A Comparison of Several Greatest Common Divisor (GCD) Algorithms

Haroon Altarawneh
Albalqa Applied University
Salt, Jordan

ABSTRACT
This paper about Greater Common Divisor GCD, the paper shows that there is a lot of algorithms, some of these algorithm is good in timing and make low number of iteration, the other make a lot of iteration with a lot of time! But as we see in the analysis of the algorithms that some of the algorithms is faster than the others in small numbers (like Brute force is faster than Bishop Algorithm in the small numbers, but in the large numbers the Bishop Algorithm is too fast with comparison with the brute force) so the researchers recommend to develop the Bishop algorithm the make it more efficient in computing the GCD for small numbers. In the other hand the Dijkstra algorithm is too close in timing and number of iteration with the Bishop algorithm. But as we see in the analysis the best algorithm to use in computing the GCD in all type of integers is the Extended Euclidean algorithm which makes few loops with small or large numbers.

Keywords
Brute Force Algorithm, Dijkstras Algorithm., Extended Euclidean Algorithm, Lehmers GCD Algorithm, Bishops Method for GCD , Fibonacci GCD's.

1. INTRODUCTION
In this paper the researchers will present and analysis the next algorithms of the Greatest Common Divisor (GCD):

1- Brute Force Algorithm.
2- Dijkstras Algorithm.
3- Extended Euclidean Algorithm.
4- Lehmers GCD Algorithm.
5- Bishops Method for GCD.
6- Fibonacci GCD's.

The research will start about history of GCD algorithms, the definition of the GCD, Some Properties of GCD; discuss the selected algorithms for computing GCD, and analysis of these algorithmsIN 300 B.C, the Euclids method for finding GCD for two integers was described. This simple algorithm is often regarded as the grandfather of all algorithms in number theory today. This algorithm will not be discussed in this paper, but we will discuss the mentioned algorithms in the introduction [1].

The **Greatest Common Divisor (GCD)** of two integers A and B, not both zero, is the largest integer that divides both of them. It is convenient to set GCD(0,0) = 0.We assume that A and B are nonnegative integers.

For example: GCD(3,5) = 1 , but GCD(3,6) = 3 , that means 3 and 5 divide on the common integer 1 which is the greatest integer that divides the two numbers, so with 3 and 6 , 1 divide 3 and 6 but 3 divide 3 and 6 and 3 is greater than 1 then the GCD(3,6) is 3 not 1. you can analysis the number to make sure of the GCD, as example 6 and 12:

6 : 1 3 6

12 : 1 2 3 4 6 12

So the GCD (5, 12) is 6 as shown above. The GCD(A,B,C,….) can be also defined for more than two integers as the largest divisor positive integer shared by all of the number A,B,C…. [2]

And here are Some Properties of GCD:

1- Every common divisor of A and B divides GCD of A and B.
2- If m is ant integer, the GCD(mA,mB) = m GCD(A,B).
EX: GCD(2,3) = 1 , GCD(2*2,3*2) = 2* GCD(2,3) = 1*2 = 2 where GCD(4,6) = 2.

3- If m is a nonzero common divisor of A and B then GCD(A/m,B/m) = GCD(A,B)/m.
EX: GCD(6,12) =6 , 6 and 12 divide on 2, GCD(6/2,12/2) = GCD(3,6) = 3 equal to GCD(6,12) = 6/2 = 3.

4- GCD of three numbers GCD(A,B,C) can be computed as GCD(GCD(A,B),C) = GCD(A,GCD(B,C)).
EX: GCD(2,4,6) = 2 = GCD(GCD(2,4) ,6) = GCD(2,6) = 2 = GCD(2,GCD(4,6))= GCD(2,2) = 2.

5- GCD(A,B) = GCD(B,A).
EX:GCD(6,12)= 6 = GCD(12,6).

6- The GCD of A and B is closely related to their least common multiplier LCM(A,B): where GCD(A,B) * LCM(A,B) = AB.
EX: GCD(18,12) =6 , LCM(18,12) = 36 , 6 * 36 = 216 = 18*12.

7- GCD(A,B) = GCD(A,B+An) for all integers n.
EX: GCD(2,4) = 2 , GCD(2,4+2*2) = GCD(2,8) = 2 , GCD(2,4+2*3) = GCD(2,10) = 2[3].

