A New Method of Factoring Large Integers

Shahid Nawaz

e-mail: snafridi@gmail.com

Abstract In this paper, we reduce a large integer N to an integer N', which has a smaller number of decimal digits than N. Then we find the greatest common divisor (gcd) of N and N' to return a nontrivial factor of N.

Keywords: Factoring, Python, algorithm, running time.

1 Introduction

The branch of mathematics that deals with integers is called number theory. Integers of particular interest are natural numbers which are given by the set $N = \{1, 2, 3, \ldots\}$. The elements of this set are mainly divided into two parts: prime numbers and composite numbers. Prime numbers can only be divided by 1 and itself. They are given by the set $\{2, 3, 5, 7, \ldots\}$. On the other hand, composite numbers have more than two divisors. They are given by the set $\{4, 6, 8, 9, \ldots\}$. Normally a composite number is factored into smaller numbers $N = n_1 n_2$, where the factors n_1 and n_2 may be again composite numbers. The goal is to express every positive number greater than 1 in terms of prime numbers, which is actually the statement of the fundamental theorem of arithmetic (FTA). According to FTA, one can write $N = p_1 p_2 \ldots p_n$, where the prime factors p_i's are not necessarily distinct. Although FTA says that prime factors are possible, it does not tell us how to obtain them. This is one of the hardest problems in mathematics and computer science to factor a large number.

There are several factoring methods. The classical one is the trial division. Let N is to be factored. Divide N by all primes starting from 2 upto $\sqrt{N}$. In this way, the integer would eventually be factored. This method may work well for small numbers, but very slow for large number. Modern methods, however, use the fact that every odd composite number can be written as the difference of two squares [1], [2]. The trick is to find two integers x and y such that $x \not\equiv y($ mod $N)$, but $x^2 \equiv y^2 (\text{mod } N)$. Then $\gcd(x - y, N)$ and $\gcd(x + y, N)$ are the non-trivial factors of N[6].

The difference of square fact is used in several modern factoring methods. The most successful methods are as follows. The continued fraction method of Morrison and Brillhart [3], J.M. Pollard's Rho method [4], the quadratic sieve of Pomerance [1], the elliptic curve method of Lenstra [5], and the general field sieve method, see for example, [6].

This paper is based on the following conjecture. But first some notation in order. The greatest common divisor of integers a and b is denoted by $\gcd(a, b)$. The integer function $\lfloor z \rfloor$ means the greatest integer smaller than or equal to z.

Conjecture 1. *Let N be a composite number and let m be the number of digits in N. If m is odd, then* $\gcd(N', N)$ *returns at least one non-trivial factor, where* $N' = \lfloor \frac{N}{10^{\lfloor m/2 \rfloor}} \rfloor - x$, *for some x in* $\{0, 1, 2, \ldots, 10^{\lfloor m/2 \rfloor}\}$.

The paper is organized as follows. In section 2 we introduce the background of conjecture 1. In section 3 we provide our algorithm. In section 4 we provide the experimental results while in section 5 we calculate the running time of our algorithm.

2 Background of conjecture 1

Let N be an integer in base-10 given by

$$N = 10^n a_n + 10^{n-1} a_{n-1} + \ + 10a_1 + a_0, \tag{1}$$

where $a_i \in \{0, 1, 2, \ldots, 9\}$ are the digits, with $a_n \neq 0$. Further

$$N = XY, \tag{2}$$

where X and Y are the factors which are not necessarily primes. The factors X and Y can also be expressed in base-10 as follows

$$X = 10^k g_k + 10^{k-1} g_{k-1} + \ldots + 10g_1 + g_0, \tag{3}$$

and

$$Y = 10^l h_l + 10^{l-1} h_{l-1} + \ldots + 10h_1 + h_0. \tag{4}$$

Now we write equations (3) and (4) as

$$X = 10^{j+1} x + 10^j g_j + \ldots + 10g_1 + g_0, \tag{5}$$

where all the higher order terms of X are absorbed in the definition of x. We further simplify equation (5)

$$X = 10^{j+1} x + r, \tag{6}$$

where

$$r = 10^j g_j + \ldots + 10g_1 + g_0. \tag{7}$$

Similarly

$$Y = 10^{j+1} y + s, \tag{8}$$

with

$$s = 10^j h_j + \ldots + 10h_1 + h_0. \tag{9}$$

Use equations (6) and (8) in equation (2)

$$(10^{j+1} x + r)(10^{j+1} y + s) = N, \tag{10}$$

where on the left are the factors of N. After a few steps we get

$$10^{j+1}y = \frac{N - 10^{j+1}xs - rs}{10^{j+1}x + r}, \tag{11}$$

where the factor in the denominator on the right is nothing but X. Bring it to the left so that

$$10^{j+1}yX = N - 10^{j+1}xs - rs, \tag{12}$$

We observe

$$N = XY \equiv rs \equiv 10^j a_j + \ldots + 10a_1 + a_0 \pmod{10^{j+1}}, \tag{13}$$

where base-10 expansion of r and s are given by equations (7) and (9) respectively. Without loss of generality[1], we can set $s = 1$, this implies that $h_j = \ldots = h_1 = 0$, and $h_0 = 1$. It gives $g_j = a_j, \ldots, g_0 = a_0$. Use this result in equation (12) and also write N in the base-10 expansion, which is given by equation(1), we have

$$\begin{aligned} 10^{j+1}yX &= 10^n a_n + \ldots + 10^{j+1}a_{j+1} + 10^j a_j + \ldots + 10a_1 + a_0 - 10^{j+1}x \\ &\quad -10^j a_j - \ldots - 10a_1 - a_0. \end{aligned} \tag{14}$$

