortly after number field sieve was implemented and was generalized to be general purpose algorithm. It is the most complex factoring algorithm but it is also the fastest factorization method of a 512 bit composite (Integer factorization, 2010).

2.2 Fundamental Theorem of Arithmetic

The prime numbers are the integers which greater than 1 that can factored into two positive integers. For example, integers 2, 3, 5, 7, ... are primes while 1, 4, 6, 8, .. are non prime integers. The non prime integers which greater than 1 are called

composite numbers. For example, 10 is composite number such that 10 can be factored into two distinct ways as 1 x 10 and 2 x 5.

The fundamental theorem is useful to break down integers into smallest prime numbers. Thus 35 is 5 x 7 and 90 is 2 x 3 x 3 x 5, this implies that every positive integer can be factored uniquely. Before we show the fundamental theorem of arithmetic, we present some definitions related to our work.

Definition 2.2.1: A **Composite Number** is a number which can be divided evenly by numbers other than 1 or itself.

Definition 2.2.2: A **prime number** is a positive integer greater than 1 that is divisible by no positive integers other than 1 and itself.

Definition 2.2.3: A **trivial factor** is a positive integer factor x of n such that $x = 1$ or $x = n$.

Definition 2.2.4: A **nontrivial factor** is a positive integer factor x of n such that x is between 1 and n.

Definition 2.2.5: A **floor function** gives the greatest integer less than or equal to n.

Definition 2.2.6: A **ceiling function** gives the smallest integer greater than or equal to n.

Theorem 2.2.1: If n is a positive integer and all its prime factors are smaller than B, then n is called ***B*-smooth**.

Definition 2.2.7:

Let two numbers x and y are defined as **congruent modulo** n if the difference between x and y is an integer multiple of n (Pollard Rho method, 2010).

Example 2.2.1: Suppose that $x = 37$, $y = -14$ and $n = 17$. x is congruent to y modulo n. We find the difference x and y, $(x-y) = 37 - (-14) = 51 = 3 * 17$.

Definition 2.2.8:

The **greatest common divisor** of a and b is the largest positive integer dividing both a and b and is denoted by either gcd(a, b) or by (a, b).

Example 2.2.2: Compute gcd(482, 1180) (Trappe, 2005).

Solution: First, we divide 1180 by 482. Then, we get the quotient is 2 and the remainder is 216. Now we divide 482 by the remainder 216. Then the quotient is 2 and the remainder is 50. We repeat this process of dividing the previous one by the most recent remainder until the last nonzero remainder is the gcd which is 2.

$$1180 = 2.482 + 216$$

$$482 = 2.216 + 50$$

$$216 = 4.50 + 16$$

$$50 = 3.16 + 2$$

$$16 = 8.2 + 0$$

Theorem 2.2.2: A **fundamental theorem of arithmetic** is every integer greater than 1 can be factored uniquely into product of primes (Wikipedia, 2010).

Given that $n = p_1^{e1} \; p_2^{e2} \ldots p_m^{em}$ where $p_1 < p_2 < \ldots < p_m$ are primes and $e_1 > 0$, $e_2 > 0$, $\ldots$, $e_m > 0$ are integers. Before we proof the theorem, let us give briefly a preliminary fact about primes.

Proposition 2.2.1: (Euclid's First Theorem) An integer $p > 1$ is prime if and only if it satisfies the property: for all integers a, b, $p| ab$ implies $p|a$ or $p|b$.

Proof: Assume that a, b are non-zero. In this case, we have either gcd(a, p)= 1 or gcd(a, p)= p since p is a prime.

Conversely, assume for all integers a, b, $p|ab$ implies $p|a$ or $p|b$. If p is composite then $p=ab$ for some $a < p$ and $b < p$. But $p|p = ab$ then implies $p|a$ or $p|b$, both are impossible.

In fact, rather than using Euclid's First Theorem directly in the proof of the Theorem of Arithmetic, we use the following consequence of it.

Corollary 2.2.1: Let p be a prime. For any integers $a_1, a_2, \ldots, a_k$, $p| a_1, a_2, \ldots, a_k$ implies $p| a_1$ or $p| a_2$ or ... or $p| a_k$.

Here we need the following basic result.

