Licenciatura en ciencia de la computación

Algoritmo Euclideano

Matemática Computacional

Profesor:
Nicolas Thériault

Autor:
Sergio Salinas
Danilo Abellá

Introducción

Informe sobre las diferentes implentaciones del Algoritmo Euclidiano.

En la implemetanción se uso 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 sumo el número 2^n, donde n es la cantidad de bits a calcular, de esta forma, a siempre cumplia con la contidad de bits que se pedia.

Los gráficos fueron creado usando un script en R.

1 Formulación experimentos

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(233567642346898765322334567888876543234567,
627710173538668076383578942320766641608390870039032496127 9) - 0.0001140 segundos
gcd(212301062088922360897718636909718564329586602 2
044800274887840195619371824911497555030419 50,
681853621494866504851239871975137877171876522893 62
471432326798471942847720461220914688877 14) - 0.0002170 segundos

1.3 Algoritmo Euclidiano Binario

Este algoritmo estubo 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(233567642346898765322334567888876543234567,
627710173538668076383578942320766641608390870039032496127 9) - 0.0000730 segundos
gcd(212301062088922360897718636909718564329586602 2
044800274887840195619371824911497555030419 50,
681853621494866504851239871975137877171876522893 62
471432326798471942847720461220914688877 14) - 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(233567642346898765322334567888876543234567,
627710173538668076383578942320766641608390870039032496127 9) - 0.0000700 segundos
gcd(212301062088922360897718636909718564329586602 2
044800274887840195619371824911497555030419 50,
681853621494866504851239871975137877171876522893 62
471432326798471942847720461220914688877 14) - 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(233567642346898765322334567888876543234567,
627710173538668076383578942320766641608390870039032496127 9) - 0.0002210 segundos
gcd(212301062088922360897718636909718564329586602 2
044800274887840195619371824911497555030419 50,
681853621494866504851239871975137877171876522893 62
471432326798471942847720461220914688877 14) - 0.0004830 segundos

2 Información de Hardware y Software

2.1 Notebook - Danilo Abellá

2.1.1 Software

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

2.1.2 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

2.2 Notebook - Sergio Salinas

2.2.1 Software

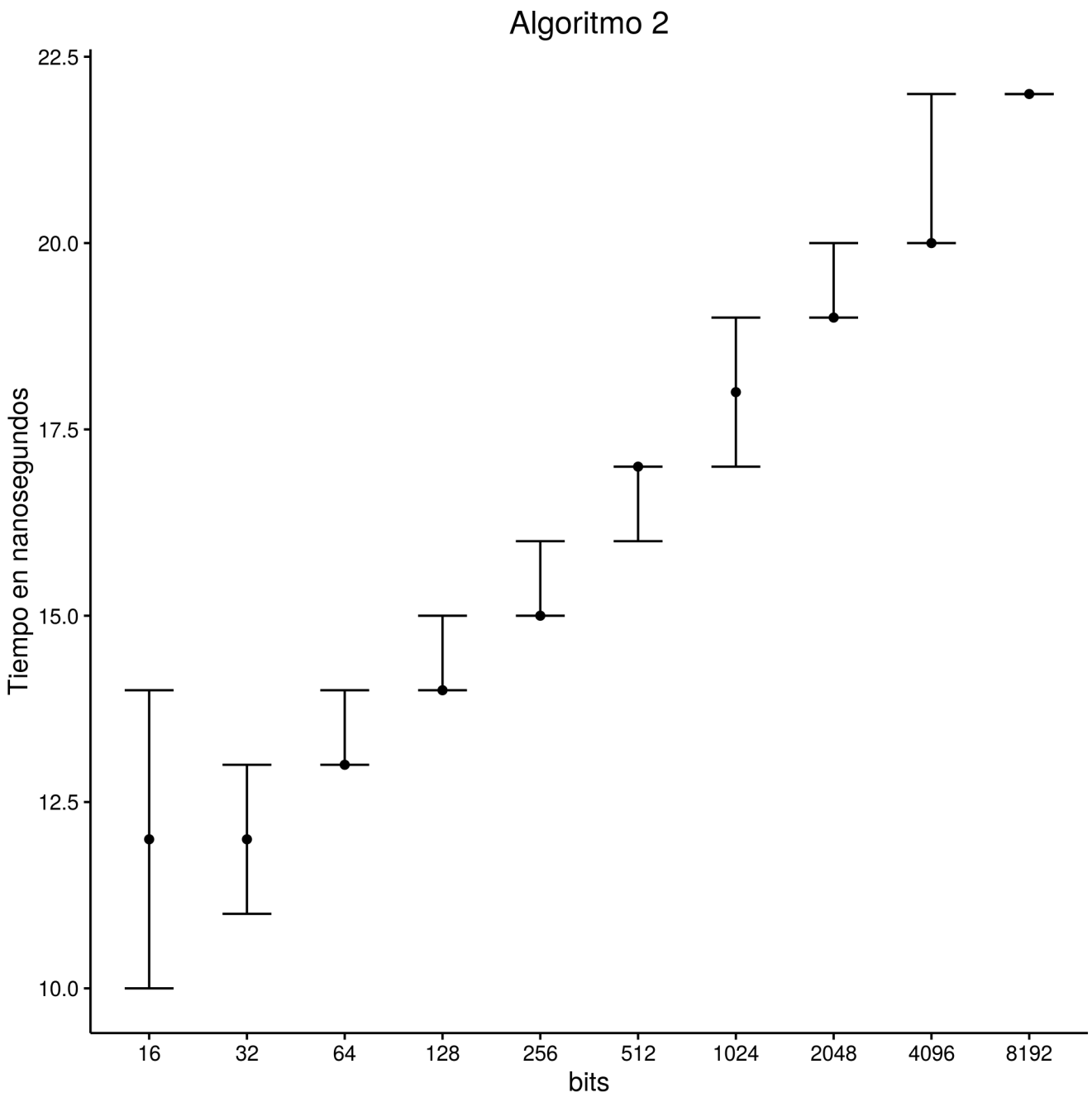
- SO: ubuntu Gnome 16.04 LTS
- Compilador: gcc version 5.4.0 20160609
- Editor de text: Atom

