Algorithm Available for Factoring Big Fermat Numbers

Xingbo Wang[1,2,3*]

[1]Department of Mechatronic Engineering, Foshan University, Foshan City, PRC, 528000, China.
[2]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China.
[3]Guangdong Engineering Center of Information Security for Intelligent Manufacturing System, China.

* Corresponding author. Tel.: +86075782988845; email: xbwang@fosu.edu.cn; dr.xbwang@qq.com
Manuscript submitted March 12, 2020; accepted April 19, 2020.
doi: 10.17706/jsw.15.3.86-97

Abstract: The paper proves that an odd composite integer N can be factorized in at most $O(0.125u(\log_2 N)^2)$ searching steps if N has a divisor of the form $2^D u+1$ or $2^D u-1$ with $D>1$ being a positive integer and $u>1$ being an odd integer. Theorems and corollaries are proved with detail mathematical reasoning. Algorithms to factorize the kind of odd composite integers are designed and tested with certain Fermat numbers. The results in the paper might be helpful to factorize certain big Fermat numbers.

Key words: Algorithm, integer factorization, fermat number, cryptography.

1. Introduction

Fermat numbers are integers of the form $F_m = 2^{2^m}+1$ with integer $m \geq 1$. This kind of integers has been a research issue ever since they had come into being for Prof. Wilfrid Keller has set up a website to introduce the progress of the researches on the numbers, as seen in [1]. Factoring a Fermat number like factoring any other bigger integer is a hard problem in number theory as well as cryptography. Although there are several algorithms to factorize a composite number, as overviewed in [2] and [3], new and efficient methods are still in need. It is known that the divisor of a Fermat number is of the form $2^D u+1$. Accordingly, algorithm that is able to factorize an odd integer with such forms of divisors is surely worthy of study. Paper [4], a very recent publication, proved that an odd composite integer N could be factorized in $O((\log_2 N)^4)$ bit operations if $N=pq$ with $q=2^D u+1$ and $r=p+2^D-1$ or 2^D-1-p and $2^{D+1}-1$, where u is an odd integer and D is a positive integer. This result might be marvelous for factoring the kind of N. However, the algorithm designed in the paper [3] is not able to factorize the Fermat numbers because no Fermat number is able to fit the conditions set for p and q. Hereby, this paper investigates the case $2^{D+E}+1$ and $2^{D+E-1}-1$ with E being a positive integer to find an algorithm that is available for factoring the Fermat numbers. By mathematical reasoning, an algorithm is found and it is proved that the algorithm is theoretically able to factorize very fast any F_m with $m>100000$. Experiments on certain Fermat numbers with Maple mathematical software show that the new algorithm is fast indeed.

The contents include five main parts: section 2 presents certain preliminaries, section 3 shows main results including, theorems and corollaries plus algorithms for each case, section 4 presents numerical

experiments on factoring the related Fermat numbers, section 5 points out the advantage of the algorithms and the barrier to apply thealgorithms as well as the possible future attempts.

2. Preliminaries

2.1. Definitions and Notations

In this whole paper, the symbol x«¬ denotes the floor function, an integer function of the real number x such thatx 1 x«¬ dx or equivalently x ¼ dx x«¬1. Symbol {x} x x«¬is the fractional part of x. Symbol A Ÿ Bmeans resultB is derived from condition A or A can derive B out. Symbol GCD is to express the greatest common divisor of integers a and b

2.2. Lemmas

Lemma 1 (see in [5]). Properties of the floor functions with real numbers x, y and integer n

(P1) x ¼ y«¬ d x ¼ y¬ d x »¼ y ¬1

(P2) x ¼ y«¬1» d x ¼ y¬ d x »¼ y ¬» x«¬ ¬y 1« » ¬ x«¬ »¬ ¼

(P13) x dy Ÿ x d y‹

(P14) x rn ¼ x«¬r n.

(P32) Dx ¼ 1 2¬ x(Dx 1); particularly n x d nx¬ d(¼x 1)¬ 1, Taking n=2 yields 2 x d 2x¬ ¬2 x«¬d 1 ¼

Lemma 2. (See in [6]). Odd integer N > 1 satisfies2 log2 N ¼ 1 dN d2 log2 N ¼ 1 .