2. Discussion of Selected Algorithms for Computing GCD

2.1 Brute Force Algorithm:

Definition:

The idea of this algorithm is to try all integers from n down until finding one that divides m and n evenly[4].

Algorithm:

1- Take two positive integers m,n , where m<n.
2- Initialize i=m.
3- While (m mod i <> 0 or n mod i <> 0) do
i=i-1;

GCD=i;

Example:

1- m=6 , n = 12.
2- I=6.
3- 6 mod 6 = 0 and 12 mod 6 = 0 then GCD=6.

Code:

```
var n,m,i : integer;
begin
        m=4; n=11;
While (m mod i <> 0 and n mod i   <> 0)  do
                i=i-1;
        GCD =i;
End.
```

Tracing:

m	N	i
4	11	4
		3
		2
		1

GCD = 1

Program:

```
DIM m, n, i AS INTEGER
INPUT m
INPUT n
i = m
WHILE ((n MOD i <> 0) OR (m MOD i <> 0))
i = i - 1
WEND
PRINT "GCD="; i
```

2.2 Dijkstra's Algorithm:

Definition:

This algorithm is developed by Dijkstra who is a Dutch mathematician and a computer scientist.

The idea of this algorithm is : if m>n, GCD(m,n) is the same as GCD(m-n,n). why ? if m/d both leave no remainder, then (m-n)/d leaves no remainder).

The best thing in this algorithm is that it uses the data structure recursion algorithm[5].

Algorithm:

For m,n >0 , GCD(m,n) = { m if m=n
GCD(m-n,n) if m>n

Example:

1- m=4 , n=4 : GCD(4,4)=4
2- m=6 , n=4 : GCD(6,4) = GCD(6-4,4) = GCD(2,4) = GCD(2,4-2) = GCD(2,2) = 2
3- m=4 , n=8 : GCD(4,8) = GCD(4,8-4) = GCD(4,4) = 4

Code:

```
var n,m : integer;
m=6; n=4;
Begin
```

```
int GCD(int m , int n)
{
if (m=n) return m;
if (m>n)
        return GCD(m-n,n);
Else
        return GCD(m,n-m);
}
```

End.

Tracing:

m	n	GCD
6	4	GCD(6,4)
2	4	GCD(2,4)
2	2	GCD(2,2)
		2

GCD = 2

Program:

```
DECLARE FUNCTION GCD (n1 AS INTEGER, n2 AS INTEGER)
        DIM m,n AS INTEGER
        INPUT n
        INPUT m
        PRINT GCD(n, m)
        '------------------------- end of main program
        FUNCTION GCD (n1 AS INTEGER, n2 AS INTEGER)
        IF n1 = n2 THEN GCD = n1
        IF n1 > n2 THEN
        GCD = GCD((n1 - n2), n2)
        ELSE
        GCD = GCD(n1, n2 - n1)
        END IF
END FUNCTION
```

2.3 Extended Euclidean Algorithm:

Definition:

It is a version of the Euclidean algorithm, input of this algorithm is two positive integers m ,n , the algorithm computer the GCD as well as integers A and B such that Am + Bn = GCD(m,n)[6].

Algorithm:

1- input two positive integers m,n
2- initialize: A0=1 , A1=0 , B0=0 , B1=1 , R0=m , R1=n
3- While R1 <>0 do
{

q= [R0/R1]

temp=A0-A1*q , A0=A1 , A1=temp

temp=B0-B1*q , B0=B1 , B1=temp

temp=R0-R1*q , R0=R1 , R1=temp

}

Return A=A0 , B=B0 , g=R0 (the GCD)

Example:

1- m=5 , n=15
2- A0=1 , A1=0 , B0=0 , B1=1 , R0=5 , R1=15
3- R1<>0
q= [5/15] = 0

temp=1-0*0=1 , A0=0 , A1=1

temp=0-1*0=0 , B0=1 , B1=0

temp=5-15*0 = 5 , R0=15 , R1=5

R1<>0

Q=[15/5]=3

temp=0-1*3=-3 , A0=1 , A1=-3

temp=1-0*3 = 1 , B0=0 , B1=1

temp=15-5*3=0 , R0=5 , R1=0

R1=0 exit loop

A=A0=1 , B=B0=0 , g=R0= 5 (GCD)

