Central Asian Business Journal

Volume 23, No. 2, pp. 1-12
https://doi.org/XXXXXXXXXXXX

KIMEP UNIVERSITY

A simple enumerative polynomial-time algorithm for the prime factorization

Vassilly Voinov[1]

***Abstract**: A simple enumerative polynomial-time algorithm for the prime factorization of composite numbers $n \in \mathbb{N}^+$ is introduced. The worst case complexity of the algorithm is $O(\sqrt{n})$. This implies that the RSA algorithm used in cryptography is not secure. Results of the research can also be considered as a strong argument in favor of equality P=NP.*

***Keywords**: Prime factorization, RSA encryption, algorithm's complexity, P=NP*

***AMS Subject Classifications**: 11A51, 11Y05, 11Y16, 68Q15, 68Q17, 94A60*

1. Introduction

The prime factorization problem is formulated as follows: given a composite number $n \in \mathbb{N}^+$ find two different primes a and b such that $n = a \cdot b$. Fibonacci, Fermat, Euler, and many other famous mathematicians tried to find the effective method for solving this problem. In 1974 Pollard [5] suggested the well-known ***p*-1** special-purpose searching algorithm with the running time $O(B\log B\log n\log n)$, where B is a smoothness bound B. Unfortunately, as Jacqueline Speiser [9] noted, this algorithm is "effective at factoring numbers with a small prime-factor and only potentially retrieves a non-trivial factor". In 1971 Dixon [2] introduced a general-purpose factorization searching algorithm with the runtime of $\exp(\sqrt{2\log n\log\log n})$. In 2008 Pomerance [6] using the quadratic sieve improved the runtime of Dixon's algorithm to $\exp(\sqrt{\log n\log\log n})$.

Stephen Cook [1], p.6 noted that this problem "is a good example of a problem in NP that seems unlikely to be either in P or NP-complete". Stephen Cook also wrote that the computational complexity, i.e. inability to solve the problem in polynomial time on Turing machines "plays an important role in modern cryptography. The security of the Internet, including most financial

[1] Independent scholar, Kazakhstan. Email: voinovv@mail.ru

transactions, depends on complexity-theoretic assumptions such as the difficulty of integer factoring or breaking the Data Encryption Standard. If P=NP, these assumptions are all false. Specifically an algorithm solving 3-SAT in n^2 steps could be used to factor 200-digit numbers in a few minutes".

The best known so far algorithm of Müeller [4] solves 3-SAT problem in n^{15} steps, and, hence, does not help to solve the prime factorization problem. It is worth to mention also the following remark of Stephen Cook [1], p.4 "Shor [8] has shown that some quantum computer algorithm is able to factor integers in polynomial time, but no polynomial-time integer-factoring algorithm is known for Turing machines".

The paper is organized as follows. Section 2 considers and analyses the simple enumerative polynomial-time algorithm for the prime factorization. A brief discussion and conclusions are given in Section 3. The Appendix provides the R-script that can be used by an interested reader for reproducing the numerical results of this research or solving their own instances.

2. Main result and a computer experiment

Consider a product of two unknown primes $n = a \cdot b$. If, without loss of generality, $a < b$, then $a < \sqrt{ab}$. From this it follows that *a* belongs to the finite sorted set of primes $\{2,3,5, \dots, c\}$, where *c* is the largest prime less than $\sqrt{n}$. Since *a* divides *n* by condition, the following simple algorithm for determining *a* and respectively $b = n/a$ can be used:

Step 1. Open an existing file of primes or generate all $\pi(\sqrt{n})$ primes less than $\sqrt{n}$. π(x) denotes the number of primes less than or equal to $x \in \mathbb{R}^+$.
Step 2. Verify that *c* divides *n* evenly.
Step 3. If "yes" then $a = c$ and $b = n/c$. Otherwise, replace *c* by the first smaller than *c* prime and go to *Step 2*. Repeat while *c* is more than 2.

