.

COMPARACION ENTRE ALGORITMOS DE MAXIMO COMUN DIVISOR (GCD)

Nicole Alexia Durand Zeballos

RESUMEN: *Para cada uno de los algoritmos se calculó el máximo común divisor para los enteros 412 y 260, como ejemplo, realizando un seguimiento numérico manual, indicando la variación de las variables paso a paso. Además, se implementó cada uno de los algoritmos, mostrando en pantalla paso a paso, Finalmente se analiza cada uno de los algoritmos en tiempo y numero de vueltas (recursivas o de bucle) realiza el algoritmo, respecto a números de diferentes de cifras (cinco, diez, quince y veinte), y naturaleza (par e impar), para determinar las ventajas y desventajas de cada uno de los algoritmos, además de concluir cuál de ellos es más eficiente, es decir cual converge más rápido.*

PALABRAS CLAVE: Euclides, Máximo común divisor, gcd binario.

1 INTRODUCCIÓN

El máximo común divisor (GCD) de dos enteros A y B, no ambos cero, es el entero más grande que los divide a ambos. Es conveniente establecer GCD (0,0) = 0. Suponemos que A y B son enteros no negativos.

En este paper se presentan 7 algoritmos de máximo común divisor, para los cuales se incluye un pseudocódigo, ejemplo, código mejorado en C++, y resultados de sus pruebas respecto al tiempo y numero de vueltas, para números de distintas cifras y naturaleza

2 ALGORITMOS

2.1 ALGORITMO 1

Este algoritmo calcula el modulo o resto entre los valores iniciales y lo utiliza recursivamente hasta encontrar el gcd, hasta que este sea 0.

Pseudocódigo:

1. Leer a y b
2. Calcular el resto, r de a y b
3. Si el resto es igual a 0 entonces:
   El gcd (a,b)=b
   FIN
4. Si no:
   a=b;
   b=r;
   Ir al paso 2;

Código C++:

```
unsigned long long algoritmo1(unsigned long
        long  a, unsigned long long b){
    cout<<"a= "<<a<<"  b= "<<b<<endl;
    unsigned long long r=mod(a,b);
    if (r==0){
        return b;
    }
    else{
        a=b;
        b=r;
        algoritmo1(a,b);
    }
}
```

Ejemplo:

```
a= 412  b= 260
r=mod(412,260)=152;
        a= 260  b= 152
r=mod(260,152)=108;
        a= 152  b= 108
r=mod(152,108)=44;
        a= 108  b= 44
r=mod(108,44)=20;
        a= 44  b= 20
r=mod(44,20)=4;
        a= 20  b= 4
r=mod(20,4)=0;
```

Pruebas:

Tabla 1.Analisis Algoritmo 1, 5 cifras

Algoritmo 1	Tiempo	N° vueltas
Par - Par	0.002 s	10
Par - Impar	0 s	8
Impar - Par	0.001 s	10
Impar – Impar	0.001 s	9

Tabla 2.Analisis Algoritmo 1, 10 cifras

Algoritmo 1	Tiempo	N° vueltas
Par - Par	0 s	19
Par - Impar	0.002 s	20
Impar - Par	0.001 s	18
Impar – Impar	0.001 s	23

.

Tabla 3.Analisis Algoritmo 1, 15 cifras

Algoritmo 1	Tiempo	N° vueltas
Par - Par	0,001 s	27
Par - Impar	0.001 s	32
Impar - Par	0.001 s	30
Impar – Impar	0.001 s	28

Tabla 4.Analisis Algoritmo 1, 20 cifras

Algoritmo 1	Tiempo	N° vueltas
Par - Par	0,002 s	38
Par - Impar	0.001 s	36
Impar - Par	0 s	41
Impar – Impar	0.001 s	40

Análisis:

Este algoritmo demuestra ser eficiente para números pequeños como de 5 cifras, hasta números grandes como 20 cifras, tanto en tiempo, como numero de vueltas, además que los resultados no varían tan radicalmente dependiendo de la naturaleza ni de las cifras de los números evaluados, sin embargo no puede considerarse optimo en un 100%, pues al calcular el resto entre los dos números evaluados se hace 1 división por vuelta en el algoritmo, y ya que la división es costosa computacionalmente eleva los valores del tiempo de ejecución, pero este aumento es compensado con las demás operaciones que son solo comparaciones y asignaciones, en un swap de valores, que no son costosas computacionalmente.

