

Licenciatura en ciencia de la computación



ALGORITMO EUCLIDEANO

Matemática Computacional

Profesor:
Nicolas Thériault

Autor:
Sergio Salinas
Danilo Abellá

Introducción

Informe sobre las diferentes implementaciones del Algoritmo Euclidiano.

En la implementación se usó la librería GNU MP 6.1.1 en C++.

Para crear los gráficos se crearon dos números al azar a y b , además al número a se le sumó el número 2^n , donde n es la cantidad de bits a calcular, de esta forma, a siempre cumplía con la cantidad de bits que se pedía.

Los gráficos fueron creados usando un script en R.

1 Algoritmo Implementado

A continuación se mostrará que tan eficientes fueron los distintos tipos de algoritmos.

1.1 Algoritmo Euclidiano Simple

Este algoritmo demostró ser altamente ineficiente dado que cuando se trabaja con números muy grandes el tiempo requerido es muy alto, a pesar de que su lógica sea muy simple. Tiempos de espera para los siguientes gcd:

gcd(291,252) - 0.0002230 segundos
gcd(16261,85652) - 0.0002450 segundos
gcd(897279761,914407221) - 0.0009650 segundos
gcd(16534528044,8332745927) - 0.0008580 segundos
gcd(43263441545690516,43312793054108111) - 0.0068580 segundos

1.2 Algoritmo Euclidiano

Este algoritmo resultó ser la mejor opción en cuanto a complejidad y eficiencia ya que no demostró una lógica muy complicada ni un tiempo de espera demasiado alto, aunque no halla sido de los que menos tiempo de ejecución requirió.

Tiempos de espera para los siguientes gcd:

gcd(291,252) - 0.0000610 segundos
gcd(16261,85652) - 0.0000780 segundos
gcd(897279761,914407221) - 0.0000780 segundos
gcd(16534528044,8332745927) - 0.0000920 segundos
gcd(43263441545690516,43312793054108111) - 0.0000960 segundos
gcd(23356764234689876532233456788876543234567,

6277101735386680763835789423207666416083908700390324961279) - 0.0001140 segundos
gcd(2123010620889223608977186369097185643295866022
04480027488784019561937182491149755503041950,
68185362149486650485123987197513787717187652289362
47143232679847194284772046122091468887714) - 0.0002170 segundos

1.3 Algoritmo Euclidiano Binario

Este algoritmo estuvo de entre los más eficientes ya que a pesar de su complejidad fue uno de los más rápidos en ejecutarse.

Tiempos de espera para los siguientes gcd:

gcd(291,252) - 0.0000510 segundos
gcd(16261,85652) - 0.0000580 segundos
gcd(897279761,914407221) - 0.0000630 segundos
gcd(16534528044,8332745927) - 0.0000610 segundos
gcd(43263441545690516,43312793054108111) - 0.0000780 segundos
gcd(23356764234689876532233456788876543234567,
6277101735386680763835789423207666416083908700390324961279) - 0.0000730 segundos
gcd(2123010620889223608977186369097185643295866022
04480027488784019561937182491149755503041950,
68185362149486650485123987197513787717187652289362
47143232679847194284772046122091468887714) - 0.0001170 segundos

1.4 Algoritmo Euclidiano Extendido

Este sin duda resultó ser el algoritmo con tiempo de espera más corto haciendolo el más eficiente de todos los evaluados anteriormente. Tiempos de espera para los siguientes gcd:

gcd(291,252) - 0.0000350 segundos
gcd(16261,85652) - 0.0000340 segundos
gcd(897279761,914407221) - 0.0000360 segundos
gcd(16534528044,8332745927) - 0.0000400 segundos
gcd(43263441545690516,43312793054108111) - 0.0000410 segundos
gcd(23356764234689876532233456788876543234567,
6277101735386680763835789423207666416083908700390324961279) - 0.0000700 segundos
gcd(2123010620889223608977186369097185643295866022
04480027488784019561937182491149755503041950,
68185362149486650485123987197513787717187652289362
47143232679847194284772046122091468887714) - 0.0001410 segundos

1.5 Algoritmo Euclidiano Extendido Binario

Pese que con números de pocas unidades el programa tiene un tiempo de ejecución a la par con el resto de algoritmos, cuando se utilizan números con muchas cifras (> 10) ya el tiempo de espera se dispara quedando bastante atrás con respecto a sus otras opciones.