Upon simplification we get

$$\begin{aligned} 10^{j+1}yX &= 10^n a_n + \ldots + 10^{j+1}a_{j+1} - 10^{j+1}x \\ &= 10^{j+1}\left\lfloor \frac{N}{10^{j+1}} \right\rfloor - 10^{j+1}x \end{aligned} \tag{15}$$

Or

$$yX = \left\lfloor \frac{N}{10^{j+1}} \right\rfloor - x\,. \tag{16}$$

Experimentally we observe that

$$j + 1 = \frac{n}{2} = \left\lfloor \frac{m}{2} \right\rfloor, \tag{17}$$

where $m = n + 1$ is the number of digits in N. To correctly obtain the factor X, we observe that m must be an odd number and the data set x must take values in $\{0, 1, 2, \ldots, 10^{\lfloor \frac{m}{2} \rfloor}\}$. That is

$$\gcd\left(N, \left\lfloor \frac{N}{10^{\lfloor m/2 \rfloor}} \right\rfloor - x\right) = X\,, \tag{18}$$

for some x in $\{0, 1, 2, \ldots, 10^{\lfloor \frac{m}{2} \rfloor}\}$. Thus we arrive at conjecture 1

Remark 1. Equation (18) can also be used for primality test. If X turns out to be one for all values of x, then N is a prime number.

Remark 2. Since conjecture 1 requires N to have odd number of digits, if the number of digits are even, then multiply N by a small prime number say, 3 or higher so that the new N has odd number of digits, then apply conjecture 1 again.

[1] When $s \neq 1$, the data set x becomes xs and we arrive at the same final result i.e., equation (18).

3 Factoring algorithm

Here is our algorithm that will be used to factor the integer N.

- **Input:** integer N
- **Output:** factor X
- Find the number of digits in N and call it m.
- Calculate $\left\lfloor \frac{N}{10^{\lfloor m/2 \rfloor}} \right\rfloor$
- Create an array $\{0, 1, 2, \ldots, 10^{\lfloor m/2 \rfloor}\}$ and call it x
- Create another array call it N', where $N' = \left\lfloor \frac{N}{10^{\lfloor m/2 \rfloor}} \right\rfloor - x$
- **for** item **in** N':
  if $\gcd(N, N') > 1$ % do not return trivial factors.
  return $\gcd(N, N')$ % return the nontrivial factor X

4 Experimental results

In this section, we include several experimental results. First, we consider integers that have odd number of decimal digits and then we will also consider integers which have even number of digits. For all cases, the code is presented below. Only the value of N is changing for each case. In case, if we are dealing with the integers that have even number of digits, we might add an additional line in the code.

Here is our code written in Python. In the first example, we start with a three-digit number, say, $N = 377$. When we run the code, it returns $X = 29$ as a factor. The other factor can be obtained by dividing 377 by 29. Thus $377 = 29 \cdot 13$.

```
import math
N = 377
m = len('377')
m = math.floor(m/2)
N1 = math.floor(N/(10**m))
import numpy as np
array1 = np.arange(0,10**m+1)
array1 = N1-array1
for item in array1:
    if math.gcd(N, item)>1:
        print(math.gcd(N, item))
```

Next, we take a five-digit number, say, $N = 10337$. In this case, when we run the code, it returns no value which means that 10337 is a prime number.

Let us take a slightly different number, say, $N = 10339$, which returns 49 as a factor. So $10339 = 49 \cdot 211$.

Let us move on to a seven-digit number, say, $N = 2737859$, We get $X = 433$. Thus $2737859 = 433 \cdot 6323$. Let us now take a 13-digit number $N = 4199954106281$, We get $X = 23327$. So that $4199954106281 = 23327 \cdot 180046903$.

Unfortunately, my personal computer is not capable to test beyond 13-digit, because the program halts.

Next we consider an integer of even number of digits, say, $N = 307997$. Since 307997 is a six-digit number, multiply by 7 to make it a seven-digit number. The new number is $M = 7 \cdot 307997 = 2155979$. We make some changes in our code given above. We replace line-10 by "if math.gcd(N, item)>7:" In this case our nontrivial factors are 1 and 7. Here is the updated code:

```
import math
N = 2155979
m = len('2155979')
m = math.floor(m/2)
N1 = math.floor(N/(10**m))
import numpy as np
array1 = np.arange(0,10**m+1)
array1 = N1-array1
for item in array1:
    if math.gcd(N, item)>7:
        print(math.gcd(N, item))
```

We at once get $X = 643$ and $X = 479$, so that $M = 2155979 = 7 \cdot 643 \cdot 479$. Since our original number is $N = 307997$. Thus $N = 307997 = 479 \cdot 643$.