2.2 ALGORITMO 2

Este algoritmo calcula el modulo o resto entre los valores iniciales y lo utiliza recursivamente hasta encontrar el gcd, mientras sea menor a la mitad del número inicial, hasta que este sea 0.

Se mejoró este algoritmo reemplazando las divisiones, por corrimientos en operadores de bits.

Pseudocódigo:

1. Leer a y b
2. Calcular el resto, r de a y b
3. Si el resto es igual a 0, entonces:
   El gcd (a,b)=b
   FIN
4. Si r es mayor a b/2, entonces:
   r=b-r;
5. a=b;
6. b=r;
7. Ir al paso 2;

Código C++:

```
unsigned long long algoritmo2(unsigned long long a,
        unsigned long long b){
  cout<<"a= "<<a<<"  b= "<<b<<endl;
  unsigned long long r=mod(a,b);
  if (r==0){
    return b;
  }
  if(r>(b>>1)){
    r=b-r;
  }
  a=b;
  b=r;
  algoritmo2(a,b);
}
```

Ejemplo:

a= 412 b= 260
r=mod(412,260)=152
a= 260 b=(260-152)= 108
r=mod(260,108)=44
a= 108 b= 44
r=mod(108,44)=20
a= 44 b= 20
r=mod(44,20)=4
a= 20 b= 4
r=mod(20,4)=0

Pruebas:

Tabla 5.Analisis Algoritmo 2, 5 cifras

Algoritmo 2	Tiempo	N° vueltas
Par - Par	0.001 s	7
Par - Impar	0 s	7
Impar - Par	0.001 s	7
Impar – Impar	0.002 s	7

Tabla 6.Analisis Algoritmo 2, 10 cifras

Algoritmo 2	Tiempo	N° vueltas
Par - Par	0 s	14
Par - Impar	0.001 s	15
Impar - Par	0 s	13
Impar – Impar	0.001 s	16

Tabla 7.Analisis Algoritmo 2, 15 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0 s	20
Par – Impar	0.001 s	22
Impar – Par	0.001 s	21

.

Impar – Impar	0.001 s	20

Tabla 8.Analisis Algoritmo 2, 20 cifras

Algoritmo 2	Tiempo	N° vueltas
Par - Par	0,001 s	27
Par - Impar	0.001 s	26
Impar - Par	0.001 s	28
Impar – Impar	0.002 s	27

Análisis:

Este algoritmo no varía mucho en número de vueltas, ni tiempo, respecto a los números evaluados, es decir es eficiente, tanto para números pequeños y grandes, de cualquier naturaleza, sin embargo presenta las mismas limitantes que el algoritmo 1, pues al evaluar el resto entre dos valores, se realiza una división, que es poco optima, por su elevado costo computacional, además de otra división en un condicional, que fue mejorada en el algoritmo por un corrimiento a nivel de bits, evitando el costo de otra división, además de esto presenta comparaciones, swap o intercambio de valores, asignación y resta

2.3 ALGORITMO 3

Es el algoritmo de Euclides para calcular el gcd, pese a ser uno de los más conocidos su eficiencia es puesta en duda seguidamente, pues muchos no lo consideran óptimo.

