A Study Of Euclidean K-Ary Gcd Algorithm

Nikolai Andreevich Antonov[1], Shamil Talgatovich Ishmukhametov[1],
Al Halidi Arkan M[1], Mariia Evgenevna Maiorova[2]

1. **Kazan Federal University, Shamil.Ishmukhametov@kpfu.ru**
2. **Knrtu-Kai, Fsbei**

ABSTRACT

The problem of computing the greatest common divisor GCD of natural numbers is one of the most common problems that is solved in modern computational mathematics and its applications. At the moment, many algorithms are known to solve this problem, but every day the requirements for the effectiveness of such algorithms become more stringent. As a result, there is a need to create new algorithms that are more efficient in time and number of the operations performed. Besides, they must allow the possibility of their transformation into modern programming languages while maintaining the efficiency of the algorithm.

This article presents an analysis of the realization features and the results of testing the speed of three GCD computation algorithms: the classical Euclidean algorithm, the Sorenson k-ary algorithm, and the approximating k-ary algorithm developed by the second of the authors in MicrosoftVisualStudio in C #. Qualitative and quantitative data on the effectiveness of these algorithms in terms of time and number of the steps within the main cycle are have been obtained.

The concluding part of the article contains the analysis of the results obtained, their representation in the diagrams, and gives the recommendations on the choice of the parameters of the methods.

Keywords:GCD of natural numbers, Euclidian GCD algorithm, k-ary GCD algorithm, approximating k-ary algorithm.

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

1. INTRODUCTION

As a rule, modern cryptographic systems work with long numbers and provide a procedure for calculating their greatest common divisor GCD (Ishmukhametov S.T. 2011), (Ishmukhametov S., Mubarakov B., Mochalov A. 2015). The most common of these procedures is the classical Euclidean algorithm. However, other algorithms are also known, the overall goal of which was to reduce the complexity of GCD computation, and to reduce the search time.

One of the most promising algorithms is the k-ary algorithm of the GCD search, published in the 1990s by Jonathan Sorenson (Sorenson J. 2004,Sorenson J. 1990,Sorenson J. 1994). In comparison with the classical Euclidean algorithm, this algorithm makes it possible to significantly reduce the complexity of calculations with a careful choice of the k parameter. Sorenson and other scientists subsequently proposed several variantsto improve and modify the k-ary algorithm ((Weber K. 1995,Jebelean T. 1993,Wang X., Pan V. 2003)).

One of them is the approximating k-ary algorithm proposed by S. T.Ishmukhametov (Ishmukhametov S.T. 2016).

The aim of this paper is to realize and study the three mentioned algorithms for computing GCD. The next three parts deal with the classical Euclidean algorithm, the original version of k-ary algorithm and its modification - an approximating k-ary algorithm. For each of the algorithms, a theoretical justification is given, an estimate of the complexity is indicated, and a realization of the algorithm using the programming language C # is performed. Next, the results of testing each of the algorithms by run-time usingthe numbers of different lengths are presented. The final part of the work compares the algorithms according to the efficiency of the time of work and the number of steps within the main cycle.

2. METHODS

The tables belowpresentthe results of testing of the classical Euclidean algorithm, the Sorenson k-ary algorithm and the approximating k-ary algorithm. All algorithms were realizedusing the C # programming language in MicrosoftVisualStudio 2010.

In the tables compiled from the results of testing the original and approximating k-ary algorithms the “arity” of the algorithm is reservedvertically, and horizontally - the

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

length of the numbers in decimal positions. Each cell of the table is divided into two parts. At the top of the cell, the average search time of the GCD algorithm of a given "arity" on one of a thousand pairs of randomly generated numbers of a given lengthis indicated. By analogy, at the bottom of the cell thereis shown the number of iterations of the main loopdone by the algorithm of a given"arity" also by one of a thousand pairs of randomly generated numbers of a given length. It should be noted that the classical Euclidean algorithm has no "arity", so the results of its testing are tabulated, in the upper line of which, horizontally, the length of the numbers in decimal places is postponed, and below, the corresponding results are given for the average working time of the algorithm realizationby thousand pairs randomly generated numbers and the average number of steps made within the main loop.

3. RESULTS

Table 1. ClassicalEuclidianAlgorithm

D ***(numbers' length in decimal digits)***	***10***	***25***	***50***	***100***	***250***	***500***	***1000***
Working time (μs)	4,5	16	35	97	313	760	2650
Average number of iterations (iter)	19	48	96	194	485	969	1940

Table 2.SorensonK-aryAlgorithm

K\D		***10***	***25***	***50***	***100***	***250***	***500***	***1000***
16	***μs***	*17*	*70*	*175*	*450*	*1001*	*5763*	*20744*
	iter	*17*	*70*	*94*	*200*	*483*	*959*	*1900*
64	***μs***	*12*	*61*	*133*	*367*	*918*	*4641*	*15437*
	iter	*15*	*40*	*80*	*160*	*403*	*806*	*1615*
256	***μs***	*8*	*48*	*120*	*315*	*1161*	*3701*	*13583*
	iter	13	32	66	133	331	664	1331
421	***мк***	*15*	*41*	*111*	*257*	*1018*	*3243*	*11367*
	iter	13	31	65	129	329	661	1328
853	***μs***	*10*	*43*	*93*	*244*	*969*	*3047*	*10493*