Lemma 2.2.1: If $m > 1$ is an integer, then the smallest $d > 1$ which divides m must be a prime number.

Proof: Let $d > 1$ be the smallest positive divisor of m. Suppose d has a factor b with $1< b\leq d$ and $d = bc$. But $b|d$ and $d|m$ implies $b|m$. Since d is the smallest factor of m which is greater than 1, we must have $b = d$, so d is prime.

Proof of Theorem of Arithmetic: First, we show that n is a product of primes. If n is a prime then we are done. Let us assume that n is not a prime.

Let $n_0= n$ and let $i = 0$.

1) Let d_i denote the smallest divisor of n_i. (From the above lemma, d_i is a prime.) Let

$n_{i+1} = n_i / d_i$ (Note: $n_{i+1} < n_i$).

2) If n_{i+1} is prime then stop. If n_{i+1} is not a prime, then replace i by $i+1$ and go to (1).

Since $n_0 > n_1 > \ldots$, this process will be terminated. Let say n_k is a prime. Then $n = n_0 = n_k\, d_{k-1} \ldots d_0$, so n is a product of primes. By arranging the primes in increasing order and grouping identical primes in the product together, we may write n as in the theorem. Now we prove uniqueness. Suppose $n = p_1^{e_1} p_2^{e_2} \ldots p_m^{e_m} = p_1^{f_1} p_2^{f_2} \ldots p_m^{f_m}$ where the p_i's are distinct primes but not necessarily in any particular order and the $e_1, e_2, \ldots, e_m$ and $f_1, f_2, \ldots, f_m$ are non-negative integers. By the **corollary 2.2.1**, we must have $p_1 | p_1^{f_1}$ or $p_1 | p_2^{f_2}$ or.... $p_1 | p_m^{f_m}$. Since the p_i's are distinct, the only possibility is if $p_1 | p_1^{f_1}$ so in particular $f_1 > 0$. If $f_1 > e_1$ then $p_2^{e_2} \ldots p_m^{e_m} = p_1^{(f_1 - e_1)} p_2^{f_2} \ldots p_m^{f_m}$ and the **corollary 2.2.1** implies $p_1 | p_2^{e_2}$ or ... $p_1 | p_m^{e_m}$. This is impossible since the p_i's are distinct. Similarly, $e_1 > f_1$ leads to a contradiction. This leaves the only possibility $e_1 = f_2$, so $p_2^{e_2} \ldots p_m^{e_m} = p_2^{f_2} \ldots p_m^{f_m}$. Proceeding inductively, we find $e_2 = f_2, \ldots, e_m = f_m$. This leads to uniqueness and the theorem.

2.3 Integer Factorization Algorithms

In this topic, we will be discussed about a few factoring algorithms. Every integer can be represented uniquely as a product of prime numbers. For example, 90 is easy to factor or we can write it as 2 x 3 x 3 x 5.

Cryptography is an important building block of e-commerce systems. RSA is one of the public key for ensuring the authenticity of information in an organization. The basis of security of RSA is depending on difficulty of factoring large numbers. Integer factorization problem is to find number's prime factor.

There are several factoring algorithms can be used to factor the numbers such as Trial division, Pollard Rho, Pollard p-1, elliptic curve factorization, Number Field Sieve,etc. There are three algorithms that are most effective which can factor large number such as number field sieve, elliptic curve factorization and general number field sieve. For the next section, we will be going to discuss more detail about several algorithms.

CHAPTER 3

FACTORING ALGORITHMS AND DISCUSSIONS

3.1 Introduction

In this chapter, we are going to discuss a few factoring algorithms. Each algorithms are explained and pseudocodes are given (Connelly Barnes, 2010). We also will discuss the findings of the algorithms. In this chapter, we discussed four factoring algorithms such as the Trial division algorithm, Fermat factorization algorithm, Pollard Rho factorization algorithm and the Pollard p-1 factorization algorithm.

3.2 Trial Division Algorithm

Trial division often called the naïve method of factoring. It is the simplest algorithm for factoring an integer. Given an integer n, n refers to "the integer to be factored", trial division consists of testing whether n is divisible by any number. Note that, it is only work to test the factors less than n.

Assume that, x and y are nontrivial factor of n such that $n= xy$ and $x \leq y$.