Pseudocódigo:

1. Leer a y b
2. Si b es igual a 0, entonces:
   El gcd (a,b)=a
   FIN
3. Repetir algoritmo(b,a mod b);

Código C++:

```
unsigned long long EuclidesMCD(unsigned long
        long a, unsigned long long b){
  cout<<"a= "<<a<<"  b= "<<b<<endl;
  if(b==0){
    return a;
  }
  return EuclidesMCD(b,mod(a,b));
}
```

Ejemplo:

```
a= 412  b= 260
        EuclidesMCD(260,mod(412,260)=152)
a= 260  b= 152
        EuclidesMCD(152,mod(260,152)=108)
a= 152  b= 108
        EuclidesMCD(108,mod(152,108)=44)
a= 108  b= 44
        EuclidesMCD(44,mod(108,44)=20)
a= 44  b= 20
        EuclidesMCD(20,mod(44,20)=4)
a= 20  b= 4
        EuclidesMCD(4,mod(20,4)=0)
a= 4  b= 0
```

Pruebas:

Tabla 9.Analisis Algoritmo 3, 5 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.001 s	11
Par - Impar	0 s	9
Impar - Par	0.001 s	11
Impar – Impar	0.001 s	10

Tabla 10.Analisis Algoritmo 3, 10 cifras

Algoritmo 2	Tiempo	N° vueltas
Par - Par	0 s	20
Par - Impar	0.001 s	21
Impar - Par	0.001 s	19
Impar – Impar	0.002 s	24

Tabla 11.Analisis Algoritmo 3, 15 cifras

Algoritmo 2	Tiempo	N° vueltas
Par - Par	0 s	28
Par - Impar	0.001 s	33
Impar - Par	0.001 s	31
Impar – Impar	0.001 s	29

Tabla 12.Analisis Algoritmo 3, 20 cifras

Algoritmo 2	Tiempo	N° vueltas
Par - Par	0,001 s	39
Par - Impar	0.001 s	37
Impar - Par	0.001 s	42
Impar – Impar	0.001 s	41

Análisis:

Es el algoritmo prueba ser eficiente para diferentes números, el tiempo no se ve muy afectado, es casi idéntico entre números pequeños y grandes de diferentes naturalezas, y respecto al número de vueltas si se ve afectado, pero no muy radicalmente, sin embargo este número de vueltas a su vez significa una división por vuelta, pues calcula el modulo o resto entre los valores evaluados, lo que eleva el costo

.

computacional de la función, sin embargo su recursividad la limita solo a una comparación que es el criterio de parada y una división por vuelta.

2.4 ALGORITMO 4

Es el algoritmo binario de gcd recursivo, trabaja los números como binarios.

Se mejoró este algoritmo reemplazando las divisiones y multiplicaciones, por corrimientos en operadores de bits, además de definir si un número es par con el operado and..

Pseudocódigo:

1. Leer a y b, enteros positivos
2. Inicializa un entero g a 1
3. Si b es mayor que a, entonces:
   gcd (b,a)
4. Si b es igual a 0, entonces:
   El gcd (a,b)=a
   FIN
5. Si a y b son pares, entonces:
   2* gcd (a/2,b/2)
6. Si solo a es par, entonces:
   gcd (a/2,b)
7. Si solo b es par, entonces:
   gcd (a,b/2)
8. Si no:
   gcd (|a-b|/2,b)

Código C++:

```
unsigned long long BinaryGcd(unsigned long long a,
        unsigned long long b){
  cout<<"a= "<<a<<"  b= "<<b<<endl;
  if(b>a){
    return BinaryGcd(b,a);
  }
  if(b==0){
    return a;
  }
  if((a&1)==0 && (b&1)==0){
    return (BinaryGcd((a>>1),(b>>1)))<<1;
  }
  if((a&1)==0 &&(b&1)==1){
    return BinaryGcd((a>>1),b);
  }
  if((a&1)==1&&(b&1)==0){
    if(a<0){
      a*=-1;
    }
    if(b<0){
      a*=-1;
    }
    return BinaryGcd(a,(b>>1));
  }
  else{
    return BinaryGcd((a-b)>>1,b);
  }
}
```

