Int. J. Communications, Network and System Sciences, **2010, 3, 901-906**
doi:10.4236/ijcns.2010.312123 Published Online December 2010 (http://www.SciRP.org/journal/ijcns)

Enhanced Euclid Algorithm for Modular Multiplicative Inverse and Its Application in Cryptographic Protocols

Boris S. Verkhovsky
Computer Science Department, New Jersey Institute of Technology,
University Heights, Newark, USA
E-mail: *verb@njit.edu*
Received August 21, 2010; *revised October* 4, 2010; *accepted November* 16, 2010

Abstract

Numerous cryptographic algorithms (ElGamal, Rabin, RSA, NTRU etc) require multiple computations of modulo multiplicative inverses. This paper describes and validates a new algorithm, called the Enhanced Euclid Algorithm, for modular multiplicative inverse (MMI). Analysis of the proposed algorithm shows that it is more efficient than the Extended Euclid algorithm (XEA). In addition, if a MMI does not exist, then it is not necessary to use the *Backtracking* procedure in the proposed algorithm; this case requires fewer operations on every step (divisions, multiplications, additions, assignments and push operations on stack), than the XEA. Overall, XEA uses more multiplications, additions, assignments and twice as many variables than the proposed algorithm.

Keywords: Extended-Euclid Algorithm, Modular Multiplicative Inverse, Public-Key Cryptography, RSA Cryptocol, Rabin Information Hiding Algorithm, ElGamal Encryption/Decryption, NTRU Cryptosystem, Computer Simulation, Low Memory Devices

1. Introduction

Elements of modular arithmetic are essential for public-key encryption algorithms such as the RSA cryptographic algorithm [1-3], and RSA with digital signature [4]; ElGamal cryptocol [5]; Rabin encryption/decryption scheme based on extraction of square roots [6]; NTRU cryptosystem [7]; and extensions of some of these algorithms in Gaussian arithmetic require computation of a modular multiplicative inverse (MMI) [8-10]. The MMI is also computed for cryptanalysis of public-key cryptographic protocols [11-13].

A new algorithm, called the E*n*hanced-*E*uclid *A*lgorithm (NEA), for modular multiplicative inverse (MMI) is described and validated in this paper.

Definition 1: Given relatively prime integers p_0 and p_1, there exists an unique integer *x* such that

$$p_1 x = 1 \pmod{p_0}. \tag{1}$$

Then *x* is defined as the *modular multiplicative inverse* of p_1 modulo p_0 or, for short, MMI.

The NEA finds for two relatively prime integers p_0 and p_1 an integer number *x* that satisfies the Equation (1). And if p_0 and p_1 are not relatively prime, then NEA finds a gcd(p_0, p_1). The E*x*tended-*E*uclid *a*lgorithm (XEA) also finds a MMI of p_1 modulo p_0 if and only if gcd(p_0, p_1) = 1 [1,2,7].

This paper proves the validity of NEA and provides its analysis. The analysis demonstrates that NEA is faster than the Extended Euclid algorithm. Preliminary results of this paper are published in [8].

2. Basic Arrays and Their Properties

Let's consider *five* finite integer arrays:

$$\{p_i\};\{c_i\};\{t_k\};\{w_k\};\{z_k\} \tag{2}$$

Definition 2: Let $\{p_i\}$ and $\{c_i\}$ be integer arrays defined according to the following generating rules:

Given two relatively prime integers p_0 and p_1 such that $p_0 > p_1$, **for** $i \ge 1$ **do while** $p_i \ge 2$,

$$p_{i+1} := p_{i-1} \bmod p_i;\ c_i := \lfloor p_{i-1}/p_i \rfloor. \tag{3}$$

Definition 3*:* Let $\{t_k\}$ be an arbitrary integer array for all $k \ge 1$; let for initially specified w_0, w_1, z_0 and z_1 the following generating rules be defined for all $k \ge 2$:

$$w_k := w_{k-1}t_{k-1} + w_{k-2};$$

and

$$z_k := z_{k-1}t_{k-1} + z_{k-2}. \quad (4)$$

***Proposition* 1**: Let's define for $k \geq 1$

$$D_k := \begin{vmatrix} w_k & w_{k-1} \\ z_k & z_{k-1} \end{vmatrix}. \quad (5)$$

Then

$$D_k = (-1)^{k-1} D_1. \quad (6)$$

Proof: Consider D_k and substitute in the left column the values of w_k and z_k defined in (4). After simplifications we derive that $D_k = -D_{k-1}$, which recursively implies (6).