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

	iter	13	30	65	131	327	655	1313
1307	***µs***	*14*	*49*	*103*	*69*	*894*	*2947*	*10267*
	iter	12	28	58	116	291	583	1167
1801	***µs***	*10*	*48*	*101*	*104*	*1041*	*2832*	*1026*
	iter	9	23	47	99	240	481	959

Table 3. ApproximatingK-aryAlgorithm

K\D		***10***	***25***	***50***	***100***	***250***	***500***	***1000***
16	***µs***	60	189	479	1152	4257	12946	44579
	iter	8	20	41	82	207	414	829
64	***µs***	63	194	475	1108	3824	11306	37393
	iter	6	16	33	67	167	335	671
256	***µs***	62	201	517	1046	3566	10290	32783
	iter	5	13	27	55	139	279	557
422	***µs***	62	212	552	1201	3871	10237	33000
	iter	5	13	27	55	138	277	551
852	***µs***	56	281	591	1175	3767	10165	32460
	iter	5	13	26	53	133	266	534
1506	***µs***	61	207	501	1155	3655	9848	30785
	iter	4	12	24	49	123	247	494
3026	***µs***	78	243	524	1195	3897	10078	29698
	iter	4	11	22	45	114	230	461

4. DISCUSSION

CLASSICAL EUCLIDIANALGORITHM. The natural numbers*A* and *B*, A> Bare transmittedto the input of the algorithm. The classical Euclidean GCD search algorithm is based on the following recurrence equation:

$$GCD(A,B) = GCD(B, A \bmod B).$$

The algorithm is realizedsuccessively, called iterations, in the course of each of which the remainder is calculated $C = AmodB$, a transition to a new pair (B, C)takes placeandthe

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

calculation continues with the new pair. The calculation stops when the second argument of the pair is equal to 0, then the first argument is the sought GCD. The very algorithm can be written in the C # as follows:

```
publicBigInteger Method(BigInteger number1, BigInteger number2)
{   if (number1 <= 1 || number2 <= 1)
return 1;
BigInteger R;
if (number2 > number1)  {    R = number1;
number1 = number2;
number2 = R;   }
while (number2 != 0)
{  R = number1 % number2;
number1 = number2;
number2 = R;   }
returnnumber1;
}
```

Estimating the complexity of this algorithm, we note that for every two iterations of the cycle, the dividend decreases at least twice. This means that the number of iterations is $O(n)$, where$n$ is the length of the input numbers. The worst estimate is achieved by the neighbor numbers of the Fibonacci series, and the average number of iterations is estimated by the value (Dixon (DixonJ. 1970)):

$$Number\ of\ iteration \sim \frac{12}{\pi^2}\log_2 B$$

The operation of calculating the remainder of $AmodB$can be performed within the time $O(n\log_2 n)$, so that the overall performance of the algorithm is estimated by $O(n^2\log_2 n$.

K-ARY ALGORITHMOF SORENSON. First, we consider a binary algorithm for calculating GCD for a pair of natural numbers. The input data of this algorithm are odd natural numbers. We designate them by u, v. The binary algorithm can be represented as follows:

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

while(u!= 0) and (v!=0) do:

if (u is even) u:=u/2

else if (v is even) v:=v/2

else t:= abs(u - v)/2;

if (u > v) then u:=t else v:=t;

if (u = = 0) then t:=v else t:=u;

output(t);

Generalizing this algorithm, we will obtain the k-ary GCD algorithm. The main idea of the iteration of the k-ary algorithm is to search for a given pair of numbers$(A, B)$of the integers $x, y$for a small fixed k:

$0 < x < \sqrt{k}$ $-\sqrt{k} < y < \sqrt{k}$,

such that the identity

$$ux + vy \equiv 0 \; mod \; k.$$

isperformed

Then, at the step of the algorithm, we can pass from the pair(u, v) to the new pair (v, w), where $w = (ux + vy)/k$(it is also necessary to reduce *w*by the divisors*k*).

Sorenson in the article (3) proposed to perform k-ary algorithm in four stages: the stage of precomputations, the first “pilot” division, the main part, the second “pilot” division.

Themainpartofthealgorithmis performed in the following way :

while(v!= 0){

if (gcd(u,k) > 1) u=u/gcd(u,k);

else (if gcd(v,k) > 1) v=v/ gcd(v,k);

else

find integers x, y, ux + vy ≡0 (modk);

w= | ux+ vy|/k;

if (v > w){u=v ;v=w ;} else u= w;

}

return u ;

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

When trying to realize the k-ary algorithm, it becomes clear that the way to choose integers *x* and *y* in many ways affects its performance. According to the Sorenson theorem, the coefficients *x* and *y* can be chosen $0 < |x| + |y| \leq 2\sqrt{k}$.