5 Time complexity of the algorithm

In this section we calculate the time complexity (running time) of our algorithm developed in section 4. It should be noted that running time of an algorithm is not the physical time of the clock but it is the number of operations by a program to return a result.

In our case it is not difficult to estimate the the running time as there is only one "for loop" involved. To estimate the running time, one uses Big-O notation. First, we quickly introduce the Big-O notation, see for example [7].

$$\text{If} \quad |f(x)| \leq Cg(x), \qquad \text{we write} \qquad f(x) = O(g(x)),$$

we say that $f(x)$ is big oh of $g(x)$. The notation tells us the growth (or shrink) rate of a function as $x \to \infty$ or $x \to 0$.

Now we return to our algorithm. We notice that there is only one "for loop" which takes values in the interval $[N', 10^{\lfloor m/2 \rfloor}]$. As far as the growth rate is concerned we can take the interval as $[0, 10^{\lfloor m/2 \rfloor}]$. Thus, the running time is $O(10^{\lfloor m/2 \rfloor})$. One might observe that exponential running time is slow for large

number, however, here m is the number of decimal digits in an integer. The actual number to be factored is N. It is tempted to find a relation between m and N. We obtain this relation in the following theorem. In what follows $\log z$ would mean $\log z$ to the base 10.

Theorem 1. *Let N be a positive integer and m be the number of digits in N, then*

$$\sqrt{\frac{N}{10}} < 10^{\lfloor m/2 \rfloor} \leq \sqrt{10N}\,. \tag{19}$$

Proof. One can note that

$$m = \lfloor \log N \rfloor + 1\,, \tag{20}$$

or

$$\lfloor m \rfloor = \lfloor \log N + 1 \rfloor\,. \tag{21}$$

The result is unaffected if we insert a factor of $1/2$ in the above equation, i.e.,

$$\left\lfloor \frac{m}{2} \right\rfloor = \left\lfloor \frac{\log N + 1}{2} \right\rfloor\,. \tag{22}$$

So that

$$\begin{aligned} 10^{\lfloor m/2 \rfloor} &= 10^{\left\lfloor \frac{\log N+1}{2} \right\rfloor} \\ &= 10^{\frac{\log N+1}{2}} \cdot 10^{-\{\frac{\log N+1}{2}\}}\,, \end{aligned} \tag{23}$$

where we have used

$$z = \lfloor z \rfloor + \{z\}\,. \tag{24}$$

where z is a generic number. Note that

$$0 \leq \{z\} < 1\,, \tag{25}$$

where $\{z\}$ is the decimal part of z. It means

$$1 \leq 10^{\{z\}} < 10\,, \tag{26}$$

or

$$\frac{1}{10} < 10^{-\{z\}} \leq 1 \Rightarrow \frac{10^z}{10} < 10^z \cdot 10^{-\{z\}} \leq 10^z\,. \tag{27}$$

Set

$$z = \frac{\log N + 1}{2}\,. = \log N^{1/2} + \frac{1}{2} \tag{28}$$

In order to simplify equation (23), we also need to note

$$10^{\log N^{1/2}} = N^{1/2}\,. \tag{29}$$

Collecting the results obtained from equations (23) through (29), we arrive at the final result after a few steps

$$\sqrt{\frac{N}{10}} < 10^{\lfloor m/2 \rfloor} \leq \sqrt{10N}\,. \tag{30}$$

□

We can write inequality (30) in terms of Big-O notation, where the left inequality implies

$$\sqrt{\frac{N}{10}} = O(10^{\lfloor m/2 \rfloor}), \tag{31}$$

and the right inequality implies

$$10^{\lfloor m/2 \rfloor} = O(\sqrt{10N}) \tag{32}$$

The two estimates (31) and (32)give us the same information which is actually the running time of our algorithm expressed in terms of the integer N to be factored.

Acknowledgment: I would like to thank M.A. Mubeen, A. Raza, and K. Khan for valuable discussion on coding. My special thanks also go to F. Ghafoor for critically reading the manuscript.

References

[1] C. Pomerance, "A Tale of Two Sieves," The Notices of Amer. Math. Soc., **43**, (1996) 1473–1485.

[2] R. Crandall and C. Pomerance, "Prime Number: A computation perspective," 2nd edition (Springer, New York, 2005)

[3] M. A. Morrison, and J. Brillhart, "Method of Factoring and the Factorization of F_7," Mathematics of Computation, **29**, (1975) 183–205.

[4] J. M. Pollard, "A Monte Carlo Method for Factrization," BIT, **15**, (1975) 331–334.

[5] H.W. Lenstra, J. "Factoring Integers with Elliptic Curves." Annals of Mathematics, **126**,(1987) 649-673.

[6] M. Briggs, "An Introduction to the General Number Field Sieve." Master thesis, (Virginia, 1998).

[7] T.M. Apostol, "Introduction to Analytic Number Theory," (Springer, New York, 1976)