3. Main Results

3.1. Math Foundations

Theorem 1. Let N pq be an odd integer with p and q being odd integers bigger than 1; supposeq 2Du 1and 2D E 1 dp d2D E 1 with D, Ebeing positive integers and u !1 being an odd integer; then

$$2 \quad D \quad 1 \; dE \log_2 N\!«¬$$

Proof . The conditions u t3 and q 2Du 1yield

$$N \quad (2^D u \; 1)p \quad 2^D up \quad p \quad 2^D(u \; \frac{1}{2^D})p$$

$$Ÿ 2^{2D \; E}(u \; \frac{1}{2}) \quad N \quad 2^{2D \; E1}(u \; \frac{1}{2^D})$$

$$Ÿ 2^{2D \; E}u3 \quad N \quad 2^{2D \; E1}(u \; \frac{1}{2^D})$$

$$Ÿ 2D \quad E1 \quad \log_2 N \quad 2D \quad E1 \quad \log_2(u \; \frac{1}{2^D})$$

$$Ÿ 2 \quad D \quad 1 \; dE \log_2 N\!«¬ \quad ¼$$

Similarly, the conditions u t3 andq 2Du 1yield

$$N = (2^D u - 1)p = 2^D up - p = 2^D(u - \frac{1}{2^D})p$$
$$= 2^{2D-E}(u - \frac{1}{2}) N = 2^{2D-E}(u - \frac{1}{2^D})$$
$$= 2^{2D-E}u2 N = 2^{2D-E}(u - \frac{1}{2^D})$$
$$= 2D - E1 \log_2 N = 2D - E1 \log_2(u - \frac{1}{2^D})$$
$$= 2 D 1 d\log_2 N\rfloor$$

Remark 1. During the reasoning process, the properties list in Lemma 1 are frequently applied, and so they are with the following reasoning.

Proposition 2. Let $N = pq$ be an odd integer with p q being odd integers bigger than 1; suppose $q = 2^D u - 1$ and $2^{D-E} + 1 = p$ or $2^{D-E} - 1$ with D, E being positive integers and $u \neq 1$ being an odd integer; then

$$D = \lfloor \frac{\log_2 N}{2} \rfloor - 1$$

Proof. We consider two cases: $q > p$ and $q < p$. For the case $q > p$, it follows

$$q > p \Rightarrow q > \sqrt{N} > p \Rightarrow 2^D < D + E \log_2 \sqrt{N} < D + E = \lfloor \frac{\log_2 N}{2} \rfloor$$

Since $E > 1$, it follows

$$\lfloor \frac{\log_2 N}{2} \rfloor - 1 < D < \lfloor \frac{\log_2 N}{2} \rfloor - 1$$

For the case $q < p$, consider the case $q = 2^D u - 1$; then

$$q < p \Rightarrow q < \sqrt{N} \Rightarrow 2^D < \frac{\sqrt{N} + 1}{u} < \frac{\sqrt{N}}{3}$$
$$\Rightarrow D < \frac{1}{2}\log_2 N - \log_2 3 < \frac{1}{2}\log_2 N - 1$$
$$\Rightarrow D = \lfloor \frac{\log_2 N}{2} \rfloor - 1$$

Likewise, the case $q = 2^D u + 1$ yields

$$q < p \Rightarrow q < \sqrt{N} \Rightarrow 2^D < \frac{\sqrt{N} - 1}{u} < \frac{\sqrt{N} - 1}{3}$$
$$\Rightarrow D < \log_2(\sqrt{N} - 1) - \log_2 3 < \frac{1}{2}\log_2 N - \log_2(1 - \frac{1}{\sqrt{N}}) - \log_2 3$$

Since $0 < \log_2(1 - \frac{1}{\sqrt{N}}) < 1$ and $1.5 < \log_2 3 < 2$, it knows

$$\frac{1}{2}\log_2 N - 2 < \frac{1}{2}\log_2 N - \log_2(1 - \frac{1}{\sqrt{N}}) - \log_2 3 < \frac{1}{2}\log_2 N - 0.5$$
$$\Rightarrow D = \lfloor \frac{\log_2 N}{2} \rfloor - 1$$

Remark 2. The case $q = 2^D u - 1$ actually yields $D = \lfloor \frac{\log_2 N}{2} \rfloor - 2$ because $D < \frac{1}{2}\log_2 N - \log_2 3 <$ $D = \lfloor \frac{\log_2 N}{2} \rfloor - 2$.