Ejemplo:

```
a= 412  b= 260
        2*BinaryGcd(412/2=206,260/2=130)
a= 206  b= 130
        2*BinaryGcd(206/2=103,130/2=65)
a= 103  b= 65
        BinaryGcd((103-65)/2=19, 65)
a= 19  b= 65
        BinaryGcd(65,19)
a= 65  b= 19
        BinaryGcd((65-19)/2=23,19)
a= 23  b= 19
        BinaryGcd((23-19)/2,19)
a= 2  b= 19
        BinaryGcd(19,2)
a= 19  b= 2
        BinaryGcd(19,2/2=1)
a= 19  b= 1
        BinaryGcd((19-1)/2=9,1)
a= 9  b= 1
        BinaryGcd((9-1)/2=4,1)
a= 4  b= 1
        BinaryGcd(4/2=2,1)
a= 2  b= 1
        BinaryGcd(2/2=2,1)
a= 1  b= 1
        BinaryGcd((1-1)/2,1)
a= 0  b= 1
        BinaryGcd(1,0)
a= 1  b= 0
```

Pruebas:

Tabla 13.Analisis Algoritmo 4, 5 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.001 s	26
Par – Impar	0.001 s	30
Impar – Par	0.001 s	27
Impar – Impar	0.001 s	21

Tabla 14Analisis Algoritmo 4, 10 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.002 s	49
Par – Impar	0.003 s	62
Impar – Par	0 s	59
Impar – Impar	0.002 s	62

Tabla 15.Analisis Algoritmo 4, 15 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.001 s	80

.

Par – Impar	0.001 s	85
Impar – Par	0.001 s	84
Impar – Impar	0.002 s	88

Tabla 16.Analisis Algoritmo 4, 20 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0,001 s	109
Par – Impar	0.001 s	113
Impar – Par	0.001 s	118
Impar – Impar	0 s	114

Análisis:

Este algoritmo es muy eficiente respecto al tiempo para diferentes números, respecto a naturaleza y tamaño, sin embargo se ve gran variación respecto al número de vueltas, si se ve gran variación entre los números pequeños y grandes, más no ente números de diferente naturaleza.

Se mejoró este algoritmo reemplazando las divisiones y multiplicaciones, por corrimientos en operadores de bits, además de definir si un número es par con el operador and.

Este algoritmo es uno de los mejores, para trabajar con números grandes, si bien es cierto que el número de vueltas, puede ser algo elevado, este no representa un gran costo computacional, pues por vuelta realiza 5 comparaciones y cumple solo una de ellas, que en el peor de los casos lo lleva a una división, pero que al ser reemplazada por un corrimiento, se ahorra esta operación tan costosa computacionalmente, por lo que pese a dar más vueltas respecto a otros algoritmos, es muy eficiente y poco costoso

2.5 ALGORITMO 5

Es el algoritmo binario de gcd iterativo, trabaja los números como binarios, pero con repetidos bucles while, y while anidados.

Pseudocódigo:

1. Leer a y b, enteros positivos
2. Inicializa un entero g a 1
3. Si b es mayor que a, entonces:
   gcd (b,a)
4. Mientras a y b son pares, entonces:
   a=a/2 b=b/2
   g=g*2
5. Mientras x diferente de 0:
   6. Mientras a es par, entonces:
      a=a/2
   7. Mientras b es par, entonces:
      b=b/2
   8. Un entero t igual a (|a-b|/2)
   9. Si a mayor a b, entonces:
      a=t
   10. Si no:
      b=t

Código C++:

```
unsigned long long algoritmo5(unsigned long long x,
        unsigned long long y){
  unsigned long long g=1;
  if(x<0||y<0){
    return 0;
  }
  while((x&1)==0 &&(y&1)==0){
    cout<<"a= "<<x<<"  b= "<<y<<endl;
    x=(x>>1);
    y=(y>>1);
    g=g<<1;
  }

  while(x!=0){
    cout<<"a= "<<x<<"  b= "<<y<<endl;
    while ((x&1)==0){
      x=(x>>1);
    }
    while ((y&1)==0){
      y=(y>>1);
    }
   unsigned long long t;
   long long temp=(x-y);
    if(temp<0){
      t=-1*(temp>>1);
    }
    else{
     t=((x-y)>>1);
    }
     if(temp<0){

      y=t;
     }
     else{
       x=t;
     }
  }
   cout<<"a= "<<x<<"  b= "<<y<<endl;
  return(g*y);
}
```