For the maximum performance of the algorithm, it is also advantageous to choose the number*x*being small and positive, the number*y* being sufficiently large negative.

The stage ***of preliminary calculations:*** if the base of algorithm k is already chosen, then for all natural numbers less than k, the algorithm involves calculating their greatest commonness with the number k of the divisor and searching for the inverse element by the modul k (the corresponding tables will be designatedwith the letters G, I).

In addition, onecompilesa special table for rapid calculation of the numbers *x*, *y* and a table for holding some small common divisors of the numbers u and$v$, which are cut off at the pilot division stage. All these calculations do not depend on the numbers u and $v$transferred to the algorithm, and therefore can be performed in advance and only once.

The first "trial" division: as it was mentioned earlier, at this stage all common dividers of the given numbers $u$and $v$are firstly cut off. Later they are stored in a separate table, up to the stage of the second test division. Also at this stage, those numbers being the dividers of one of the numbers$u$and v and not the divisors of the other are cut off. Such divisors do not participate in the formation of GCD, so one does not need to store them. Then one executes the main cycle during which the number $M$is calculated. This number is not necessarily equal to the desired GCD, but is equal to their multiple, that is, it can contain extraneous factors. To exclude them, the second test division is performed.

The second "trial" division: this stage of the algorithm is the final one and uses the result obtained after the realization of the main part of the algorithm. To get rid of extraneous factors, divide $M$by all small divisors in a pre-compiled table, and then$M$ is multiplied by the result of the first test division. As a result, we obtain GCD of the given numbers u and v.

The most significant part of the algorithm is in the main loop, which represents the main stage of the algorithm. In C #, it can be written as follows:

```
privatevoidMainLoop()
{
```


Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

```
int u1, v1;
int a, b, x;
BigInteger t;
while ((u != 0) && (v != 0))
{   u1 = (int)(u % k);  v1 = (int)(v % k);
if (G(u1) > 1)   u /= G(u1);
else
{  if (G(v1) > 1)   v /= G(v1);
else
{  x = (u1 * I(v1)) % k;
if (x < 0)   x += k;
a = A(x);
b = ((-1) * a * x) % k;
t = BigInteger.Abs(a * u + b * v) / k;
if (u > v)    u = t;
else  v = t;    }}}
}
```

Variables G, I, A denote the arrays the dimension of which is equal to the base (“arity”) of the algorithm “k”. These arrays realize the tables of preliminary calculations.
Let usaddtherealizationofthestagesofthe first and second trial divisions and obtain the function that realizes the k-ary algorithm:

```
// First trial division
PrivatevoidTrialDivision1()
{
BigIntegerg = 1;
for (inti =1; i<P.Length; i++)
while (u % P(i) == 0 &&v % P(i) == 0)
{    u /= P(i);v /= P(i);g *= P(i);      }
}
//Second trial division
PrivatevoidTrialDivision2()
```


Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

```
{
BigInteger t;
if (v == 0)    t = u;
else     t = v;
for (inti = 0; i<P.Length; i++)
while (t % P(i) == 0)
t /= P(i);
g = t * g;
}
// k-ary algorithm
PublicBigInteger Method(BigInteger number1, BigInteger number2)
{
if (number1 <= 1 || number2 <= 1)    return 1;
TrialDivision1();
MainLoop();
TrialDivision2();
returnGetGCD();
}
```

The complexity of the k-ary algorithm for the numbers $u$and$v$ with length $n$bits in the optimal case of the choice of the parameter k is equal to Sorenson's estimate $O(n^2/\log k)$.

APPROXIMATING K-ARY ALGORITHM. Let there be given natural numbers $A > B > 0$. For simplicity, suppose that A, B are odd. We choose a parameter k to be equal to the power of two $k = 2^s$. It is assumed that $A, B$are sufficiently long, therefore, to simplify further calculations, we find

$a = A mod k,\ b = B mod k.$

Now, by analogy with Sorenson's k-ary algorithm, it is required to find such numbers x and y that

$Ax + By \equiv 0\ mod k,$

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

.and then calculate$C = (Ax + By)/k$, replace $(A, B)$with the new pair (B, C) and pass on to the next iteration.

The approximating k-ary algorithm offers a new way of calculating the numbers x, y, using the approximate value of r, calculated by the formula

$$r = \left\lceil \frac{r' - q}{k} \right\rceil,$$

where $r' = A/B, q = AB^{-1} mod\ k,$ $\lceil s \rceil$ – integer parts.

First, an approximate value $r' = A/B$is calculated to $1/k$. Then the accuracy of computing *r* is$1/k^2$. Then approximate *r*accurate within$3/2k^2$by the fraction of Farey ((Hardy G. H., Wright E. M.1959), chapter 2). The Farey fraction $m/n$is a proper fraction with a numerator and denominator smaller than k. Then, the difference between $m/n$and r does not exceed $5/2k^2$.