Theorem 2. Let N pq be an odd integer with p and being odd integers bigger than 1; supposeq $2^D u$ 1and $2^{D\ E}$ 1 dp d$2^{D\ E1}$ 1 with D, Ebeing positive integers and u !1 being an odd integer; then

$$\left(\frac{u}{N}\right) = \left(\frac{N \bmod 2^{D+2E}}{2^{D+2E}-u}\right)\left(\frac{u}{2}\right)^{D+2E}$$

Proof. See the following reasoning process, respectively.

$$\begin{aligned}
&N = (2^D u - 1)p = 2^D up - p \\
&\Rightarrow 2^D u(2^{D+E}-1) - (2^{D+E}-1) = N = 2^D u(2^{D+E}-1) - (2^{D+E}-1) \\
&\Rightarrow 2^{2D+E}u - 2^D u - 2^{D+E} = N = 2^{2D+E}u - 2^D u - 2^{D+E} \\
&\Rightarrow 2^{2D+E}u - 2^{D+E} = N = 2^{2D+E}u - 2^{D+E} - 2^{2D+E}u - 2^{D+E} \\
&\Rightarrow 2^{2D+E}\left(u - \frac{1}{2}\right) = N = 2^{2D+E}\left(u - \frac{1}{2^D}\right) \\
&\Rightarrow \frac{1}{4}\left(u - \frac{1}{2^D}\right) = \frac{N}{2^{2D+2E}} = \frac{1}{2}\left(u - \frac{1}{2^D}\right) \\
&\Rightarrow \left(\frac{u}{N}\right)\left(\frac{u}{4}\right) = \left(\frac{1}{2^{D+2}}\right) = \left(\frac{N}{2^{2D+2E}}\right)\left(\frac{u}{2}\right) = \left(\frac{1}{2}\right)\left(\frac{u}{2}\right) \\
&\Rightarrow \left(\frac{u}{N}\right) = \left(\frac{N}{2^{D+2E}-u}\right)\left(\frac{u}{2}\right)^{D+2E}
\end{aligned}$$

Theorem 3. Let N pqbe an odd integer with p and q being odd integers bigger than 1; suppose q $2^D u$ 1 and $2^{D\ E}$ 1 dp d$2^{D\ E1}$ 1 with D, Ebeing positive integers and u !1 being an odd integer; then

$$\left(\frac{u}{N}\right) = \left(\frac{N \bmod 2^{D+2E}}{2^{D+2E}-u}\right)\left(\frac{u}{2}\right)^{D+2E}$$

Proof. See the following reasoning processes.

$$\begin{aligned}
&N = (2^D u - 1)p = 2^D up - p \\
&\Rightarrow 2^D u(2^{D+E}-1) - (2^{D+E}-1) = N = 2^D u(2^{D+E}-1) - (2^{D+E}-1) \\
&\Rightarrow 2^{2D+E}u - 2^D u - 2^{D+E} = N = 2^{2D+E}u - 2^D u - 2^{D+E} \\
&\Rightarrow 2^{2D+E}u - 2^{D+E} = N = 2^{2D+E}u - 2^{D+E} - 2^{2D+E}u - 2^{D+E} \\
&\Rightarrow 2^{2D+E}\left(u - \frac{1}{2^1}\right) = N = 2^{2D+E}\left(u - \frac{1}{2^{D-1}}\right) \\
&\quad \frac{1}{4}(u - \frac{1}{2^{D-1}}) = \frac{N}{2^{2D+2E}} = \frac{1}{2}\left(u - \frac{1}{2^{D-1}}\right) \\
&\Rightarrow \left(\frac{u}{N}\right)\left(\frac{u}{4}\right) = \left(\frac{1}{2^{D-1}}\right) = \left(\frac{N}{2^{2D+2E}}\right)\left(\frac{u}{2}\right) = \left(\frac{1}{2}\right)\left(\frac{u}{2}\right) \\
&\Rightarrow \left(\frac{u}{N}\right) = \left(\frac{N}{2^{D+2E}-u}\right)\left(\frac{u}{2}\right)^{D+2E}
\end{aligned}$$

3.2. Algorithms for General Cases

Corollary 1. Let N pqbe an odd integer with 1 q p being odd integers; supposeq $2^D u$ 1 and $2^{D\ E}$ 1 dp d$2^{D\ E1}$ 1 with D>1, E t1 being integers and u > 1 being an old integer; then N can be factorized at most $O(\frac{u}{8}(\log_2 N)^2)$ searching steps.