Ejemplo:

g=1
a= 412 b= 260
  a=412/2=206
  b=260/2=130
  g=1*2
a= 206 b= 130
  a=206/2=103
  b=130/2=65
  g=2*2
a= 103 b= 65
  t=(103-65)/2=19
  a=t
a= 19 b= 65
  t=-1*((19-65)/2)=23
  b=t
a= 19 b= 23
  t=-1*((19-23)/2)=2
  b=t
a= 19 b= 2
  t=(19-2)/2=9

.

a=t
a= 8 b= 1
a=8/2=4
a= 4 b= 1
a=4/2=2
a= 2 b= 1
a=2/2=1
a= 1 b= 1
t=(1-1)/2=0
a=t
a= 0 b= 1

Pruebas:

Tabla 16.Analisis Algoritmo 5, 5 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0 s	19
Par – Impar	0.016 s	23
Impar – Par	0.015 s	19
Impar – Impar	0.007 s	15

Tabla 17.Analisis Algoritmo 5, 10 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.035 s	38
Par – Impar	0.08 s	49
Impar – Par	0.035 s	45
Impar – Impar	0.025 s	47

Tabla 18.Analisis Algoritmo 5, 15 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.024 s	64
Par – Impar	0.037 s	67
Impar – Par	0.039 s	62
Impar – Impar	0.040 s	65

Tabla 19.Analisis Algoritmo 5, 20 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0,047 s	81
Par – Impar	0.056 s	88
Impar – Par	0.048 s	88
Impar – Impar	0.053 s	90

Análisis:

En este algoritmo se puede observar que el tiempo puede variar respecto a la naturaleza de los números evaluados, como a su tamaño, sin embargo la variación en el número de vueltas es muy ligera respecto a la naturaleza del número, y poca respecto al tamaño del mismo.

En el primer while se realiza comparaciones y dos modulos, o restos, para determinar la paridad de los números evaluados, esto significaría una costosa división, pero fue reemplazado por el operador and a nivel de bits, que ahorra el costo computacional de dos divisiones por vuelta que realice el while, seguidamente se realiza el while más grande del algoritmo, el cual tiene como condición una comparación, sin embargo este while tiene dentro mas while, que evalúan la paridad de los números, donde nuevamente se evita el costo de la división trabajado con operadores de bits, finalmente se asigna la resta de los valores, dividida entre 2, equivalente al corrimiento, y se hace un sawp o cambio de valores. Entonces si este algoritmo lo trabajamos como números normales sin las modificaciones realizadas, sería muy costoso computacionalmente, pues realiza muchas divisiones a lo largo de los while, una en el primero y dos en el segundo, para luego comparar lo valores, y por lo tanto no sería un algoritmo muy recomendado, sin embargo las modificaciones nos ahorran el costo de la división, devolviendo resultados muy similares al algoritmo de gcdBinario recursivo, inclusive con menor número de vueltas, pues dentro de una sola vuelta evalúa dos posibilidades.

2.6 ALGORITMO 6

Es el algoritmo de gcd que se basa en la resta, no realiza divisiones ni es recursivo.

Pseudocódigo:

1. Leer a y b, eneros positivos
2. Mientras a diferente de b:
    3. Si a mayor a b, entonces:
       a=a-b
    4. Sino :
       b=b-a
5. Retorno a

Código C++:

```
unsigned long long algoritmo6(unsigned long long
        a,unsigned long long b){
   cout<<"a= "<<a<<"  b= "<<b<<endl;
   while(a!=b){
      if(a>b){
         a=a-b;
      }
      else{
         b=b-a;
      }
      cout<<"a= "<<a<<"  b= "<<b<<endl;
   }
   return a;
}
```

.