2.2.2 Hardware

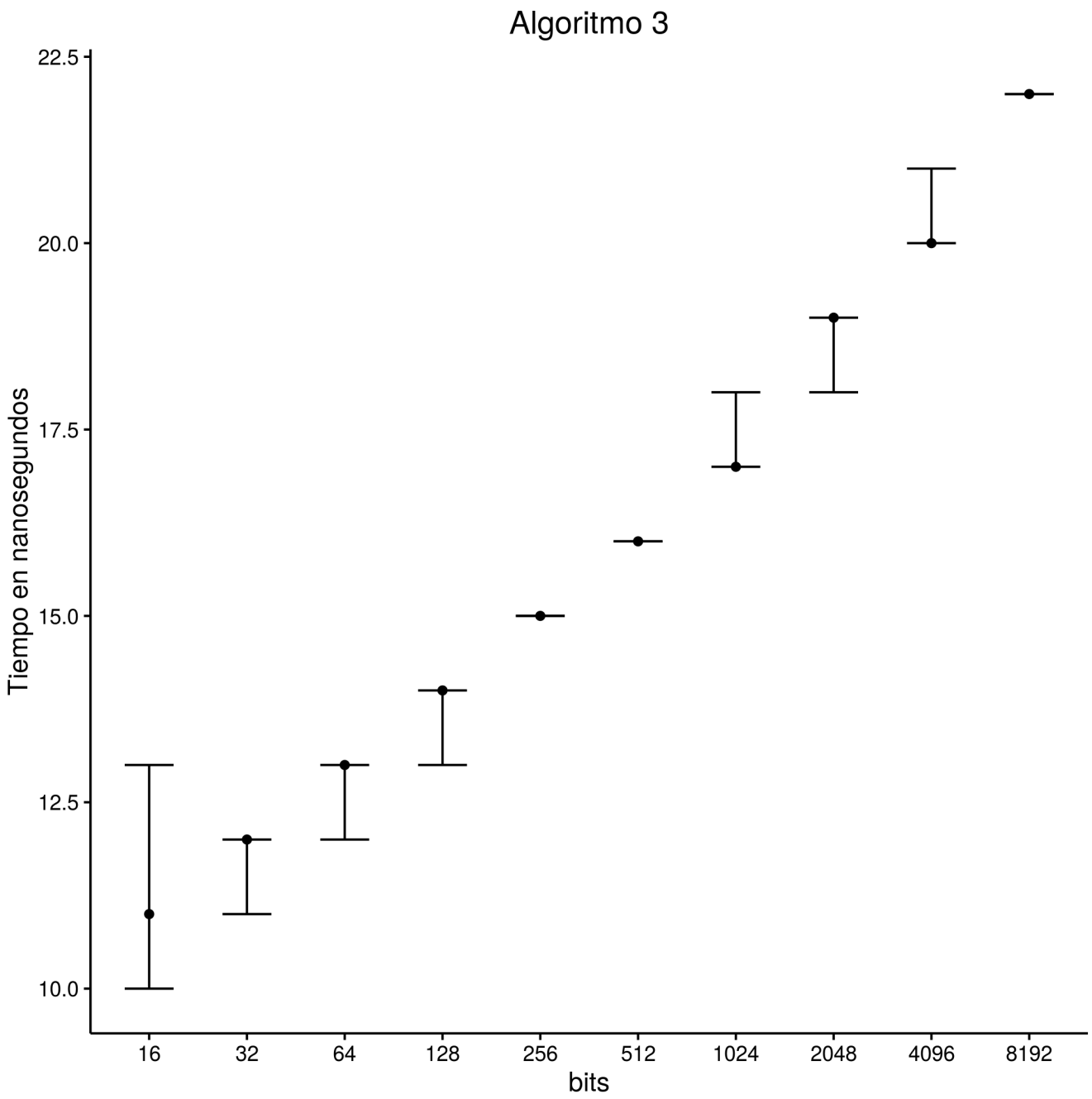
- Procesador: Intel Core i7-6500U CPU 2.50GHz x 4
- Video: Intel HD Graphics 520 (Skylake GT2)