Proof. First, it follows when D>1 and Ɛ t1

$$N \quad 2^{D}up \quad p\ddot{Y}u \quad \frac{N \quad p}{2^{D}p} \quad \frac{N}{2^{D}p} \quad \frac{1}{2^{D}}$$

$$\ddot{Y}\frac{N}{2^{2D\,E1.}} \quad \frac{1}{2^{D}}u \quad \frac{N}{2^{2D\,E}} \quad \frac{1}{2^{D}}$$

$$\ddot{Y}\frac{N\ 1}{2^{2D\,E1.}} \quad \frac{1}{2^{2\ D\ 1E}} \quad \frac{1}{2^{\ D}}u \quad \frac{N\ 1}{2^{2D\,E}} \quad \frac{1}{2^{2\ D\ E}}\frac{1}{2^{\ D}}$$

$$\ddot{Y}\frac{«N\ 1}{2^{2D\,E1.}}\Big»_{¼}1\ du\ d\frac{N«1}{2^{2«D\,E}}\Big»_{¼}$$

Let

$$n_u \quad \frac{\frac{«N\ 1}{2^{2D\,E}}\Big»_{¼}\left(\frac{N«1}{2^{2«D\,E1.}}\right)1\Big»_{¼}1}{2}$$

Then n_u is the number of the odd integers that u possibly takes. Since

$$\frac{«N\ 1}{2^{2D\,E1.}}\Big»_{¼}d\frac{N«1}{2^{2«D\,E}}\Big»_{¼}\frac{N«1}{2^{2«D\,E1.}}\ d\frac{N«1}{2^{2«D\,E1.}}\ ¬\ 1$$

it holds

$$\frac{1}{2}\frac{«N\ 1}{2^{2D\,E1.}}\Big»_{¼}1\ dn_u\ d\frac{1}{2}\frac{«N\ 1}{2^{2D\,E1.}}\Big»_{¼}1\ \ 1$$

and thus

$$\frac{«N\ 1}{2^{2D\,E2}}\Big»_{¼}1\ \frac{1}{2}\ n_u\ \frac{«N\ 1}{2^{2D\,E2}}\Big»_{¼}2\ \ 1$$

That is

$$\frac{«N\ 1}{2^{2D\,E2}}\Big»_{¼}1\ dn_u\ d\frac{N«1}{2^{2«D\,E2}}\Big»_{¼}2$$

By Theorem 2,

$$\frac{«u}{4}\Big»_{¼}d\frac{N«1}{2^{2«D\ 2E}}\ d\frac{»u}{2}\Big»_{¼}\frac{«1}{¬}$$

It results in

$$\frac{«u}{4}\Big»_{¼}1\ dn_u\ d\frac{u«3}{2«¬}\Big»_{¼}$$

This finishes proving that there are at most $\frac{u«}{4«¬}$ 2 searching steps to find a u_0 that fits q $2^{D}u_0$ 1 around $\frac{N\ 1}{2^{2\ D\ 2}«¬}$. Now consider the ranges of $\mathcal{L}$ and Ɛ in algorithm design. By Theorem 1 and Proposition 1,

$$2 \quad D \quad 1\ dƐ\log_2 N«, \text{ and } \quad D \quad \frac{\log_2 N«}{2}«d\ 1$$

It seems that $\log_2 N$ might be taken to be the maximal value of $2D-1$. However this does not work because by Lemma 2

$$2^{\log_2 N} \equiv 1 \pmod{N} \Rightarrow 2^{\log_2 N} - 1 \equiv 0 \pmod{N}$$

which leads to

$$\gcd\left(N, 2^{\frac{\log_2 N}{2}} - 1\right) = p \text{ or } q \text{ and } \gcd\left(N, 2^{\frac{\log_2 N}{2}+1} - 1\right) = N$$

Accordingly, it is suitable to take $\log_2 N - 2$ to be the maximal value of $2D-1$ and then the condition that odd integer $u > 1$ can be ensured. Consequently, an algorithm is designed as follows