3.4 Examples of Trial Division

To implement the algorithm above, we use a software Mathematica version 7.0. Now, we look for some examples to justify the algorithm.

Example 3.4.1: Factor 12 by using trial division method (Tripod, 2010).

Solution:

The basic idea is to find the factor by attempting to divide a number by prime numbers which are less than the square root of the number.

Assume that, x and y are nontrivial factor of 12 such that $12 = xy$ and $x \leq y$. Then we use command for to do loop until the test fail to give true. We want the output list all possible factors of 12.

For[$x= 2$, $x \leq$ Floor[$\sqrt{12}$], x++ , $y = 12 / x$];Print[" $x =$ ",x ," , " ," $y =$ " , y]]

The output we get,

$x= 2,\ y = 6$

$x= 3,\ y = 4$

Since we get the value of x, we can get the value of y directly by dividing the integer 12 with the value of x. Here, we say that the square root of 12 lies between $x = 3$ and $y = 4$.

Now, we look at another example.

Example 3.4.2: Factor 91 by using trial division method.

Solution:

The basic idea is to find the factor by attempting to divide a number by prime numbers which are less than the square root of the number.

Then we find x and y as $p=x-y$ and $q=x+y$. We solve this equation by simultaneous equation, and we get $x= (p+q)/2$ and $y = (q-p)/2$. Since $p>1$ and $q \geq p$, we find that $x \geq 1$ and $y \geq 0$.

For the algorithm, we check for each i whether $y_i = \sqrt{x_i^2 - n}$ is an integer and whether $(x_i + y_i)$, $(x_i - y_i)$ are nontrivial factors of n. If both of these are satisfies, we get the nontrivial factors. Otherwise we continue the iteration until $x_i = n$.

3.6 Pseudocode:Fermat Factorization

```
function fermatFactor(n)
    for x from ceil(sqrt(n)) to n
        ysquared: = x * x –n
        if issquare(ysquared) then
          y: = sqrt(ysquared)
          p: = (x-y)
          q: = (x+y)
          if p < > 1 and p < > n then
             return p, q
          end if
        end if
    end for
end function
```

From the output above, we noticed that not all possible factors, p and q are integer numbers. In this factorization method, we only consider the integer numbers. Hence, we get $p = 3$, $q = 7$ and $p = 1$, $q = 21$ are the factor of 21. Because of the value of $p = 1$ is not a prime number. Therefore, we take $p = 3$ and $q = 7$ as the factor of 21.

Example 3.7.2: Factor 45 by using Fermat factorization.

Solution:

We need to find the factors p and q which can be written as the difference of squares such that $n = x^2 - y^2$.

Assume that p and q are nontrivial odd factors of n such that $n = p.q$ and $p \leq q$ where $p = x-y$ and $q = x+y$.

We use command for to do loop until the test fail to give true. We also use command break to omit the value of q when it is larger than 45.

For[x= Ceiling [$\sqrt{n}$], $x \leq n$, x++, $y1 = x*x - n$; y = N[$\sqrt{y1}$, 5] ; p = N[x-y, 5] ;

q =N[x +y, 5] ; Print[" x = ", x, " , " ," y = ", y, " , " , "p = ", p , " , ", "q = " , q] ;

If [q> 44, Break[], {x, x+y}]]

The output we get,

x = 7 , y = 2.0000 , p = 5.0000 , q = 9.0000

x = 8 , y = 4.3589 , p = 3.6411 , q = 12.359

x = 9 , y = 6.0000 , p = 3.0000 , q = 15.000

x = 10 , y = 7.4162 , p = 2.5838 , q = 17.416

x = 11 , y = 8.7178 , p = 2.2822 , q = 19.718

x = 12 , y = 9.9499 , p = 2.0501 , q = 21.950

x = 13 , y = 11.136 , p = 1.864 , q = 24.136

x = 14 , y = 12.288 , p = 1.712 , q = 26.288

end if

end do

end function

3.10 Examples of Pollard Rho Factorization

Now, we look at some examples by using software Mathematica version 7.0.

Example 3.10.1: Factor 31861 by using Pollard Rho Method with $f(x) = x^2 + 1$ and x_0=1 (Pollard Rho method, 2010).

Solution:

Our aim is to find two iterates i, j where gcd $(x_i - x_j, 31861) = a > 1$ where a is an integer and we compute the greatest common divisor with n= 31861.