Tiempos de espera para los siguientes gcd:

`gcd(291,252)` - 0.0000390 segundos

`gcd(16261,85652)` - 0.0000420 segundos

`gcd(897279761,914407221)` - 0.0000570 segundos

`gcd(16534528044,8332745927)` - 0.0000620 segundos

`gcd(43263441545690516,43312793054108111)` - 0.0000910 segundos

`gcd(23356764234689876532233456788876543234567,`

`6277101735386680763835789423207666416083908700390324961279)` - 0.0002210 segundos

`gcd(2123010620889223608977186369097185643295866022`

`04480027488784019561937182491149755503041950,`

`68185362149486650485123987197513787717187652289362`

`47143232679847194284772046122091468887714)` - 0.0004830 segundos

2 Formulación experimentos

3 Información de Hardware y Software

3.1 Notebook - Danilo Abellá

3.1.1 Software

- SO: Xubuntu 16.04.1 LTS
- GMP Library
- Mousepad 0.4.0

3.1.2 Hardware

- AMD Turion(tm) X2 Dual-Core Mobile RM-72 2.10GHz
- Memoria (RAM): 4,00 GB(3,75 GB utilizable)
- Adaptador de pantalla: ATI Raedon HD 3200 Graphics

3.2 Notebook - Sergio Salinas

3.2.1 Software

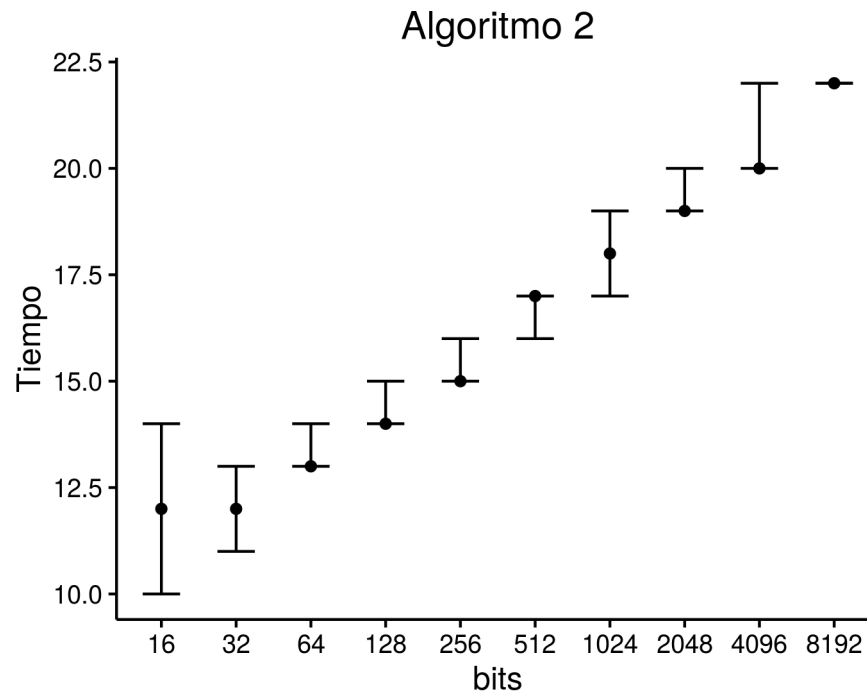
- SO: ubuntu Gnome 16.04 LTS
- Compilador: gcc version 5.4.0 20160609
- Editor de text: Atom

3.2.2 Hardware

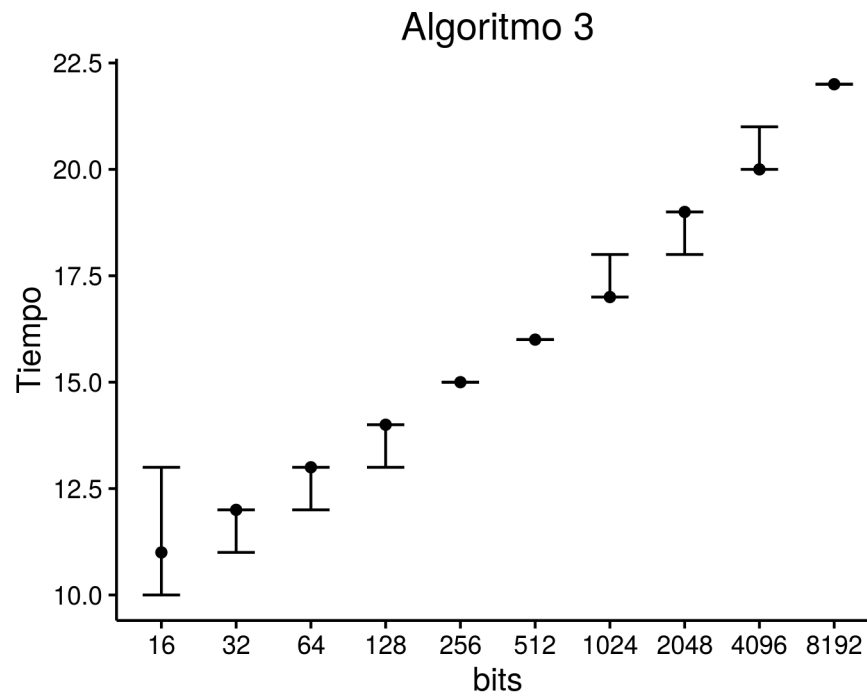
- Procesador: Intel Core i7-6500U CPU 2.50GHz x 4
- Video: Intel HD Graphics 520 (Skylake GT2)