Procedure 1 SearchProc1

```
1: Comment: Factorization by Calculating GCD
2: Input Parameters:N;
3: Begin
4: Calculate b = log2 N - 2, a = log2 N / 2 - 1;
5: for i from b downto 3 do
6:   for u from 1 to r step 2 do
7:     for j from 2 to a do
8:       Calculate l = (N - 1) / 2^i + 1, r = N - 1 / 2^(i-1);
9:       Calculateg = gcd(N, 2^j u - 1);
10:        if (g > 1) then return g;end if
11:      end for j;
12:    end for u;
13: end for i
14: End proc
```

Obviously, the number of total searching steps is at most $O\left(\frac{u}{4} \cdot \frac{\log_2 N}{2} \cdot (\log_2 N - 2)\right) = O\left(\frac{u}{8} \log_2^2 N\right)$.

Remark 3. When u is relatively larger, we can choose a smaller b in the procedure by

$$2 \le \gcd\left(N, 2^{\frac{\log_2 N}{2}+F-1} - 1\right) < N$$

For example, if we can first estimate that $u < 2^F$, we can choose $b = \log_2 N - F$. By such means, a lot of computing time is saved.

Corollary 2. Let $N = pq$ be an odd integer with $q > p$ being odd integers; suppose $q - 2^D u - 1$ and $2^{D+E} - 1 < p < 2^{D+E} + 1$ with $D > 1$, $E \ge 1$ being integers and $u > 1$ being an old integer; then N can be factorized at most $O\left(\frac{u}{8}(\log_2 N)^2\right)$ searching steps.

Proof. First, it yields when $D > 1$ and $E \ge 1$

$$N \quad 2^{D}up \quad p\ddot{Y}u \quad \frac{N \quad p}{2^{D}p} \quad \frac{N}{2^{D}p} \quad \frac{1}{2^{D}}$$

$$\ddot{Y}\frac{N}{2^{2D\ E1}} \quad \frac{1}{2^{D}}u \quad \frac{N}{2^{2D\ E}} \quad \frac{1}{2^{D}}$$

$$\ddot{Y}\frac{N\ 1}{2^{2D\ E1}} \quad \frac{1}{2^{2\ D\ 1E}} \quad \frac{1}{2^{\ D}}u \quad \frac{N\ 1}{2^{2D\ E}} \quad \frac{1}{2^{2\ D\ E}}\frac{1}{2^{\ D}}$$

$$\ddot{Y}\frac{«N\ 1»}{«2^{2D\ E1}»}du\ d\frac{N«\ 1»}{2^{2«D\ E»}}\ 1$$

Let

$$\eta_u \quad \frac{\frac{«N\ 1»}{«2^{2D\ E»}}1 \quad \frac{N«\ 1»}{2^{2«D\ E1»}}}{2}$$

Then referring to the proof of Corollary 1, it knows

$$\frac{«N\ 1»}{«2^{2D\ E2»}}1\ d\eta_u\ d\frac{N«\ 1»}{2^{2«D\ E2»}}$$

By Theorem 3

$$\frac{«u»}{«4»}1\ d\ \frac{N«\ 1»}{2^{2«D\ 2E»}}\ du\frac{«\ »}{«\neg»}$$

Next is referring to the proof of Corollary 1. Accordingly, an algorithm is designed as follows

Procedure 2 SearchProc2

1: Comment: Factorization by Calculating GCD
2: Input Parameters:N;
3: Begin
4: Calculate $b \quad \log_2 N«,\ 2$, $a \quad \frac{\log_2 N«}{2\ «\neg}\ 1$;
5: for i from b downto 3 do
6: Calculate $l \quad \frac{«N\ 1»}{«\neg\ 2^{i}»}, r \quad \frac{N«\ 1»}{2^{«\neg 1»}}$
7: for u from l to r step 2 do
8: for j from 2 to a do
9: Calculateg = gcd(N, $2^{j}u\ \ 1$);
10: if (g > 1) then return g;end if
11: end for j;
12: end for u;
13: end for i
14: End proc

Referring to the analysis of Corollary 1, it knows Corollary 2 holds.

Theorem 4 . Odd composite integer N can be factorized in at most $O(0.25u(\log_2 N)^2)$ searching steps if N = pq with $q \quad p$, p satisfies $2^{D\ E}\ 1\ dp\ d2^{D\ E1}\ 1$ with $D>1$, $E\ddot{Y}1$ being positive integers and q is of the form $q \quad 2^{D}u\ \ 1$ or $q \quad 2^{D}u\ \ 1$ with u >1 being an odd integer.