Ejemplo:

a= 412 b= 260
    a=412-260=152
a= 152 b= 260
    b=260-152=108
a= 152 b= 108
    a=152-108=44
a= 44 b= 108
    b=108-44=64
a= 44 b= 64
    b=64-44=20
a= 44 b= 20
    a=44-20=24
a= 24 b= 20
    a=24-20=4
a= 4 b= 20
    b=20-4=16
a= 4 b= 16
    b=16-4=12
a= 4 b= 12
    b=12-4=8
a= 4 b= 8
    b=8-4=4
a= 4 b= 4

Pruebas:

Tabla 20.Analisis Algoritmo 6, 5 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.015 s	21
Par – Impar	0.039 s	35
Impar – Par	0.033 s	25
Impar – Impar	0.012 s	11

Tabla 21.Analisis Algoritmo 6, 10 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.065 s	65
Par – Impar	0.123 s	108
Impar – Par	0.121 s	113
Impar – Impar	0.085 s	76

Tabla 22.Analisis Algoritmo 6, 15 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.148 s	140
Par – Impar	0.172 s	153
Impar – Par	0.084 s	143
Impar – Impar	0.088 s	139

Tabla 23.Analisis Algoritmo 6, 20 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0,151 s	131
Par – Impar	0.203 s	205
Impar – Par	0.301 s	286
Impar – Impar	0.533 s	688

Análisis:

Este algoritmo es muy poco eficiente, pues el crecimiento en tiempo y número de vueltas es muy grande respecto al tamaño de los números evaluados, y también de la naturaleza de los mismos.

Este algoritmo es iterativo con un while, realiza una resta por vuelta, o un swap, cambio de valores, no es muy costoso computacionalmente ya que solo consta de dos comparaciones y una resta, pero es poco eficiente ya que cuando los números son muy grandes la resta no abastece para disminuir el número hasta salir del bucle

2.7 ALGORITMO 7

Es el algoritmo de gcd de Euclides, basado en divicion y resta.

Se mejoró este algoritmo reemplazando las divisiones y multiplicaciones, por corrimientos en operadores de bits.

Pseudocódigo:

1. Leer a y b, enteros positivos
2. Mientras b mayor a 0:
    3. Un entero q=a/b;
    4. Un entero r=a-q*b
    5. a=b
    6. b=r
7. Retorno a

Código C++:

```
unsigned long long mcd(unsigned long long
        a,unsigned long long b, int &cont){
  unsigned long long r1=a;
  unsigned long long r2=b;
  while(r2>0){
    cout<<"a= "<<r1<<"  b= "<<r2<<endl;
    unsigned long long q=r1/r2;
    unsigned long long r=r1-q*r2;
    r1=r2;
    r2=r;
  }
  cout<<"a= "<<r1<<"  b= "<<r2<<endl;
  return r1;
}
```

Ejemplo:

a= 412 b= 260
    q=412/260
    r=412-q*260
    a=b

.

```
        b=r
a= 260  b= 152
        q=260/152
        r=260-q*152
        a=b
        b=r
a= 152  b= 108
        q=152/108
        r=152-q*108
        a=b
        b=r
a= 108  b= 44
        q=108/44
        r=108-q*44
        a=b
        b=r
a= 44  b= 20
        q=44/20
        r=44-q*20
        a=b
        b=r
a= 20  b= 4
        q=20/4
        r=20-q*4
        a=b
        b=r
a= 4  b= 0
```

Pruebas:

Tabla 24.Analisis Algoritmo 7, 5 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.015 s	10
Par – Impar	0 s	8
Impar – Par	0.016 s	10
Impar – Impar	0.015 s	9

Tabla 25.Analisis Algoritmo 7, 10 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.058 s	23
Par – Impar	0.021 s	18
Impar – Par	0.031 s	20
Impar – Impar	0.018 s	19

Tabla 26.Analisis Algoritmo 7, 15 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0.006 s	27
Par – Impar	0.038 s	32
Impar – Par	0.047 s	30
Impar – Impar	0.038 s	28

Tabla 27.Analisis Algoritmo 7, 20 cifras

Algoritmo 2	Tiempo	N° vueltas
Par – Par	0,035 s	38
Par – Impar	0.046 s	36
Impar – Par	0.047 s	41
Impar – Impar	0.038 s	40

Análisis:

Es un algoritmo eficiente, podría decirse que la variación en tiempo y número de vueltas depende del tamaño del número, mas no tiene mucha relación con la naturaleza del mismo.