Am + Bn = GCD(m,n)

1*5 + 0*15 = 5 is true

Code:

```
var n,m,A0,A1,B0,B1,R0,R1,temp : integer;
m=5; n=15;
Begin
A0=1 ; A1=0 ; B0=0 ; B1=1 ; R0=m ; R1=n;
While R1 <>0 do
{
        q= [R0/R1];
        temp=A0-A1*q ;
        A0=A1;
        A1=temp;
        temp=B0-B1*q;
        B0=B1;
        B1=temp;
        temp=R0-R1*q;
        R0=R1;
        R1=temp;
}
Return “A=”,A0,”B=”,B0,”GCD=”,R0;
```

Tracing:

M	n	A0	A1	B0	B1	R0	R1	q
5	15	1	0	0	1	5	15	0
-	-	0	1	1	0	15	5	3
-	-	1	-3	0	1	5	0	-

Program:

```
DIM m, n, a0, a1, b0, b1, r0, r1, temp AS INTEGER
INPUT m
INPUT n
a0 = 1 : a1 = 0 : b0 = 0 : b1 = 1 : r0 = m : r1 = n
DO WHILE r1 <> 0
        q = INT(r0 / r1)
        temp = a0 - a1 * q: a0 = a1: a1 = temp
        temp = b0 - b1 * q: b0 = b1: b1 = temp
        temp = r0 - r1 * q: r0 = r1: r1 = temp
LOOP
a = a0 : b = b0 : g = r0
PRINT "GCD=", r0
```

2.4 Lehmers GCD Algorithm:

Definition: An alternate approach to speeding up euclids algorithm is due to lehmer. One notices that when a and b have the same size, the integer part w of the a/b is often single digit.

Assume that a,b are very big numbers, ^a,^b are small numbers such that: a/b = ^a/^b

then the sequence of quotients produced by EA(a,b) and by EA(^a,^b) will be the same in the beginning , as long as this holds one may compute EA(^a,b) instead of EA(a,b) which is much more economical[7].

Algorithm:

1- INPUT: two positive integers x and y in radix b representation, with x >= y.
2- OUTPUT: gcd(x; y).
   1. While y >= b do the following:

   1.1 Set x^, y^ to be the high-order digit of x, y, respectively (y^ could be 0).

   1.2 A=1, B=0, C=0,=D=1.

   1.3 While (y^ + C) <> 0 and (y^ + D) <> 0 do the following:

   q=floor((x^ + A)=(y^ + C)), q^=(x^ + B)=(y^ + D)

   If q <> q^ then go to step 1.4.

   t=A - qC, A=C, C=t, t=B - qD, B=D, D=t.

t=x^ - qy^, x^=y^, y^=t.

1.4 If B = 0, then t=x mod y, x=y, y=t ;

otherwise, t= Ax + By, u=Cx + Dy, x=t , y=u.

2. Compute v = gcd(x; y) using Euclids Algorithm:

input x,y where x > y

while b <> 0 do the following:

r = a mod b , a=b , b=r

3. Return(v).

Example & Tracing :

x = 768454923 , y = 542167814 , $b=10^3$

X	Y	q	q^	Precision
768 454 923	542 167 814	1	1	Single (T1)
89 593 596	47 099 917	1	1	Single (T2)
42 493 679	4 606 238	10	8	Multiple
4 606 238	1 037 537	5	2	Multiple
1 037 537	456 090	-	-	Multiple
456 090	125 357	3	3	Single (T3)
34 681	10 657	3	3	Single (T4)
10 657	2 710	5	3	Multiple
2 710	2 527	1	0	Multiple
2 527	183			(Euclids)
183	148			(Euclids)
148	35			(Euclids)
35	8			(Euclids)
8	3			(Euclids)
3	2			(Euclids)
2	1			(Euclids)
1	0			(Euclids)

(T1)

X	Y	A	B	C	D	q1	q2
768	542	1	0	0	1	1	1
542	226	0	1	1	-1	2	2
226	90	1	-1	-2	3	2	2
90	46	-2	3	5	-7	1	2