Proof . By Theorem 2 and Theorem 3, Procedure 1 and Procedure 2 are incorporated into the following Procedure 3 that can factorize N with divisor q being either $q \quad 2^{D}u\ \ 1$ or $q \quad 2^{D}u\ \ 1$.

Procedure 3 SearchProc3

1: Comment: Factorization by Calculating GCD
2: Input Parameters:N;

3: Begin
4: Calculate $b = \log_2 N\rfloor, a = \frac{\log_2 N\rfloor}{2} - 1;$
5: for i from b downto 3 do
6: Calculate $l = \left\lfloor \frac{N-1}{2^i} \right\rfloor + 1, r = \left\lfloor \frac{N-1}{2^{i-1}} \right\rfloor + 1;$
7: for u from l to r step 2 do
8: for j from 2 to a do
9: Calculate $g_1 = \gcd(N, 2^j u - 1);$
10: Calculate $g_2 = \gcd(N, 2^j u - 1);$
13: if $(g_1 > 1)$ then return g_1;end if
14: if $(g_2 > 1)$ then return g_2;end if
15: end for j;
16: end for u;
17: end for i
18: End proc

3.3. Algorithm for Factoring Fermat Numbers

Seen in [1], it is known that every prime divisor of the Fermat number $F_m = 2^{2^m} + 1$ is of the form $2^n u + 1$ with $n \geq m + 2$. Assume a Fermat number has two divisors $p = 2^{m+2+i} k + 1$ and $q = 2^{m+2+j} l + 1$ with i, j being nonnegative integers, $k \geq 1$ and $l \geq 1$ being odd integers (Note: Paper [4] proved that the case $k = 1$ or $l = 1$ was easy to factorize.); it follows

$$
\begin{aligned}
N &= pq = (2^{m+2+i} k + 1)(2^{m+2+j} l + 1) \\
&= 2^{2m+4+i+j} kl + 2^{m+3}(2^i k + 2^j l) + 1 \\
&\Rightarrow \frac{N-1}{2^{2m+4}} = 2^{i+j} kl + \frac{2^i k + 2^j l}{2^{m+1}} \geq 9 \\
&\Rightarrow 2^m < \frac{\sqrt{N-1}}{3} \\
&\Rightarrow n \leq m + 2 \left\lfloor \frac{\log_2(N-1)}{2} - \log_2 3 \right\rfloor \leq \left\lfloor \frac{\log_2(N-1)}{2} \right\rfloor
\end{aligned}
$$

This is a theoretical upper bound form. However, from practical point of view, referring to [1], we take $m = 20$ to be the maximal limit of n and design the following procedure.

Procedure 4 FactFermat
1: Comment: Factorization by Calculating GCD
2: Input Parameters: N, F;
3: Begin
4: Calculate $n = \log_2 \log_2 (N-1)\rfloor, b = \log_2 (N-1)\rfloor, F;$
$a = n - 20;$
5: for i from b downto 3 do
6: for j from n+2 to a do
7: Calculate $l = \left\lfloor \frac{N-1}{2^i} \right\rfloor + 1, r = \left\lfloor \frac{N-1}{2^{i-1}} \right\rfloor + 1;$
8: for u from l to r step 2 do
9: Calculate $g = \gcd(N, 2^j u - 1);$
10: if $(g > 1)$ then return g;end if
11: end do u;
12: end for j;
13: end for i

14: End proc

Remark 4 The parameter F in the procedure is related with the range of u and is used to limit the upper bound of b, as mentioned in Remark 2. Referring to [1], it knows that the range of u is predicted with an upper bound L_n and L_n 7 10^7 when n !12000. This can reduce a lot of search time.

4. Numerical Experiments

We test the designed algorithms in Maple software on a HP personal computer with E5450 CPU and Windows XP OS. The source codes of the programs are list in the appendix section. Due to the limitation of the computer, we factorize some Fermat numbers with relatively small u. The computation results are list in Table 1. It can see that, one divisor is obtained within one second in most cases. The experiments show that, the algorithms acceptably fast.