The most significant part of the algorithm is the calculation of the approximating fraction m/n. The required values of the coefficients *x, y* are chosen by *m* and *n* as follows: *x* = *n*, *y* is chosen so as to minimize the sum$Ax + By$. Obviously, *y* must be a negative number, then the terms $Ax$and By have different signs and mutually cancelled. The peculiarity of this algorithm is that it operates not only with the very numbers, but also with the length of the binary representation of each of the numbers. To apply the approximation to the above relation, it is necessary that the numbers A, B be sufficiently approximate in length. Otherwise, the usual iteration of the classical Euclidean algorithm is performed, which, in the absence of approximation, speeds up the algorithm (the same substitution also accelerates the Sorenson algorithm).

Let the input data of the algorithm be odd numbers A, B, the basis of the algorithm $k = 2^s$, and the lengths of the numbers A, B, k in the binary representation are L1, L2 and L, respectively. Then the realization of one step of the approximating k-ary algorithm can have the following form:

```
PublicBigInteger Step(BigInteger A, BigInteger B)
{
// Item 1
int d = L1 - L2;
```


Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

```
BigInteger q;
if (d >= 2 * L){
q = BigInteger.Remainder(A * ReverseElement(B,
BigInteger.Pow(2, d)), BigInteger.Pow(2, d));
C = (A - q * B) / BigInteger.Pow(2, d);
if (C == 0)     C = B;
while (C % 2 == 0)     C /= 2;
returnBigInteger.Abs(C);       }
// Item 2
int a = (int)(A % k);
int b = (int)(B % k);
q = (int)((a * ReverseElement(b, k)) % k);
// Item 3
int t = L2 - L;
if (t <= 0){ B = BigInteger.GreatestCommonDivisor(A, B);    return B;   }
BigInteger A1 = BigInteger.Divide(A, BigInteger.Pow(2, t));
BigInteger B1 = BigInteger.Divide(B, BigInteger.Pow(2, t));
// Item 4
BigInteger n0 = B1 * k;
BigInteger m1 = A1 - ((int)q) * B1;
if (m1 < 0)  { q -= k;    m1 = (int)(A1 - ((int)q) * B1);   }
// Item 5
BigInteger m0 = m1 % n0;
BigInteger s1 = (m1 - m0) / n0;
// Item 6
BigInteger x = 1;
BigInteger y = -(((int)q) + s1 * k);
BigInteger m = 1, n = 1;
bool flag = false;
if (m0 * k >= n0){ FareyApproximation(m0, n0, out m, out n);     flag = true;  }
// Item 7
if (flag){ x = n;BigInteger s = -(s1 * x + m);   y = s * k - ((int)q) * x; }
```


Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

```
BigInteger A2 = (A - a) / k;
BigInteger B2 = (B - b) / k;
BigInteger c = (a * x + b * y) / k;
C = A2 * x + B2 * y + c;
// Item 8
if (C == 0)    C = B;
else    {   while (C % 2 == 0)  C = C / 2;  }
returnBigInteger.Abs(C); }
```

According to the theorem on the asymptotic complexity of the approximating k-ary algorithm, the number of iterations in each stage is estimated by$O(n/\log_2 k)$, where n is the length of the original numbers in bits. The complexity of the whole algorithm is estimated by

$$O(n^2/L + nL),$$

where$L$ – binary length$k$.

5. SUMMARY

Analyzing the results of the original and approximating K-ary algorithms, one can note that the optimal value of their “arity” of k, i.e. the value at which the algorithm is realizedby the numbers of a given length in the shortest time varies with the length of the given numbers. This observation is more clearly illustrated in the diagrams below:
Diagram 1.The K-ary Sorensen Algorithm