We start with initial guess x_0=1 as $x[0]$=1.

x[0]=1;

Then, we use command for to do loop until the test fail to give true.

For[$i = 0, i \leq 9, i$++, $x[i+1] = x[i]^2 + 1$; y = Mod[$x[i+1]$, 31861] ;

m=GCD[$x[i+1]$, 31861]; Print["x[", i+1,"]=", N[$x[i+1]$], " , " ," y = ", y , ", "," m = ", m]]

The output we get,

x[1] = 2. , y = 2 , m = 1

x[2] = 5. , y = 5 , m = 1

x[3] = 26. , y = 26 , m = 1

x[4] = 677. , y = 677 , m = 1

x[5] = 458330. , y = 12276 , m = 1

For [i =1, $i \leq 18$, i++, $x[i+1]$ =1024x[i]^2 + 32767 ; y =Mod[$x[i+1]$, 16843009];

Print["x[", i+1,"] = ", N[$x[i+1]$], " , " , " y = ", y]]

The output we get,

x[2]= 33791. , y = 33791

x[3]= 1.16924×10^{12} , y = 10832340

x[4]= 1.39992×10^{27} , y = 12473782

x[5]= 2.00682×10^{57} , y = 4239855

x[6]= 4.12398×10^{117} , y = 309274

x[7]= 1.74154×10^{238} , y = 11965503

x[8]= $3.105735707635590\times10^{479}$, y = 15903688

x[9]= $9.87708854853913\times10^{961}$, y = 3345998

x[10]= $9.98982432723791\times10^{1926}$, y = 2476108

x[11]= $1.021917082512122\times10^{3857}$, y = 11948879

x[12]= $1.069378072094810\times10^{7717}$, y = 9350010

x[13]= $1.171015128143066\times10^{15437}$, y = 4540646

x[14]= $1.404187064668079\times10^{30877}$, y = 858249

x[15]= $2.019063104083103\times10^{61757}$, y = 14246641

x[16]= $4.174454597908167\times10^{123517}$, y = 4073290

x[17]= $1.784429689855655\times10^{247038}$, y = 4451768

x[18]= $3.260609861671271\times10^{494079}$, y = 14770419

x[19]= $1.088673451010861\times10^{988162}$, y = 4020935

Now we check the greatest common divisor whether the gcd (x_i- x_j, 16843009) = a > 1 or not, where a is an integer. Then, we continue the next iteration if the gcd is not larger than 1.

GCD[x[1] - x[2], 16843009] = 1

3.13 Examples of Pollard $p-1$ Factorization

Now, let us look at a few examples to justify the algorithm by using software Mathematica version 7.0.

Example 3.13.1: Factor 65 by using Pollard p-1 method (Connelly Barnes, 2010).

Solution:

Our aim is to find an integer which can divide both x_k and 65 where $x_k \equiv 2^{k!}-1 (\text{mod } 65)$ for k=1, 2, 3, ... ,

We use command for to do loop until the test fail to give true.

For[k =1, $k \leq 7$, k++, $x[k]$ = N[$2^{k!}$-1, 4] ; y = Mod[$2^{k!}$-1, 65] ; s = GCD[y, 65] ;

Print["x[", k ,"] = ", $x[k]$, " , " , " y = ", y , " , " , " s = " , s]]

The output we get,

$x[1] = 1.000$, $y = 1$, $s = 1$

$x[2] = 3.000$, $y = 3$, $s = 1$

$x[3] = 63.00$, $y = 63$, $s = 1$

$x[4] = 1.678\times10^{7}$, $y = 0$, $s = 65$

$x[5] = 1.329\times10^{36}$, $y = 0$, $s = 65$

$x[6] = 5.516\times10^{216}$, $y = 0$, $s = 65$

$x[7] = 1.553\times10^{1517}$, $y = 0$, $s = 65$

Notice that, for the first 3 steps, we find that gcd (1, 65) = 1, gcd (3, 65) = 1 and gcd (63, 65) = 1. Then, when $k \geq 4$ we find that gcd (0, 65) = 65. This implies that the algorithm cannot find a nontrivial factor and the test never terminates.

Example 3.13.2: Factor 317017 by using Pollard p-1 method.