(T2)

X	Y	A	B	C	D	q1	q2
89	47	1	0	0	1	1	1
47	42	0	1	1	-1	1	1
42	5	1	-1	-1	2	10	5

(T3)

X	Y	A	B	C	D	q1	q2
456	125	1	0	0	1	3	3
125	81	0	1	1	-3	1	1
81	44	1	-3	-1	4	1	1
44	37	-1	4	2	-7	1	1
37	7	2	-7	-3	11	9	1

(T4)

X	Y	A	B	C	D	q1	q2
34	10	1	0	0	1	3	3
10	4	0	1	1	-3	2	11

Code:

```
Var b,x,y,xh,yh,q,qh,A,B,C,D : integer;
Begin
X = 768454923 , y = 542167814 , b=10^3 ;
While (y >= b) do
{
```

```
xh= HOD(x,b);
yh= HOD(y,b);
A=1; B=0; C=0;=D=1;
While ((yh+C) <> 0 and (yh + D) <> 0) do
{
    q=floor((xh + A)=(yh + C));
    qh=(xh + B)=(yh + D);
    If q = qh then
    {
        t=A – q*C; A=C; C=t; t=B – q*D; B=D; D=t;
        t=xh - q*yh; xh=yh; yh=t;
}
    else
        If B = 0 then
        t= x mod y; x=y; y=t ;
        Else
        t= A*x + B*y; u=C*x + D*y; x=t ; y=u;
}
while b <> 0 do
{
r = a mod b ;
a=b; b=r;
}
```

2.5 Bishop's Method for GCD:

Definition:

If a large and small numbers are both multiples of K , them large – small is a multiple of K. note that large-small is smaller than large, so we have reduced the problem to one easier to solve. So we need greatest multiple of large-small and small … and so on [8].

Algorithm:

1- input two positive integers X,Y
2- while (x <> y) do
   if (x>y) then

```
        x=x-y
    else
    {
        temp = y
        y=x
        x=temp
    }
    return(x)
```

Example:

1- X=30 , Y= 45
2- X <> Y
   Is X > Y no

   Temp = 45

   Y=30

   X=45

   X <> Y

   Is X > Y yes

   X=45-30 = 15

   X <> Y 15 <> 30

   Is X>Y no

   Temp = 30

   Y=15

   X=30

   X<>Y 30 <> 15

   Is X>Y yes

   X=15-30 = 15

   X = Y exit loop GCD=X=Y=15

Code:

```
var x,y, temp : integer;
Begin
    X=30 ; y = 45 ; temp=0;
    While X <> Y do
```

```
{
        if X > Y then
                X=X-Y;
        Else
                Temp=Y;
                Y=X;
                X=Temp;
        End if
}
cout<<"GCD=",X;
End.
```

Tracing:

X	Y
30	45
45	30
15	30
30	15
15	15

Program:

```
DIM X, Y, TEMP AS INTEGER
INPUT X
INPUT Y
DO WHILE X <> Y
IF X > Y THEN
X = X - Y
ELSE
TEMP = Y
Y = X
X = TEMP
END IF
LOOP
PRINT "GCD="; X
```

2.6 Fibonacci GCD's:

Definition:

Fibonacci numbers are then numbers in the series: n1, n2, n3=n1+n2,n4=n2+n3,n5=n3+n4,… so that the numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … the Fibonacci GCDs are the GCD of two Fibonacci numbers and the result is also a Fibonacci number[8].

Algorithm and Example:

You can use ant of the GCDs algorithm to calculate the GCD, this algorithm is only to denote that the result is a Fibonacci number. As example we use the euclids GCD algorithm:

F1= 34 , f2 = 8

GCD(34,8)