***Proposition* 2**: Consider the integer arrays $\{t_k\}$, $\{w_k\}$, and $\{z_k\}$ (4)
where $w_0 := 1$; $z_0 := 0$; $|z_1| := 1$.

Then $(-1)^{k-1} z_1 z_k$ is a multiplicative inverse of w_{k-1} modulo w_k for every w_1.

Proof: Indeed, since $D_1 = -z_1$, then (5) implies that

$$w_k z_{k-1} - z_k w_{k-1} = (-1)^k z_1, \quad (7)$$

or that

$$w_{k-1}\left[(-1)^{k-1} z_1 z_k\right] - z_1^2 = w_k\left[(-1)^{k-1} z_1 z_{k-1}\right]$$

Then from (7) it follows that

$$\left[w_{k-1}(-1)^{k-1} z_1 z_k - 1\right] / w_k = (-1)^{k-1} z_1 z_{k-1},$$

i.e., $x = (-1)^{k-1} z_1 z_k$.

***Proposition* 3**: If for all $0 \leq k \leq r$

$$t_k := c_{r-k}, \text{ and } w_k := p_{r-k}, \quad (8)$$

then p_0 and p_1 are the initial values that generate the arrays $\{p_i\}, \{c_i\}$,

$$\{t_k\} := \{c_{r-k}\} \text{ and } \{w_k\} := \{p_{r-k}\}.$$

Remark 1: Notice that $\{t_k\} = \{c_i\}^R$ and $\{w_k\} = \{p_i\}^R$, where the superscript *R* means that the arrays $\{c_i\}$ and $\{p_i\}$ are written in a reverse order.

Theorem 1: *For all* $k = 1, \cdots, r$, $(-1)^{k-1} z_1 z_k$ *is the inverse of* p_{r-k+1} *modulo p.*

Proof follows from *Propositions* 1-3.

Theorem 1 implies the following assertions:

1) **if** *k* is *odd* **and** $z_1 = 1$,

then $x := z_k > 0$;

2) **if** *k* is *even* **and** $z_1 = -1$,

then $x := z_k < 0$.

In the latter case select

$$x := p_0 + z_k. \quad (9)$$

3. Enhanced Euclid Algorithm for MMI

The proposed algorithm uses stack as a data structure, [2].

vars: *r*; *L*; *M*; *S*; *t*: all integer numbers; *b*: Boolean;

procedure *Forward*:

$$L := p_0; \quad M := p_1; b: = 0; \{r: = 0\};$$

repeat $t := \lfloor L/M \rfloor$; $S := L - Mt$;

$$b := 1 - b; \; \{r := r + 1\}; \quad (10)$$

push *t* {onto the top of the stack};

$$L: = M; M: = S; \quad (11)$$

until *S* = 1;

Remark 2: **if** *S* = 0, **then** $\gcd(p_0, p_1) = t$; therefore the *MMI* does not exist;

procedure *Backtracking*:

S: = 0; $M := (-1)^b$; {by (9) in the Theorem 1};

repeat pop *t* {from the top of the stack};

$$L: = Mt + S; \; S := M; \; M := L; \quad (12)$$

until the stack is *empty*;

output *x*: = *L*; {if *x* < 0, then *x*: = *x* + p_0 }.

Remark 3: *r* is the height of the stack and is used below for analysis of the proposed algorithm.

4. Two Illustrative Examples

Let's demonstrate how NEA finds a multiplicative inverse *x* of 27,182,845 modulo 31,415,926. **Table 1** below shows the computation of remainders in the upper row and stores the quotients in the middle row (the stack). Then the *Backtracking* procedure is used to compute from right to left until the stack is empty. The inputs and MMI are shown in bold, and the stack values are in italics. Since the total number of steps (the height of the stack) is equal to *fourteen* (*i.e.*, *even*), then *x* = **13,939,773**.

Direct verification confirms that indeed

27182845 * 13939773 mod 31415926 = 1.

In the second numerical example we need to find the MMI *z* of 27,319,913 modulo 177,276,627 {see **Table 2**}. Since the height of the stack is *odd*, then

z = 177276627-34480855 = **142795772**.