The main operation in the *Step 2* is the division of *n* by *c*. In the worst case the number of divisions equals to the number $\pi(\sqrt{n})$ of primes that can be approximated by $\sqrt{n}/(ln\sqrt{n} - 1)$ [3]. Since $\frac{\sqrt{n}}{ln\sqrt{n}-1} < \sqrt{n}$, the worst case complexity of the algorithm is approximately $O(\sqrt{n})$.

To check the correctness of the above algorithm consider the following computer experiment based on the R-script provided in Appendix:

Exp. 1. Let the upper bound $\mathcal{N}$ for the maximal prime generated belong to the interval [100000000, 200000000,…,1500000000]. 15 artificial instances with $a = 2$ and the largest prime *b* less than $\mathcal{N}$ were created. The script that did not use the parallelization was run for all 15 values of $n = 2b$. Results of calculations on a standard PC (Intel® Core ™ i7-2600 CPU@3.40 GHz, RAM 24.00 GB) are presented in Table 1 and Figure 1 below.

Table 1. Mean computing time Mt in seconds as a function of $p = \pi(\sqrt{n})$, where $n = 2b$ and b is the largest prime less than the upper bound $\mathcal{N}$ for the maximal prime generated.

$\mathcal{N}$	b	$n=2b$	$p = \pi(\sqrt{n})$	Mt
100000000	99999989	199999978	1663	0.0004600
200000000	199999991	399999982	2262	0.0006200
300000000	299999977	599999954	2717	0.0007400
400000000	399999959	799999918	3080	0.0008353
500000000	499999993	999999986	3401	0.0009376
600000000	599999971	1199999942	3699	0.0010190
700000000	699999953	1399999906	3961	0.0010812
800000000	799999999	1599999998	4203	0.0011490
900000000	899999963	1799999926	4435	0.0012150
1000000000	999999937	1999999874	4648	0.0012640
1100000000	1099999997	2199999994	4846	0.0013227
1200000000	1199999993	2399999986	5033	0.0013680
1300000000	1299999983	2599999966	5221	0.0014232
1400000000	1399999987	2799999974	5399	0.0014670
1500000000	1499999957	2999999914	5571	0.0015304

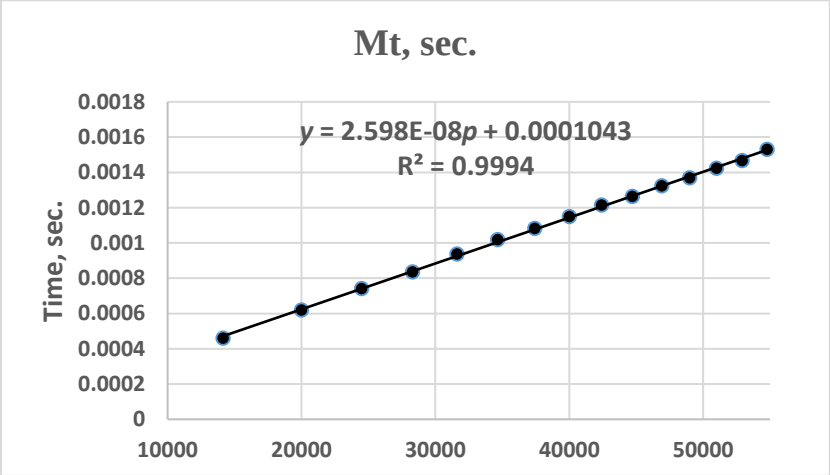

Fig. 1. The experimental dependence of Mt on $p = \pi(\sqrt{n})$ fitted by the Microsoft Excel 2018.

From this figure one sees that the linear function $y = 2.598 \cdot 10^{-8}p + 0.0001043$ (solid line with the multiple $R^2 = 0.9994$ and the adjusted one of 0.9993) provides an excellent fit for the CPU time dependence on $p = \pi(\sqrt{n})$. Two other criteria (the Residual Standard Error (RSE=$8.424 \cdot 10^{-6}$ on 13 degrees of freedom) and the Akaike's Information Criterion (AIC = -304.11)) confirm this conclusion. The script, using only values of *n*, produced the correct values of *b* implemented for creation the instances (see the second column of Table 1).