El algoritmo es muy costoso computacionalmente hablando, presenta una división, una resta y una multiplicación, y dos swap o cambio de valores por cada vuelta del bucle, además de no poder evitar la costosa división, es una formula compuesta, lo que lo hace menos eficiente que otros algoritmos

3 ANALISIS

Se realiza un promedio del tiempo demorado por el algoritmo respecto al tamaño del número y se evaluara en tiempo y número de vueltas.

Tabla 27. Grafica 1: Algoritmos respecto al tiempo

	Tiempo en 5 cifras	Tiempo en 10 cifras	Tiempo en 15 cifras	Tiempo en 20 cifras
alg 1	0.001	0.001	0.001	0.001
alg 2	0.001	0.0005	0.00075	0.00125
alg 3	0.00075	0.001	0.00075	0.001
alg 4	0.001	0.00175	0.00125	0.00075
alg 5	0.0095	0.04375	0.035	0.051
alg 6	0.02475	0.0985	0.123	0.297
alg 7	0.0115	0.032	0.03225	0.0415

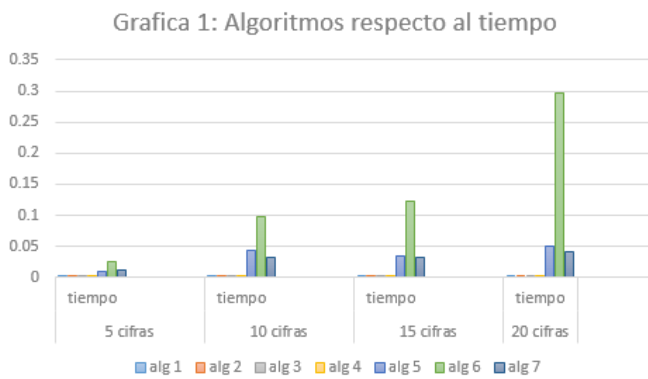

Tabla 27. Grafica 1: Graficos2: Algoritmos respecto al numero de vueltas

	Vueltas 5 cifras	Vueltas 10 cifras	Vueltas 15 cifras	Vueltas 20 cifras
alg 1	9.25	20	29.25	38.75
alg 2	7	14.5	20.75	27
alg 3	10.25	21	30.25	39.75
alg 4	26	58	84.25	113.5
alg 5	19	44.75	64.5	86.75
alg 6	23	90.5	108.75	327.5
alg 7	9.25	20	29.25	38.75

4 CONCLUSIONES

En las graficas 1 y 2 podemos observar que los primeros tres algoritmos son más eficientes, tanto en tiempo como en número de vueltas, pero sabemos también que tienen un costo computacional más elevado por la presencia de una división por vuelta como mínimo sin embargo personalmente considero el algoritmo 4, BinaryGcd, más eficiente, pues trabaja en binario, y con las mejoras realizadas para no hacer divisiones que son muy costosas computacionalmente, que fueron reemplazadas por corrimientos, este algoritmo se ejecuta en un tiempo similar sin importar el tamaño de los números evaluados, sin embargo en número de vueltas, puede ser más alto en números con tamaño pequeño, pero se mantiene para números más grandes, sin embargo este número de vueltas no necesariamente representa una desventaja, pues el costo computacional de cada vuelta es bajo al no realizar operaciones como la división. El algoritmo 5 binarygcd iterativo puede ser menos eficiente, pues realiza menos vueltas, pero más operaciones realizadas por vuelta, pese a evitar las divisiones re realizan al menos dos corrimientos dos and, dos asignaciones y dos swap o cambio de valor, lo cual lo haría menos eficiente que el Binarygcd recursivo. Por otra parte el algoritmo 6 es el menos eficiente, pues tanto en tiempo como vueltas es más grande, pues la forma iterativa da muchas vueltas por los bucles while, sin embargo computacionalmente también es muy bueno, pues sus operaciones son a bajo nivel y no requiere de gran costo computacional.