3 Curvas de desempeño de resultados

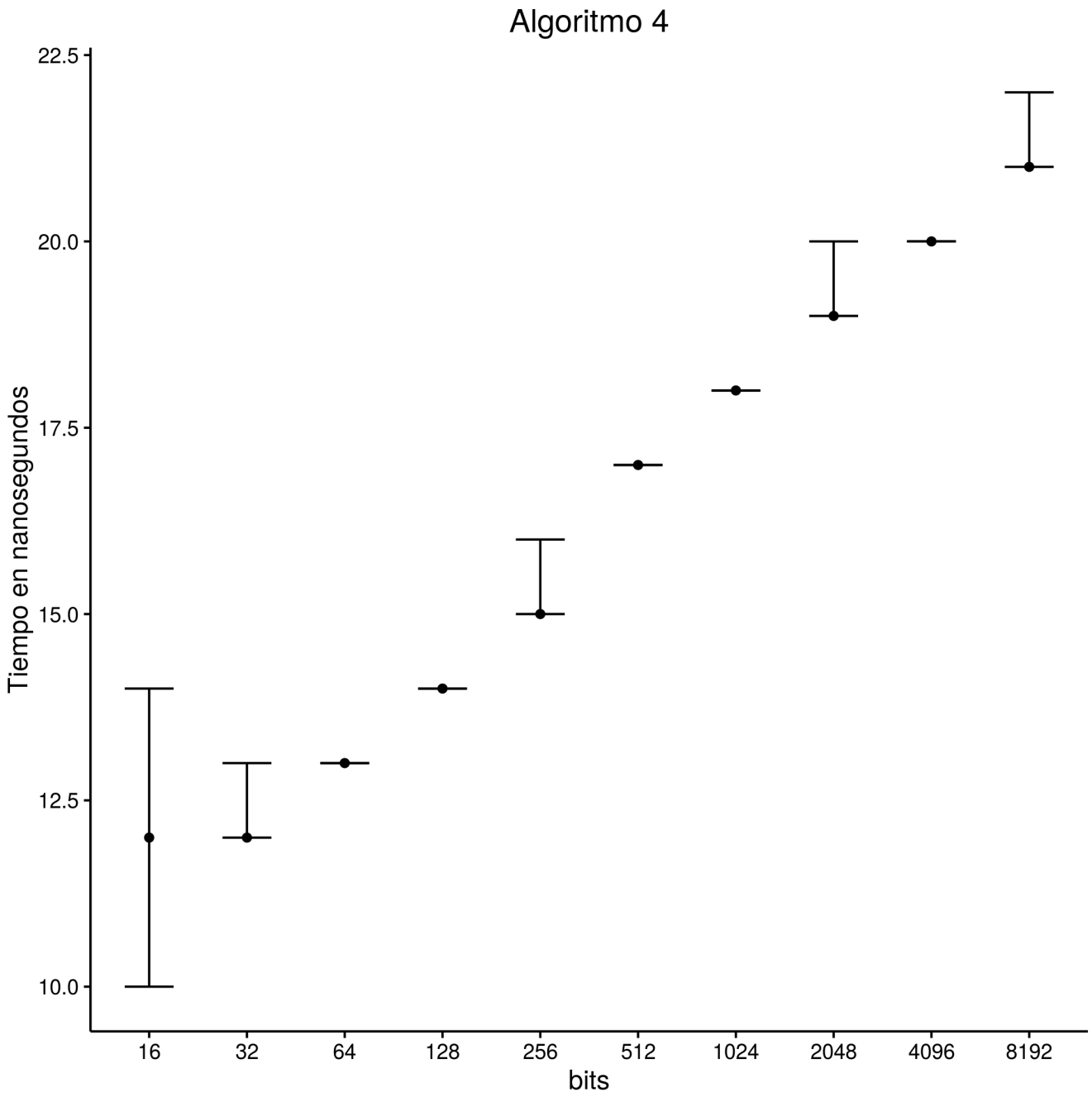
3.1 Algoritmo 2



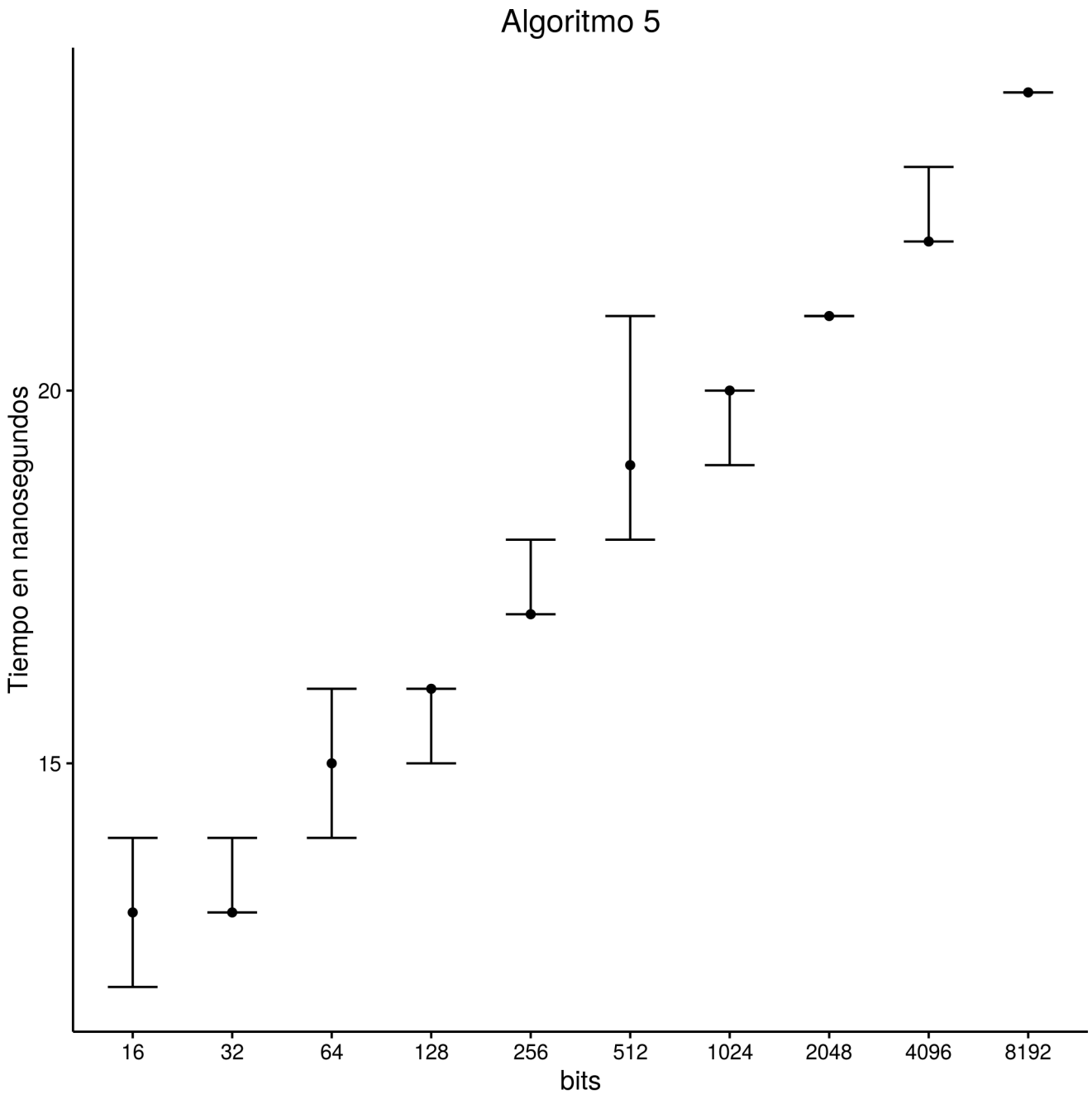
3.2 Algoritmo 3



3.3 Algoritmo 4



3.4 Algoritmo 5



4 Conclusiones

Se puede concluir que:

- El algoritmo 1 es el más ineficiente cuando se trata de calcular el gcd(a,b), tanto así que se dejo ejecuntando por tres días intentando calcular los datos suficientes para crear un gráfico y no se logro.
- El algoritmo 2 y 3 presentan un comportamiento similar, pero el dos tiene a tener un mayor rango de diferencia entre el promedio y los minimo y máximo.
- El algoritmo 4 es el más eficiente de todos, los extremos minimo y máximo están cercanos al promedio y es el que tiene menos coste de tiempo.
- El algoritmo 5, después del 1, es el que tiene mayor coste de tiempo.