Direct computation verifies that *z* is indeed the MMI of 27319913, since

27319913 * 142795772 mod 177276627 = 1.

5. Complexity Analysis of MMI Algorithm

Consider four non-negative integer arrays $\{q_k\}$, $\{d_k\}$,

Table 1. Modular multiplicative inverse algorithm in progress.

31415926	**27182845**	4233081	1784359	664363	455633	208730
Stack	*1*	*6*	*2*	*2*	*1*	*2*
13939773	12061484	1878289	791750	294789	202172	92617

(continuation)

208730	38173	17865	2443	764	151	9	7	2	1
2	*5*	*2*	*7*	*3*	*5*	*16*	*1*	*3*	***
92617	16938	7927	1084	339	67	4	3	**1**	**0**

Table 2. Second numerical illustration.

177276627	**27319913**	13357149	605615	33619	473	36	5	1
Stack	*6*	*2*	*22*	*18*	*71*	*13*	*7*	***
34480855	5313808	2598007	117794	6539	92	7	**1**	**0**

$\{p_i\}$ and $\{c_i\}$ defined as follows:

$$d_k := \lfloor q_{k-1}/q_k \rfloor; \tag{13}$$

$\{p_i\}$ and $\{c_i\}$ satisfy (3) and are defined as

$$p_{i+1} := p_{i-1} - p_i c_i;\ q_{k+1} := q_{k-1} - q_k d_k \tag{14}$$

Then (3) and (14) imply that

$$p_{i+1} := p_{i-1} - p_i c_i = p_{i-1} \bmod p_i$$

Hence both arrays $\{p_i\}$ and $\{q_k\}$ are strictly decreasing, and all terms of the corresponding arrays $\{c_i\}$ and $\{d_k\}$ are positive integer numbers.

Definition 4: $\{x_j\}_s$ is a (*s* + 1)-dimensional vector, consisting of the first *s* + 1 terms of an array $x_0, x_1, ..., x_{j-1}, x_j$, …, *i.e.*,

$$\{x_j\}_s := (x_0, x_1, ..., x_{s-1}, x_s).$$

Theorem 2: Consider

$$\{c_i\}_r \geq 1; \{p_i\}_s; \{d_k\}_s; \{q_k\}_s \geq 1$$

and $\{p_i\}_r \geq 1$.

Let $p_0 = q_0$; $\{c_i\}_s \leq \{d_k\}_s$, *i.e.*, for all $j = 1, \cdots, s$ there exists at least one $j = l$ such that $c_l < d_l$. Then for all $1 \leq j \leq s$ the following inequalities hold:

if $1 \leq j \leq l-1$, then $p_j \geq q_j$

otherwise

$$p_j > q_j. \tag{15}$$

Proof: Assuming that the statement (15) holds for all $i \leq j-1$, it can also be demonstrated by induction that (15) also holds for $i = j$.

Consider

$$\begin{aligned} t_j = d_j - c_j &= \lfloor q_{j-1}/q_j \rfloor - \lfloor p_{j-1}/p_j \rfloor \\ &\leq \lfloor p_{j-1}/q_j \rfloor - \lfloor p_{j-1}/p_j \rfloor. \end{aligned} \tag{16}$$

If $j \leq l-1$, then $t_j \geq 0$ else $t_j > 0$. Hence from (16) it follows: if $j \leq l-1$, then $p_j \geq q_j$ otherwise $p_j > q_j$.

Since $p_0 = q_0$, then (14) holds for all $j \leq s$.

Q.E.D.

Consider a pair of relatively prime seeds p_0 and p_1 that generates an array $\{c_i\}_r = 1$. Consider also another pair of relatively prime seeds p_0 and q_1 that generates an array $\{d_k\}_s \geq 1$, *i.e*. such that *not* all its terms are equal to *one*. Let *r* and *s* be the number of steps required respectively to find MMIs for the first and the second pair using either XEA or NEA. This assumption implies that $q_s = 1$. Then by Theorem 2 $\{p_i\}_s \geq \{q_k\}_s$ and $p_s > q_s = 1$. Hence $r > s$.