Table 1. Factorization of Some Fermat Numbers

Item	F	Found Divisor	D	E	u	Searching Steps	Searching time
F5	2	641	7	30	5	3	<5ms
F6	3	274177	8	54	1071	50496	30ms
F9	3	2424833	16	507	37	1462	5ms
F10	3	45592577	12	1011	11131	403468	140ms
F11	3	319489	13	2043	39	1378	5ms
F12	3	114689	14	4093	7	2	<5ms
F15	3	1214251009	21	32759	579	26348	1.2s
F16	3	825753601	19	65526	1575	51261	5.5s
F17	25	31065037602817	19	131047	59251857	12848715	1500s
F18	3	13631489	20	262141	13	5	< 5ms
F19	3	70525124609	21	524273	33629	1606849	1560s
F21	19	4485296422913	23	2097133	534689	5203	9s
F23	2	167772161	25	8388606	5	3	<5ms

5. Conclusion, Discussion & Future Work

5.1. Conclusion

Factoring big integers is both a challenge to and an exploitation of human intelligence. We know that there are many odd integers that have divisors of the form $2^D u$ r1 and factorization of such numbers is still a hard problem in the world. By investigating the intrinsic structures of such odd number, this paper shows that factorization of them is highly u-related: a small u leads to a rapid factorization. The results in this paper might be helpful to factorize certain big Fermat Numbers. For example, referring to [1], when n is bigger than 100001, u 20000. This time it might be easy to factorize n-F.

5.2. Discussion

Seeing from the numerical experiments in section 4, one might be wonder why not to choose some big Fermat numbers to be experimental samples. This involves with a very upset situation: we have no way to compute a bigger Fermat number with a daily-work computer even with the help of the tools dealing with the big integer operations.

We have made experiments by two different means. One is to test with the mathematical software Maple, and the other is to program with the help of the tools that can deal with operations of big integers, e.g., GMP library (The GNU Multiple Precision Arithmetic Library)[7], NTL (A Library for doing Number Theory)[8], MIRACL(Multi-precision Integer and Rational Arithmetic Cryptographic Library) [9] and so on. Unfortunately, neither means could compute a big Fermat number. In Maple software, it says the

computation of F_m is out of the range when $m \geq 23$ while a program that is programmed with the GMP library fails to work when $m \geq 23$. When debugging the GMP library-program in Visual C/C++ 6.0, I find the failure occurs at the step where the function mpz_gcd is called. Solving this problem is obviously out of my capability and I have to hope someone else can solve this problem.

5.3. Future Work

Corollaries 1 and 2 do provide an approach to factorize odd integers with a divisor of the form $2^{\alpha}u+1$. However, seeing the reasoning process of this paper, we have to have more researches on the speeding process for a big u, for example, the F_{29} has $q=116503103764643 \cdot 2^{9}+1$ with a big $u=116503103764643$. This remains the future work. Actually, in our experiments, we have found that, changing the parameters F will lead to different searching steps. For example, when factoring F_{15}, it took more than 340 steps when $F=3$ while it merely took 3 steps when $F=2$. Another example is factoring F_{21} that has a divisor 4485296422913. When taking $F=19$ according to Remark 2 it took 5203 steps (9 seconds) to finish the work while it took 3675387 steps (over 30 minutes) when $F=5$. These phenomena indicate that a speed-up attempt is available. One of our future tasks is trying to solve the problem. And I also hope more young join and succeed.

Conflict of Interest

The author declares no conflict of interest.

Author Contributions

Professor Xingbo Wang contributes 100% work to this paper.

Acknowledgment

The research is supported by the Open Project Program of the State Key Lab of CAD&CG (Grant No. A2002) and by Foshan University and Foshan Bureau of Science and Technology under project that constructs Guangdong Engineering Center of Information Security for Intelligent Manufacturing System.

Appendix Maple Source Codes and Its Running Results

The following screenshot (see Figure A1) shows the testing results to factorize Fermat numbers. The meanings of the output data are as follows:

The first one is the calculated D

The second one is the calculated E

The third one is the calculated u

The fourth one is the calculated $2^{\alpha}u+1$

The fifth one is the calculated divisor q

The last one is the number of searching steps.