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

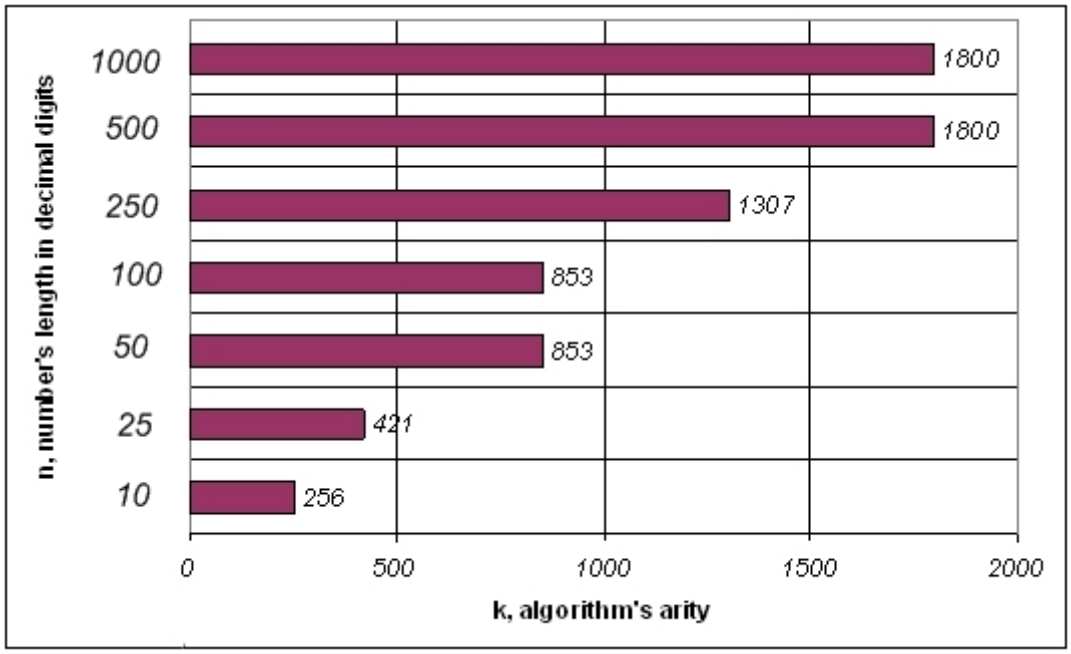
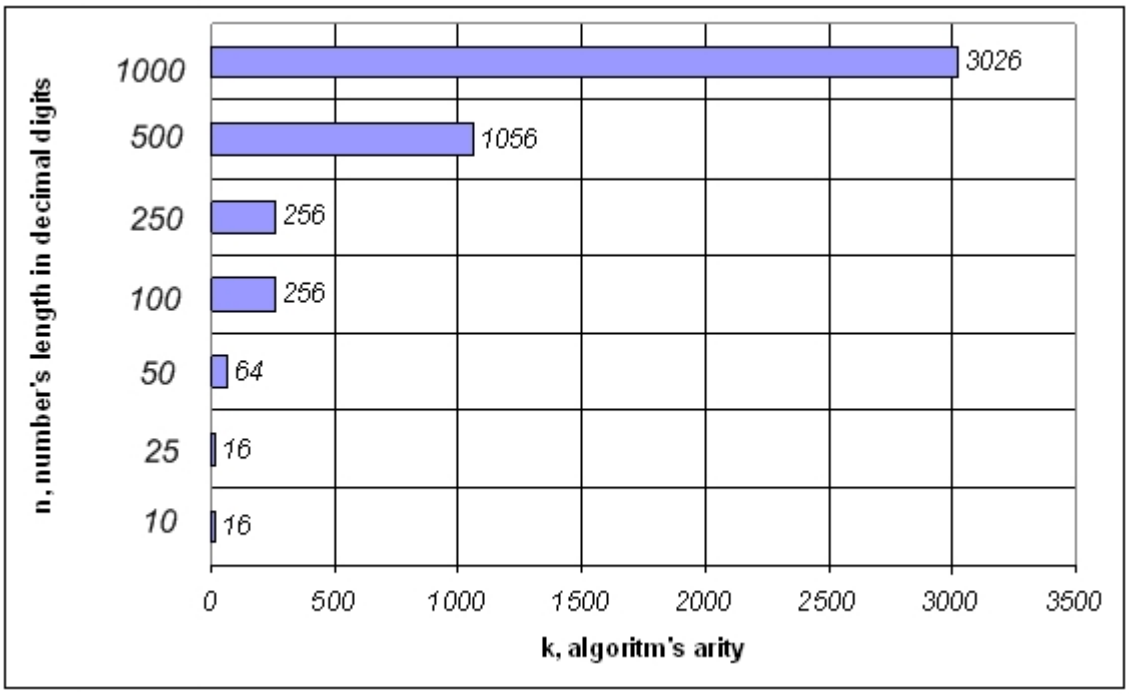


Diagram 2. Approximating K-ary algorithm



When investigating the dependence of the number of steps performed by the algorithm in the search for GCD, on the length of the given numbers, we can conclude that when searching for the GCD of two given numbers, the number of steps made by the approximating k-ary algorithm is the smallest. Following this indicator is k-arny

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

algorithm by Sorenson, and in this case, the “arities” of the algorithms are considered to be equal.The classical Euclidean algorithm takes the most number of steps for the GCD search. The difference in this parameter is noticeable even with a small base of the k (“arity”). Let’s visualize this fact in the diagrams, choosing “arity” of the original and approximating algorithms equal to 64. We can note one more fact connected with the change in the algorithm “arity” in calculating GCD numbers of the given length. With increasing “arity” of the original or approximating algorithm, the GCD search is performed in fewer steps.