Corollary: A pair of seeds, that for a given p_0 requires the maximal number of steps for computation of a MMI, generates an unary array of quotients, *i.e*., all components in $\{c_i\}_r = 1$. Thus, as it follows from (3) and (14), this pair of seeds must generate the following array of integer numbers: $p_2 := p_0 - p_1$; $p_3 := p_1 - p_2$; $\cdots$; $p_r := p_{r-2} - p_{r-1} = 1$. It is easy to verify that this array is equivalent to the sequence of Fibonacci numbers

$$\{F_{r+2}, F_{r+1}, ..., F_4, F_3, F_2\}.$$

In other words, for every $i = 0, ..., r$ $p_i := F_{r+2-i}$, [14].

Remark 4: The pair $p_0 = F_{r+2}$; $p_1 = F_{r+1}$ is *not* the only one that generates a) a unary array of quotients and b) a decreasing integer array where the r^{th} remainder equals *one*.

Indeed, the following pairs of seeds have the same property for all non-negative integer numbers t and u:

1) $p_0 = F_{r+2} + tF_r$; $p_1 = F_{r+1} + tF_{r-1}$; for $t = 1$ $\{p_i\}^R = \{L_1; L_2; \cdots; L_{r+1}\}$ is a sequence of the Lucas numbers 1, 3, 4, 7, 11, 18, $\cdots$ [15];

2) $p_0 = tF_{r+2} + (1-t)F_{r-1}$; $t \ge 1$;
$p_1 = tF_{r+1} + (1-t)F_{r-2}$;

3) $p_0 = F_{r+1} + tF_r$; $t \ge 1$; $p_1 = F_r + tF_{r-1}$;

4) $p_0 = (1+t)F_{r+2} + tF_{r-2} + uF_r$;
$p_1 = (1+t)F_{r+1} + tF_{r-3} + uF_{r-1}$.

Here the Fibonacci numbers with *zero* and *negative* indices are computed in accordance with the formula:

$$F_{-m} = (-1)^{m-1} F_m .$$

For all pairs, listed above, exactly r steps are required to find the MMI. However, all these pairs are special cases of a pair of seeds where

$$p_0 = bF_r + F_{r-1} \text{ and } p_1 = bF_{r-1} + F_{r-2} .$$

Therefore for all $0 \le i \le r$

$$p_i = bF_{r-i} + F_{r-i-1}, \quad p_{r-1} = b; \text{ and } \quad p_r = 1 .$$

Consider

$$v = \left(1-\sqrt{5}\right)/2 \text{ and } w = \left(1+\sqrt{5}\right)/2 .$$

Using a *z*-transform approach, we deduce that for all $0 \le k \le r$

$$p_{r-k} = \left[(b-v)w^k + (w-b)v^k\right]/\sqrt{5} . \quad (17)$$

Then for a large r

$$p_{r-k}\sqrt{5} - (b-v)w^k = (w-b)(-1)^k |v|^k \to 0$$

since

$$|v| < 1 . \quad (18)$$

The relation (18) implies that for a large r

$$p_0 = \left[w^r (b-v)/\sqrt{5}\right]\left[1+o(w)\right] . \quad (19)$$

Let $z := \max_{b \ge 2} r(p_0, b) = r(p_0, 2)$.
Therefore

$$z = \log_w\left[\sqrt{5}p_0/(2-v)\right] = \lfloor \log_w p_0 \rfloor\left[1+o(p_0)\right] \quad (20)$$

Thus (20) implies that the height of a stack satisfies the following inequality:

$$r \le \lfloor \log_w p_0 \rfloor \times \left[1+o(p_0)\right], \text{ [8]}. \quad (21)$$

***Example* 3**: In this example (see the table below) $1919^{-1} \bmod 3105 = 1364$.

Indeed, 1919 * 1364 mod3105 = 1.

Table 3 illustrates the case where the height of the stack is approaching the upper bound in the Inequality (21).

Remark 5: Although this upper bound is achievable if $p_0 = bF_r + F_{r-1}$ and $p_1 = bF_{r-1} + F_{r-2}$, for this pair of inputs the *MMI* can be computed explicitly: indeed, it equals $(-1)^{r-1} F_r$.

Remark 6: If in RSA public-key encryption [3], $p_0 = c \times 10^{100}$, then $r \le 100/\log_{10} w + \log_{10} c$; therefore $r \le 479 + \log c$.

Over one thousand computer simulations demonstrate that the average height of the stack is actually almost 40% smaller than the upper bound in (21).