It can see that, F_5 takes 3 steps, F_{12} takes 2 steps, F_{18} takes 5 steps, F_{23} takes 3 steps, F_9 takes 319 steps, F_{11} takes 230 steps, F_6, F_{15}, F_{16} and F_{21} take less than 10 thousand steps. F_{17} takes the longest time.

```
FactFermat := proc(N, x)
  local a, b, i, j, u, l, s, r, g, d, r, c, Z;
  c:= 1; # a counter to count steps
  Z:= N-1+0.1e-1;
  b:= floor(log[2](Z)) -x;
  s:= floor(log[2](log[2](Z)));
```

```
    a:= s+15;
    for i from b by -1 to 3 do
        l:= floor((N-1)/2^ i)-1;
        r := floor((N-1)/2^( i-1));
        if mod(l, 2) = 0 then l := l+1 end if;
        if mod(r, 2) = 0 then r := r+1 end if;
        for j from s+2 to a do
            for u from l by 2 to r do
                c:= c+1; #counting
                d:= 2^j*u+1;
                g:= gcd(N, d);
                if 1 < g then do
                    lprint ( i, j, u, d, g, c);
                    return g
                    end do
                end if
            end do
        end do
    end do
end proc
```

#Test Results

$FactFermat(2^{2^5}+1, 2)$

30, 7, 5, 641, 641, 3

641

$FactFermat(2^{2^6}+1, 3)$

54, 8, 1071, 274177, 274177, 7334

274177

$FactFermat(2^{2^9}+1, 3)$

507, 16, 37, 2424833, 2424833, 319

2424833

$FactFermat(2^{2^{10}}+1, 3)$

1011, 12, 11131, 45592577, 45592577, 59040

45592577

$FactFermat(2^{2^{11}}+1, 3)$

2043, 13, 39, 319489, 319489, 230

319489

$FactFermat(2^{2^{12}}+1, 3)$

4093, 14, 7, 114689, 114689, 2

114689

$FactFermat(2^{2^{15}}+1, 3)$

32759, 21, 579, 1214251009, 1214251009, 4764

1214251009

$FactFermat(2^{2^{16}}+1, 3)$

65526, 19, 1575, 825753601, 825753601, 8100

825753601

$FactFermat(2^{2^{17}}+1, 25)$

131047, 19, 59251857, 31065037602817, 31065037602817, 12848715

31065037602817

$FactFermat(2^{2^{18}}+1, 3)$

262141, 20, 13, 13631489, 13631489, 5

13631489

$FactFermat(2^{2^{19}}+1, 3)$

524273, 21, 33629, 70525124609, 70525124609, 230089

70525124609

$FactFermat(2^{2^{21}}+1, 19)$

2097133, 23, 534689, 4485296422913, 4485296422913, 5203

4485296422913

$FactFermat(2^{2^{23}}+1, 2)$

8388606, 25, 5, 167772161, 167772161, 3

167772161

References

[1] Wilfrid, K., (2020). Prime factors k2n + 1 of Fermat numbers Fm and complete factoring status. Retrieved from http://www. prothsearch.com /fermat.html

[2] Sonal, S., & Dinesh, G. U. G., (2014). An overview to Integer factorization and RSA in cryptography. International Journal for Advance Research in Engineering and Technology, 2(9), 21-26

[3] Kharate, S. P. (2016). An overview of cryptography, innovation in IT, 3(1), 8-11

[4] Wang, X. (2020). FAST approach to factorize odd integers with special divisors. Journal of Mathematics and Statistics, 16(1), 1-9。

[5] Wang, X (2019). Brief summary of frequently — Used properties of the floor function: Updated 2018. IOSR J. Math, 15(1), 30-33.

[6] Wang, X. (2018). T3 tree and Its traits in understanding integers. Advances in Pure Mathematics, 8(5), 494-507

[7] GMP library, The GNU multiple precision arithmetic library. Retrieved from https://gmplib.org/

[8] Victor, S. NTL: A library for doing number theory. Retrieved from https://www.shoup.net/ntl/

[9] Miracl Cryptographic SDK. Retrieved from https://github.com/miracl/MIRACL

Xingbo Wang was born in Hubei, China. He got his master and the doctor’s degrees at National University of Defense Technology of China and had been a staff in charge of researching and developing CAD/CAM/NC technologies in the university. Since 2010, he has been a professor in Foshan University with research interests in computer application and information security. He is now the chief of Guangdong engineering center of information security for intelligent manufacturing system. Prof. Wang was in charge of more than 40 projects including projects from the National Science Foundation Committee, published 8 books and over 100 papers related with mathematics, computer science and mechatronic engineering, and invented 30 more patents in the related fields.