Below, the following is designated in diagrams 3,4,5:

K – the classical Euclidean algorithm,

C – Sorensen’s k-ary algorithm (in brackets “arity”)

A – Approximating k-ary algorithm (in brackets “arity”)

Diagram 3. Number of steps within the main cycle

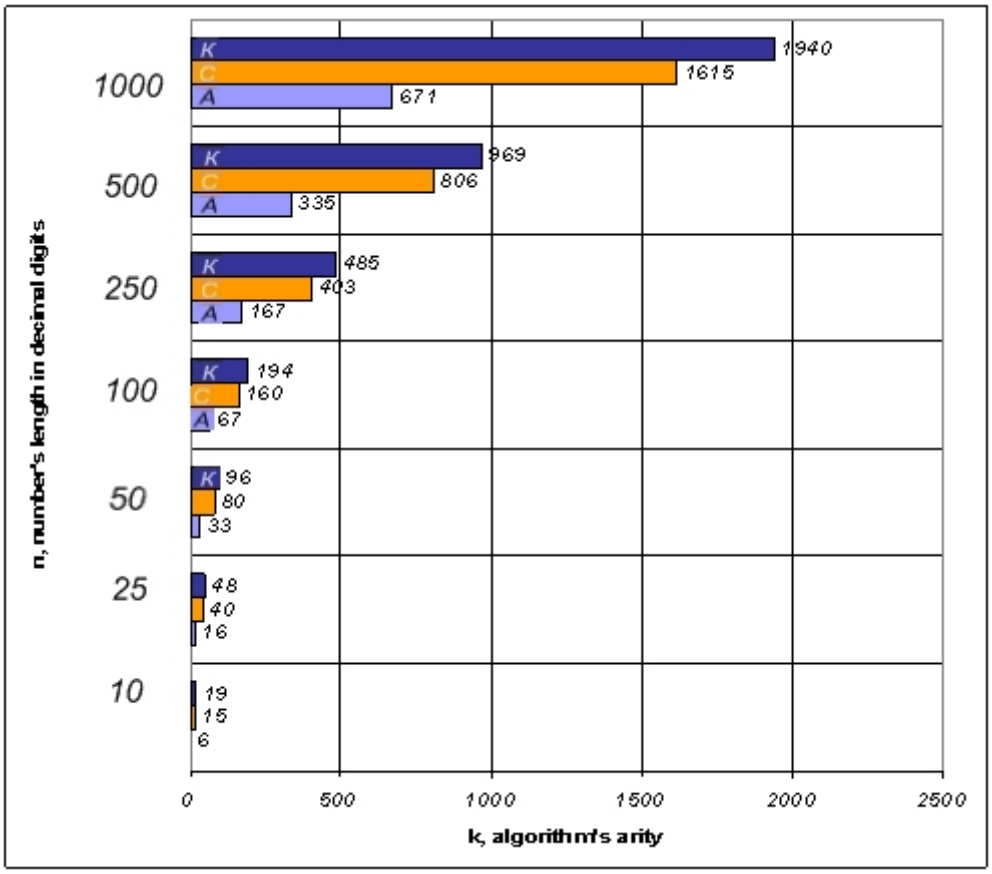


Diagram 4. 64-aryand 408-ary algorithms of Sorenson

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

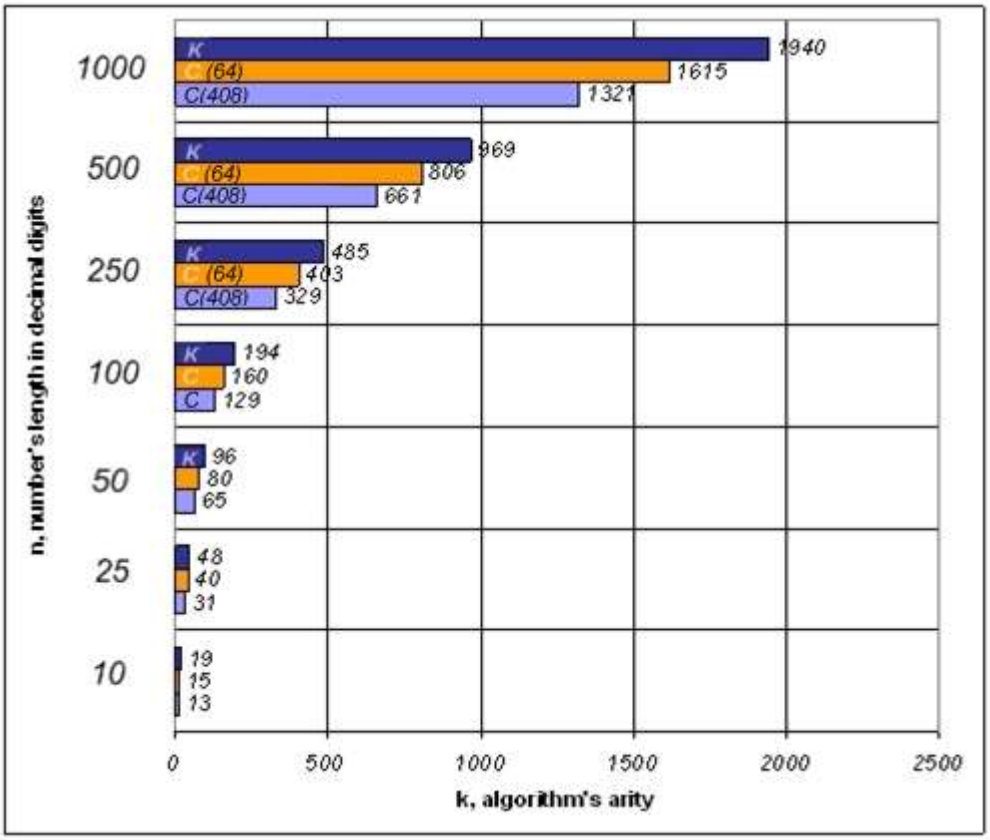
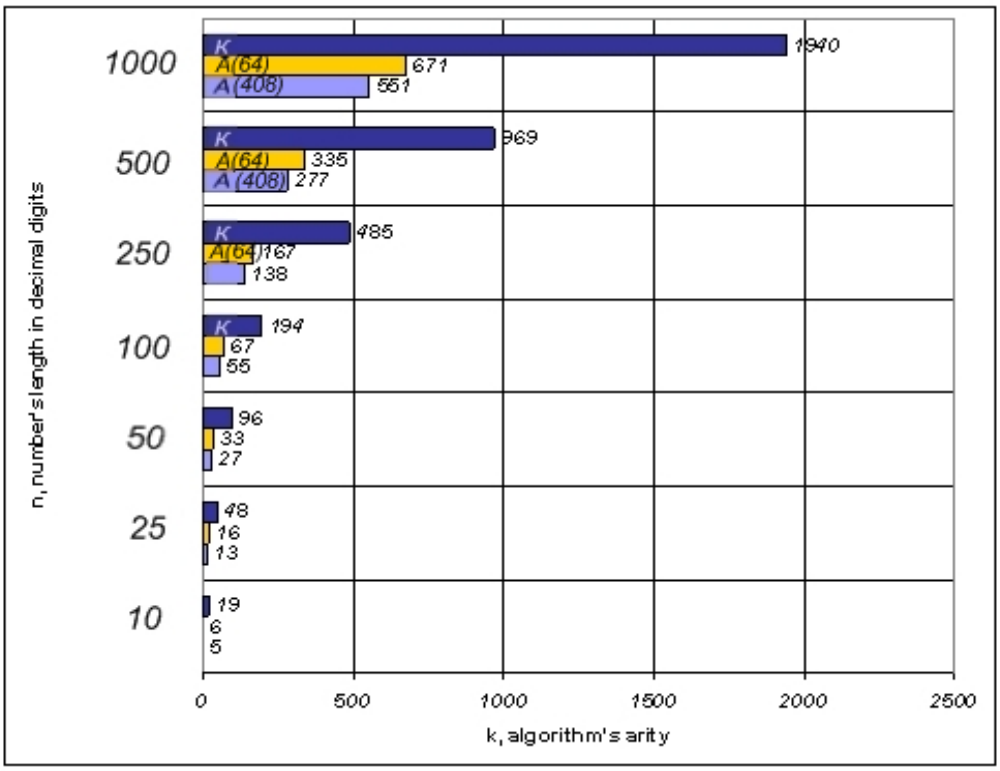


Diagram 5. 64-aryand 408-ary approximating algorithms



Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

However, this circumstance is not areason for always taking the maximum possible basis to calculate GCD by means of the k-ary algorithm. From a direct analysis of the tables based on the results of k-ary algorithms, it follows that an increase in the basis of the algorithm, without detriment to its effectiveness on time, can occur only up to some boundary (Diagrams 6 and 7)

Diagram 6.