6. Extended-Euclid Algorithm (XEA)

XEA also finds a multiplicative inverse of p_1 modulo p_0 provided that gcd(p_0, p_1) = 1, [2,7].

1) Assign ($X1$, $X2$,$X3$):= (1, 0, p_0); ($Y1$,$Y2$,$Y3$):= (0,1, p_1);

2) **if** $Y3$ = 0 **return** $X3$ = *gcd*(p_0, p_1); {there is no inverse};

3) **if** $Y3$ = 1 **return** Y3 = *gcd*(p_0, p_1);
else $Y2$ is the multiplicative inverse;

4) $$Q := \lfloor X3/Y3 \rfloor; \quad (22)$$

5) $$(T1, T2, T3) := (X1 - QY1, X2 - QY2, X3 - QY3); \quad (23)$$

6) $$(X1, X2, X3) := (Y1, Y2, Y3); \quad (24)$$

7) $$(Y1, Y2, Y3) := (T1, T2, T3); \quad (25)$$

8) **goto** 2.

7. Comparative analysis of NEA vs. XEA

Both algorithms require equal number of steps r for computation of all quotients: values of t on the *Forward* procedure in (10), and Q in (22), respectively.

In addition, NEA requires r more steps on the *Back-tracking* procedure to compute the values of L in (12).

Therefore each step of XEA requires *one* division, *three* multiplications, *three* long algebraic additions and *ten* assignments, see (22)-(25).

Table 3. {Worst-case space complexity}: Size of stack.

3105	**1919**	1186	733	453	280	173	107	66	41	25	16	9	7	2	1
Stack	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*3*	***
1364	843	521	322	199	123	76	47	29	18	11	7	4	3	**1**	**0**

XEA uses *ten* variables. Yet in both procedures NEA uses *one* division, *two* multiplications, *two* long additions, *two* stack operations, (push and pop), and *eight* assignments, see (10)-(12). NEA uses *four* integer variables, *one* binary variable and, in addition, $\Theta(\log_w p_0)$ of memory to store the stack.

Notice that if a MMI does not exist, then there is no need to use the *Backtracking* procedure in NEA. In this case NEA requires even fewer operations than XEA: *one* division, *one* multiplication, *one* addition, *one* push operation and *five* assignments per every step. Yet XEA still requires the same number of operations per step as in the case if a MMI does exist. Hence, overall XEA uses more multiplications, more additions, more assignments and twice as many variables than the proposed algorithm.

8. Average Complexity of XEA and NEA

If both inputs p_0 and p_1 are chosen randomly, then the probability that $\gcd(p_0, p_1) = 1$ equals $6/\pi^2$ [2].

Let us consider the following notations:

w_{xea} -worst-case specific complexity (per step) of XEA;

w_{nea} -worst-case specific complexity of NEA;

a_{xea} -average-case specific complexity of XEA;

a_{nea} -average-case specific complexity of NEA;

let t_d; t_m; t_a; t_s; t_{st} be time complexities of division, multiplication, addition, assignment and stack operations **push** and **pop** respectively.

Then

$$w_{xea} = t_d + 3\, t_m + 3t_a + 10t_s; \tag{26}$$

$$w_{nea} = t_d + 2\, t_m + 2t_a + 8t_s + 2t_{st}; \tag{27}$$

$$a_{nea} = (t_d + 2\, t_m + 2t_a + 8t_s + 2t_{st}) \times 6/\pi^2 + (t_d + t_m + t_a + 5t_s + t_{st}) \times (1 - 6/\pi^2) \tag{28}$$

Notice that $a_{xea} = w_{xea}$; and

$$t_d \approx t_m \gg t_a \approx t_s \approx t_{st} \tag{29}$$

Then (27)-(29) implies that

$$R := a_{xea}/a_{nea} = 2\pi^2/(3 + \pi^2) = 1.538\cdots \tag{30}$$

9. Conclusions

Analysis of the proposed algorithm (NEA) for modular multiplicative inverse demonstrates that its execution requires on average 53.8% less time, than the execution of the Extended Euclid algorithm.

Theoretical analysis of space complexity of the Enhanced-Euclid algorithm shows that it requires relatively small bit-storage for its execution. This storage does not exceed a 2*K*-bit level for a public-key encryption algorithm, where the inputs p_0 and p_1 are integers on the interval $(10^{100}, 10^{400})$.