Solution:

$x[1] = 1.000$, $y = 1$, $s = 1$

$x[2] = 3.000$, $y = 3$, $s = 1$

$x[3] = 63.00$, $y = 63$, $s = 1$

$x[4] = 1.678 \times 10^{7}$, $y = 6982$, $s = 1$

$x[5] = 1.329 \times 10^{36}$, $y = 2520$, $s = 1$

$x[6] = 5.516 \times 10^{216}$, $y = 4268$, $s = 97$

$x[7] = 1.553 \times 10^{1517}$, $y = 1358$, $s = 97$

From the output above, the value of y is actually defined as $x_k \equiv 2^{k!}-1 \pmod{8051}$ and we also find the gcd for every value of y so that we can determine whether there exists an integer which can divide both x_k and n. Hence, 97 is a factor of 8051. In this example, we find that $x_k = 4268$ and $x_k = 1358$ are divisible by $p - 1 = 96$ and 8051 is also divisible by p-1=96.

3.14 Integer Factorization Algorithms Findings

In the Trial division and Fermat factorization, we know that both will be checked for every possible factor of number, n. These factorization algorithms always exhaust search the number to be factored. However, the algorithms are suitable for factoring small numbers like two or three digit numbers. Then, the programming will become slow when the number of digit becomes larger.

The Pollard Rho and the Pollard p -1 factorizations are intermediate test between trial division and Fermat factorization. The algorithms like Pollard Rho and the Pollard p-1 are more efficient compared to the trial division and Fermat factorization.

- $2^0 3^0 5^1 7^0 = 5 \equiv 192 = 2^6 3^1 5^0 7^0$.....(2)

- $2^0 3^2 5^0 7^0 = 9 \equiv 196 = 2^2 3^0 5^0 7^2$.....(3)

- $2^3 3^0 5^0 7^1 = 56 \equiv 243 = 2^0 3^5 5^0 7^0$.....(4)

From equation (3), we say that $3^2 \equiv 14^2 \pmod{n}$, which gives the factorization 187 = gcd(14-3, 187) × gcd(14+3, 187) = 11×17 .

3.16 RSA Cryptography

RSA stands for Rivest, Shamir and Adleman who first publicly described it for public key algorithm in 1977. It supports encryption and digital signatures. It also gets its security from integer factorization problem. It is easy to understand and implement.

RSA is used in security such as email security, transport data security and many more. RSA gets its security from factorization problem. The basis of security of RSA is depending on difficulty of factoring large numbers over 1000 bits long numbers are used. Integer factorization problem is to find number's prime factor.

3.17 RSA Algorithm

We introduced briefly about key generation. First, we select random prime numbers p and q. Then, we compute modulus pq and phi, $\Phi = (p\text{-}1)\ (q\text{-}1)$. We also

CHAPTER 4

FUTURE DIRECTION AND CONCLUSION

In this chapter, we discussed future direction and conclusion of integer factorization. If we look at the current development there are number of research in integer factorization. Nowadays, computational power is increasing day by day and although there is far to a computer being able to factor a large bit number which is over 500 bit in reasonable time. But many researchers intend to continue working on applications of integer factorization, and will try some new ideas for improving the efficiency of the algorithms to factor the large bit numbers.

There are no known algorithms which can factor large integers efficiently. In the Trial division and Fermat factorization, we know that both will be checked for every possible factor of number, n. These factorization algorithms always exhaust search the number to be factored. However, the algorithms are suitable for factoring small numbers like two or three digit numbers and the programming will become slow when the number of digit becomes larger.

The algorithms such as Pollard Rho and Pollard p-1 factorization are in most cases more efficient than trial division and Fermat factorization algorithms.

For Pollard p-1 factorization, the smoothness bound needs to be small enough to ensure that the algorithm run quickly and we also do not want to choose B to be so small so that there is no hope of finding a nontrivial factor.

Cryptography is an important building block of e-commerce systems. RSA is one of the public key for ensuring the authenticity of information in an organization. The basis of security of RSA is depending on difficulty of factoring large numbers over 1000 bit long numbers are used. The security of RSA relies on it because it can be helpful for users of the RSA encryption public key algorithm for choosing suitable key for an appropriate level of security.

Therefore, integer factorization algorithms are important subject in mathematics for practical purposes such as data security.