4 Curvas de desempeño de resultados

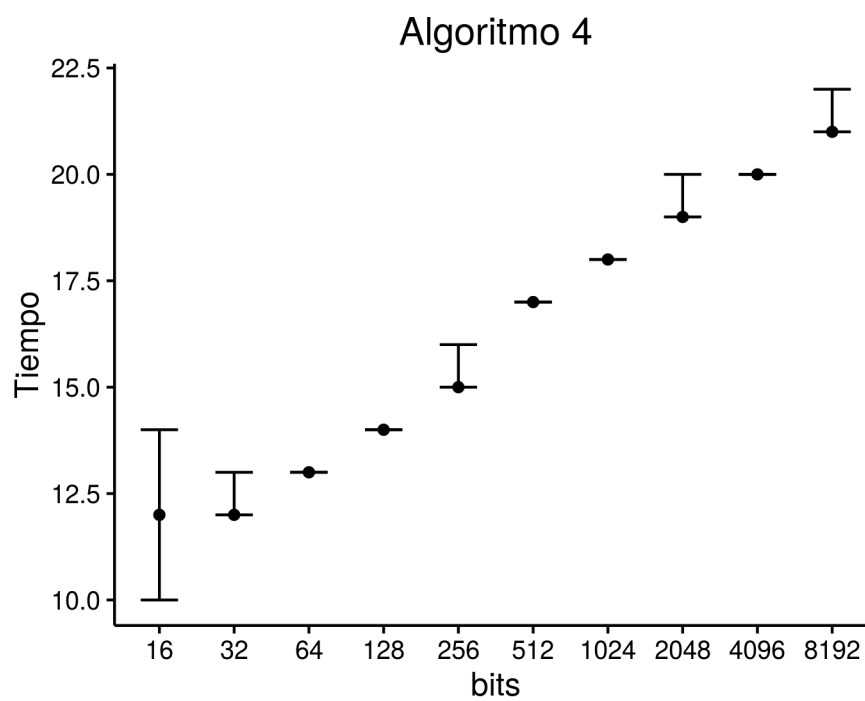
4.1 Algoritmo 2



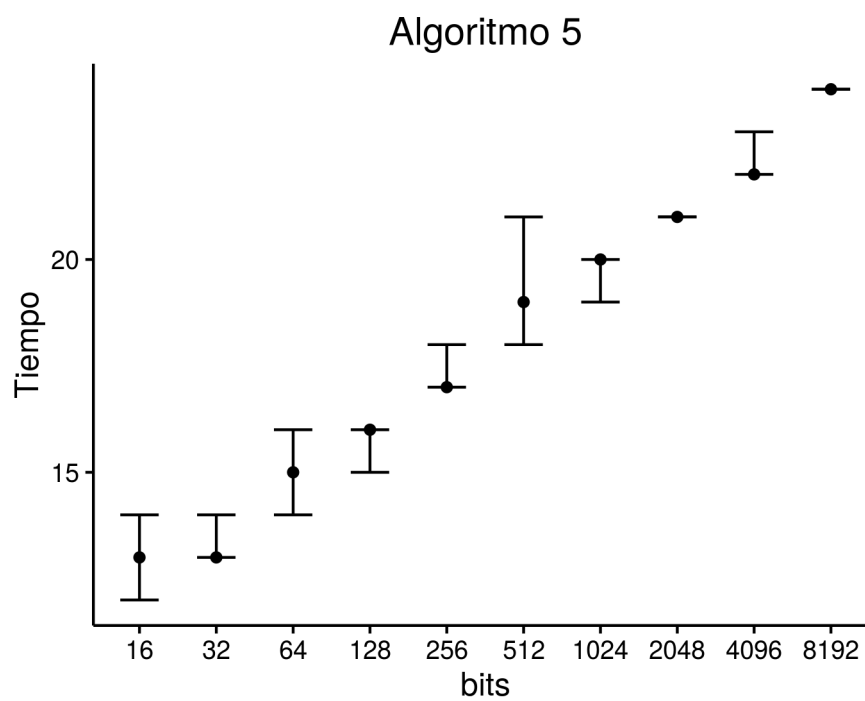
4.2 Algoritmo 3



4.3 Algoritmo 4



4.4 Algoritmo 5



5 Conclusiones