On the other hand, computer simulations demonstrate that the average bit-storage is actually 40% *smaller than* 2*K*. Hence NEA can be executed if necessary by a custom-built chip with relatively modest memory, [7]. This property of the Enhanced-Euclid algorithm is especially useful for implementation of encryption in low memory environments such as smart or PC cards, cell phones, wearable computers and other integrated devices.

10. Acknowledgements

I express my appreciation to R. Rubino, J. Runnells, M. Sikorski, C. Washington and to anonymous reviewer for comments and suggestions that improved the style of this paper; and to A. Koripella for her assistance in running computer experiments.

11. References

[1] R. Crandall and C. Pomerance, "Prime Numbers: A Computational Perspective," Springer, Berlin, 2001.

[2] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, "Handbook of Applied Cryptography," CRC Press, Boca Raton, 1997.

[3] R. L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signature and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120-126.

[4] B. Verkhovsky, "Overpass-Crossing Scheme for Digital Signature," Keynote Address, *Proceedings of International Conference on System Research*, *Informatics and Cybernetics*, Baden-Baden, 30 July-4 August 2001.

[5] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, Vol. 31, No. 4, 1985, pp. 469-472.

[6] M. Rabin, "Digitized Signatures and Public Key Functions as Intractable as Factorization," *Technical Report*: *MIT/LCS/TR*-212, MIT Laboratory for Computer Science, Cambridge, 1979.

[7] J. Hoffstein, J. Pipher and J. H. Silverman, "An Introduction to Mathematical Cryptography," Springer, Berlin, 2008.

[8] B. Verkhovsky, "Multiplicative Inverse Algorithm and Its Complexity," *Proceedings of International Conference-InterSYMP*-99, Baden-Baden, July 1999, pp. 62-67.

[9] B. Verkhovsky, "Accelerated Cyber-Secure Communication Based on Reduced Encryption/Decryption and Information Assurance Protocols," *Journal of Telecommunications Management*, Vol. 2, No. 3, 2009, pp. 284-293.

[10] B. Verkhovsky, “Hybrid Authentication Cybersystem Based on Discrete Logarithm, Entanglements and Factorization,” *International Journal of Communications*, *Network and System Sciences*, Vol. 3, No. 7, 2010, 579-584.

[11] B. Verkhovsky, “Generalized Baby-Step Giant-Step Algorithm for Discrete Logarithm Problem,” *Advances in Decision Technology and Intelligent Information Systems*, International Institute for Advanced Studies in Systems Research and Cybernetics, Baden-Baden, 2008, pp. 88-89.

[12] B. Verkhovsky, “Integer Factorization: Solution via Algorithm for Constrained Discrete Logarithm Problem,” *Journal of Computer Science*, 2009, Vol. 5, No. 9, 674-679.

[13] B. Verkhovsky, “Potential Vulnerability of Encrypted Messages: Decomposability of Discrete Logarithm Problems,” *International Journal of Communications*, *Network and System Sciences*, Vol. 3, No. 8, 2010, pp. 639-644.

[14] R. B. McClenon, “Leonardo of Pisa and His Liber Quadratorum,” *American Mathematical Monthly*, Vol. 26, No. 1, 1919, pp. 1-8.

[15] D. Harkin, “On the Mathematical Works of Francois-Edouard-Anatole Lucas,” *Enseignement Mathematique*, Vol. 3, No. 2, 1957, pp. 276-288.

Appendix

Computer experiments

1) Pairs of *N* decimal-digit long integers *A* and *B* were generated randomly, where $A > B$;

2) Then MMI *C* of *B* modulo *A* was computed; in other words we found an integer *C*, for which holds $BC \bmod A = 1$;

3) 125 experiments have been carried out for each value of $N = \{6, 8, 10, \cdots, 18, 20\}$;

4) The values of *S* (size of stack-storage) for each *N* were tabulated; (not shown in the **Table A1**).

5) The values of average *S* for every *N* and the range of *S* for each *N* are presented in the **Table A1**.

Table A1. **Results of computer experiments**.

N	**Average *S***	**Range of *S***	
		[*min*,	***max*]**
6	12.65	[7,	17]
8	16.20	[9,	21]
10	19.96	[14,	29]
12	24.91	[19,	33]
14	28.53	[18,	36]
16	32.30	[20,	41]
18	36.70	[23,	50]
20	40.29	[26,	54]