An interested reader may check that for n of 18 decimal digits 120156838817804240 the script gives the correct values of a = 240313681 and b = 499999993 for ~1.3 sec. It also confirms the numerical result of Jacqueline Speiser [9] for n = 773978585664881 obtained by Dixon's algorithm for ~20 sec. on a PC with 8 cores. Our algorithm takes 0.175 sec.

Two important conclusions follow from the above: a) the considered algorithm for the prime factorization is correct, and b) the algorithm is polynomial-time with the worst case complexity of $O(\sqrt{n})$.

3. A discussion and conclusions

The proposed polynomial-time algorithm for the prime factorization permits using even not very powerful PC to factor numbers of up to 18 decimal digits. There are much room to improve it. It is worth, e.g., to consider the following note of Speiser [9]: "to scale to larger integers would require either a cluster of many machines working in parallel or an architecture with a larger word size (128- or 256- bits)". Naturally that such an improvement will be done soon, and the numbers of 200 digits recommended in [7], p. 12 can be factored. Summarizing one may conclude that the main result of this research is that the RSA algorithm is not secure and can be hacked.

The second important conclusion is that the suggested algorithm is a polynomial-time one and that the prime factorization problem on the contrary of the opinion of Stephen Cook [1] is in P. This is a serious argument in favor of equality P=NP that has been experimentally proven in [10].

References

1. Cook, S. (2000) The P versus NP problem. *https://www.claymath.org/pvsnpPDF*.

2.Dixon, J.D. (1981) Asymptotically fast factorization of integers. *Math. of computation* **36(153)**, pp. 255-260.

3. How many primes are there? *https:// primes.utm.edu/howmany.html, accessed on March 16, 2023.*

4. Müeller, M. (2020) Polynomial Exact-3-SAT-Solving Algorithm. *Int. J. of Engineering Technologies* **9(3),** pp. 1-55, DOI: 10.14419/ijet.v9i3.30749.

5. Pollard, J.M. (1974) Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge*
Philosophical Society **76(3)**, Cambridge University Press, pp. 521-528.

6. Pomerance, C. (2008) Smooth numbers and the quadratic sieve. *Algorithmic Number Theory MSRI Publications* **44,**
pp. 69-81.

7. Rivest, R.L, Shamir, A, and Adleman, L. (1978) A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **26**, pp. 120-126.

8. Shor, P. (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. on Computing* **26**, pp. 1484-1509.

9. Speiser, J. Implementing and comparing integer factorization algorithms. *https://crypto.stanford.edu/cs359c/17sp/projects/Jacqueline Speiser.pdf, accessed on March 3, 2023.*

10. Voinov, V. (2023) An experimental supported by some strong theoretical arguments proof of the fundamental equality P=NP. *To appear in the Central Asia Business Journal*.

Appendix.

```
## To run the script you have to:
## 1. Download and install R. (E.g., version 4.2.2).
## 2. Install or invoke the R-packages: "primes" and "microbenchmark).
## 3. Copy the script and put it into the R-editor.
## 4. Set the desired value of n.
## 5. Save and run the script.
library(primes)
library(microbenchmark)
options(digits=22)
g<-generate_primes(max=500000000)
g1<-as.data.frame(g); g1<-g1[[1]]
Pr_decomp_1<-function(g33){
for(i in 1:(g33-1)){
g34<-g321[g33-i]
g35<-g3/g34
g36<-g35-floor(g35)
if(g36==0){
y10<-g34
z10<-floor(g35)
message("a=",y10)
message("b=",z10)
break
};};}
dp<-773978585664881 ## insert your n here! Values of n less than
## 18 decimal digits are recommended!
#dp<-120156838817804240
g3<-dp
message("n=",g3)
g31<-floor(g3^(0.5))
g321<-subset(g1,g<g31)
```

```
g33<-length(g321)
Ut<-microbenchmark(Pr_decomp_1(g33),times=1L)
Ut1<-Ut$time
Ut2<-Ut1/1000000000
message("Computing time, sec.=",Ut2)
## In the output you will see n, a, b, and computing time.
```