34 mod 8 = 2

8 mod 2 = 0

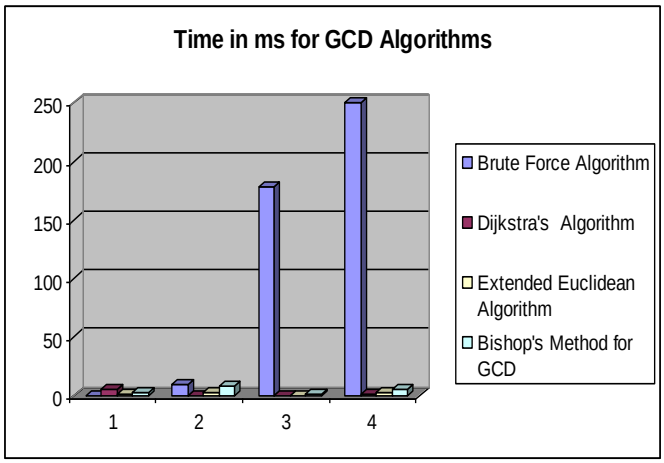
So GCD(34,8) = 2 which is a Fibonacci number.

3. Analysis of algorithms:

3.1 Time Consume:

The next table and graph show the timing in MS for the algorithms for several integer numbers length:

Algorithm	2 Digits	4 digits	6 digits	8 digis
Brute Force Algorithm	0 ms	9 ms	178 ms	Too long
Dijkstra's Algorithm	5 ms	2 ms	0 ms	1 ms
Extended Euclidean Algorithm	1 ms	1 ms	0 ms	2 ms
Bishop's Method for GCD	3 ms	8 ms	1 ms	5 ms

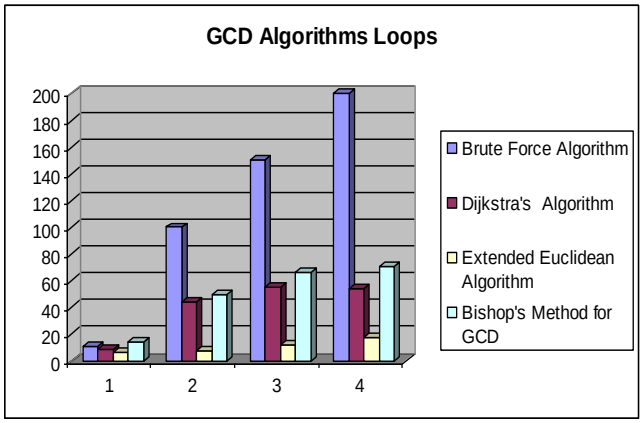


As seen in the above graph that the brute force algorithm in number that are 6 and more digits it toke a lot of time to get the result, in the other hand the Extended Euclidean algorithm takes less time when the numbers are more than 6 digits.

3.1 Number of loops:

The next table and graph show the number of loops for the algorithms for several integer numbers length:

Algorithm	2 Digits	4 digits	6 digits	8 digis
Brute Force Algorithm	11	1232	125472	1236548
Dijkstra's Algorithm	9	44	55	54
Extended Euclidean Algorithm	6	7	12	18
Bishop's Method for GCD	14	50	66	71



The graph above shows the number of loops that the algorithms take to find the answer. The Brute force algorithm make a lot of loops to get the answer in the numbers that more than 2 digits, in the other hand the Extended Euclidean algorithm make few loops to get the answer.

4. CONCLUSIONS:

As seen in this paper about Greater Common Divisor GCD that there is a lot of algorithms, some of these algorithm is good in timing and make low number of iteration, the other make a lot of iteration with a lot of time! But as we see in the analysis of the algorithms that some of the algorithms is faster than the others in small numbers (like Brute force is faster than Bishop Algorithm in the small numbers, but in the large numbers the Bishop Algorithm is too fast with comparison with the brute force) so the researchers recommend to develop the Bishop algorithm the make it more efficient in computing the GCD for small numbers.

5. REFERENCES

[1] Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.

[2] eLiteral The Moving Constant available at: http://services.eliteral.com/digital-certificate-mumbai/chap14.php

[3] Washington University, St. Louis available a: http://www.cs.wustl.edu/~kjg/cs101/Notes/Recursion/recursion.html

[4] Centre for Information Security and Cryptography available at: http://cisac.math.ucalgary.ca/news_events/hugh/talks/sorenson.pdf

[5] National Institute of Standards and Technology available at: http://www.nist.gov/dads/

[6] Computer Science Ben Gurion University of the Negev available at: http://www.cs.bgu.ac.il/~berend/teaching/Intro2CS/examples/main.html