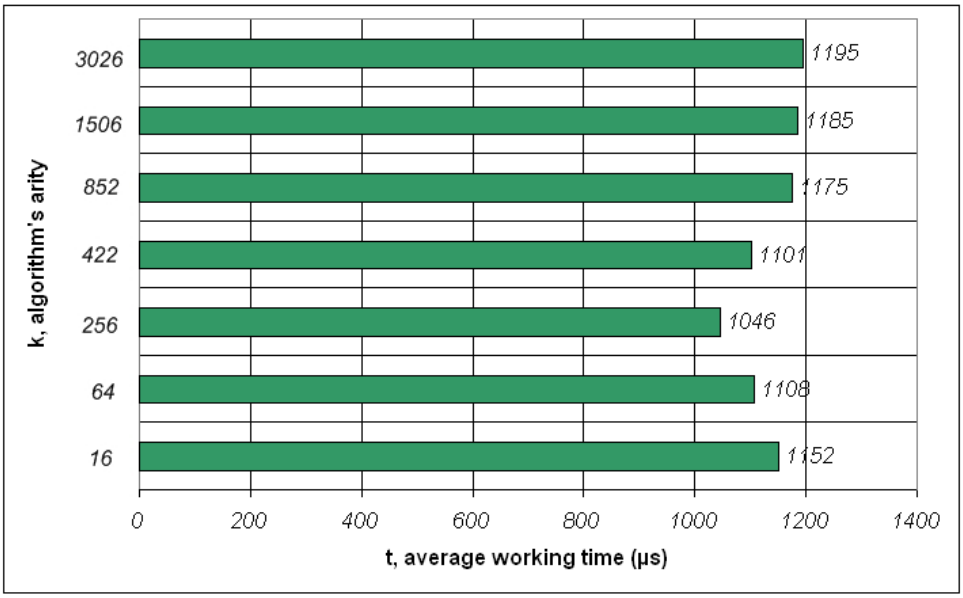
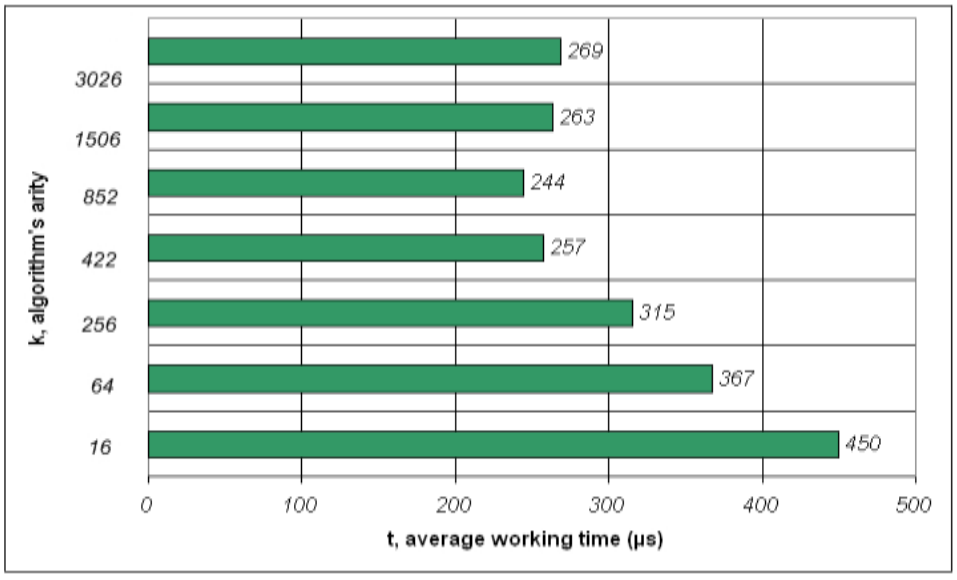


Diagram 7.



6. CONCLUSIONS

Taking into account the peculiarities of the original and approximating k-ary algorithms, as well as the classical Euclidean algorithm, it is possible to propose using

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

them sequentially, at that, first, several iterations of the approximating k-ary algorithm are applied, and the result is input to the k-ary algorithm of Sorenson or the classical Euclidean algorithm. After applying the approximating algorithm in the first stage, the length of the pair of obtained numbers with the equivalent GCD is already substantially smaller than the length of the original numbers. After a few iterations, a pair of relatively small length numbers will be obtained, and now the GCD calculation can be completed with the help of two other algorithms that work much faster than the approximating algorithm начислахwith small numbers.

7. ACKNOWLEDGEMENTS

The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

8. REFERENCES

DixonJ. ThenumberofstepsintheEuclidean algorithm // Journal of Number Theory. vol. 2, pp. 414–422, 1970.

Hardy G. H., Wright E. M. An introduction to the theory of numbers, 4th ed. (Oxford, Calrendon Press), 1959.

Ishmukhametov S.T. An approximating k-ary GCD Algorithm, Lobachevskii Journal of Mathematics, vol. 37, Issue 6, pp. 723-728, 2016.

Ishmukhametov S.T. Factorization methods of natural numbers // Kazan Federal University, Kazan (rus), 2011.

Ishmukhametov S., Mubarakov B., Mochalov A. Euclidian algorithm for recurrent sequences, Applied Discrete Mathematics and Heuristic Algorithms // International Scientific Journal. – Samara, vol. 1(2). – pp. 57–62, 2015.

Sorenson J. An analysis of the generalized binary GCD algorithm / J. Sorenson, A. van derPoorten, A. Stein (Eds.), High Primes and Misdemeanors// Lectures in Honour of Hugh Cowie Williams. – Banff, Alberta, Canada. – AMS Math. Review, vol. 41,pp. 254–258, 2004.

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017

Sorenson J. The k-ary GCD algorithm //Computer Sciences Technical Report. – 1990.

Sorenson J. Two fast GCD Algorithms // Journal of Algorithms, vol. 16(1), pp 110–144, 1994.

Weber K. The accelerated integer GCD algorithm, ACM Trans.Math.Software, 21, №1, pp. 1–12, 1995.

Jebelean T. A Generalization of the Binary GCD Algorithm, Proc. OfIntern.Symp.onSymb.and Algebra, Comp.(ISSAC'93), pp. 111-116, 1993.

Wang X., Pan V. Acceleration of Euclidian Algorithm and rational number reconstruction. Siam J. Comp,vol.32,№2, pp. 548-556, 2003.

Artículo recibido: 01-11-2017
Aprobación definitiva: 07 -12-2017