APPENDICES

APPENDIX A

Trial Division Method

The basic idea is to find the factor by attempting to divide a number by prime numbers which are less than the square root of the number.

Example 3.4.1: Factor 12 by using trial division method.

Assume that, x and y are nontrivial factor of 12 such that $12 = xy$ and $x \leq y$. Then we determine whether $x|12$ for x=2, 3, ..., Floor [$\sqrt{12}$].

First, we assign 12 as the value of n

n=12

12

Then,we determine $x|12$ for x=2, 3, ..., Floor[$\sqrt{12}$]. To do so, we use command For [start, test, incr, body] executes start, then repeatedly evaluates body and incr until test fails to give True.

For[x=2, x≤ Floor [$\sqrt{n}$], x++, y=n/x; Print [" x = ", x , " , " , "y = " , y]]

x = 2 , y = 6

x = 3 , y = 4

From the output above, we know that $12 = x\,y$ and this implies that $x \leq y$ and the upper bound of $x \leq$ Floor[$\sqrt{12}$]. Therefore the square root of 12 lies between $x = 3$ and $y = 4$.

Example 3.4.2: Factor 91 by using trial division.

Clear[x, y]

First, we assign 91 as the value of n.

n=91

91

Then, we determine $x|91$ for x=2, 3, ..., Floor[$\sqrt{91}$]. To do so, we use command For [start, test, incr, body] executes start, then repeatedly evaluates body and incr until test fails to give True.

For[x=2, x≤ Floor [$\sqrt{n}$], x++, y=N[n/x, 5]; Print[" x = ", x ," , " ," y = " , y]]

x = 2 , y = 45.500

x = 3 , y = 30.333

x = 4 , y = 22.750

x = 5 , y = 18.200

x = 6 , y = 15.167

x = 7 , y = 13.000

x = 8 , y = 11.375

x = 9 , y = 10.111

Therefore when n = 91, we get an integer number only when x = 7 and y = 13 from the output above. This implies that x and y are primes.

Example 3.4.3: Factor 1253 by using trial division.

Clear[x, y]

First, we assign 1253 as the value of n.

n=1253

1253

Then, we determine $x|1253$ for x=2, 3, ... , Floor[$\sqrt{1253}$]. To do so, we use command For [start, test, incr, body] executes start, then repeatedly evaluates body and incr until test fails to give True.

For[x=2, x≤ Floor [$\sqrt{n}$], x++, y=N[n/x, 5]; Print[" x = ", x , " , " , " y = " , y]]

x = 2 , y = 626.50

x = 3 , y = 417.67

x = 4 , y = 313.25

x = 5 , y = 250.60

x = 6 , y = 208.83

x = 7 , y = 179.00

x = 8 , y = 156.63

x = 9 , y = 139.22

x = 10 , y = 125.30

x = 11 , y = 113.91

x = 12 , y = 104.42

x = 13 , y = 96.385

x = 14 , y = 89.500

x = 15 , y = 83.533

x = 16 , y = 78.313

x = 17 , y = 73.706

x = 18 , y = 69.611

x = 19 , y = 65.947

x = 20 , y = 62.650

x = 21 , y = 59.667

x = 22 , y = 56.955

x = 23 , y = 54.478

x = 24 , y = 52.208

x = 25 , y = 50.120

x = 26 , y = 48.192

x = 27 , y = 46.407

x = 28 , y = 44.750

x = 29 , y = 43.207

x = 30 , y = 41.767

x = 31 , y = 40.419

x = 32 , y = 39.156

x = 33 , y = 37.970

x = 34 , y = 36.853

x = 35 , y = 35.800

Therefore when n = 1253, the outputs are integer only when x = 7 and y = 179. This implies that x and y are primes.

Example 3.4.4: Factor 34675 by using trial division.

Clear[x, y]

First, we assign 34675 as the value of n.

n=34675

34675

Then, we determine $x|34675$ for x=2, 3, ... , Floor[$\sqrt{34675}$]. To do so, we use command For [start, test, incr, body] executes start, then repeatedly evaluates body and incr until test fails to give True.

For[x=2, x≤ Floor [$\sqrt{n}$], x++, y=N[n/x, 5]; Print[" x = ", x , " , " , " y = " , y]]