Universidad de Talca

Facultad de Ingeniería

Departamento de Ciencias de la Computación

Informe Tarea 1

Numeros primos y conjeturas

Fecha: Viernes 20 de Abril de 2012
Autor: Erik Andres Regla Torres
e-mail: eregla09@alumnos.utalca.cl

1. Introducción

Breve descripción del problema y de su solución.

2. Análisis del problema

Se expone en detalle el problema, los supuestos que se van a ocupar, las situaciones de borde y la metodología para abordar el problema.

3. Solución del problema:

3.1. Algoritmo de solución

3.1.1. Generacion de numeros primos

Dentro de los requerimientos de la tarea solicitada, se hace evidente el desarrollo de algun sistema para generar numeros primos. Llamese numero primo a todo numero que solo es divisible por si mismo y por el numero uno, la manera obia de verificar esto, es comprobando que el numero en cuestion no sea divisible por ningun numero primo menor que el. Tambien de la conjetura anterior se desprende el que todo numero que no es primo, es "compuesto", en otras palabras que es resultado la multiplicacion de un primo por otro.

Considerando estos datos, la solucion obia para poder generar un numero primo, es tener una tabla de los numeros que uno lleva actualmente considerados como primos, y para cada numero n

que este en en el conjunto de pruebas, es primo si ningun elemento del conjunto solucion lo divide con modulo 0.
Sinembargo, esta solucion resulto no ser la mas optima, ya que esto implica que se deben hacer n comparaciones para cada elemento, por ende, obtendria un tiempo de ejecucion de n^2. Si bien para numeros menores a 10^3 no es tan notorio, ya desde 10^5 el cambio es significativo.

Despues de un tiempo de investigacion acerca de formas de generar de numeros primos, existe un sistema bastante curioso para poder generarlos. La idea principal del algoritmo, es generar un arreglo v, en donde la unica informacion que existe, es si el numero n es primo o no. Para eso, primero es generado una tabla donde todos los campos de esta estan marcados como $true$, indicando que es un numero primo. Luego, para cada elemento que no este tachado como falso, se le marcan todos los multiplos disponibles. Si el numero esta tachado, entonces se continua. Termina cuando se llega al final del arreglo, el cual ocurre en $sqrt(n)$. Todos los numeros que no posean marca, son primos.

```
void cribaDeErastotenes(bool[] v, int n){
. para i=0... n-1 {
. v[n] <- false
. }
.
. para(i=0... n-1){
. . v[n] <- false
. . si v[n] = true
. . descartarMultiplos()
. }
}
```

Este algoritmo es mas conocido como la “Criba de Erastótenes”.

3.1.2. Diseño principal

La primera conjetura par de Goldbach, indica que para cada numero par mayor de dos, se puede escribir como la suma de dos primos. Estos dos primos pueden repetirse en la suma, pero la conjetura estrictamente parte desde los pares mayores de dos, ya que los que estan antes, solo pueden ser escritos usando el mismo numero. Dado que el numero 1 no es un numero primo -de hecho, no se sabe su naturaleza, o almenos yo no la puedo determinar- este no es considerado. Por ende esa es la restriccion.

Ahora, por motivos de requerimientos del trabajo involucrado, esta restriccion fue levantada, de modo, que el numero pueda hacerse sobre una suma de si mismo, de modo que si el numero par es primo -en este caso el 2- pueda asi ser expresado como la suma de 2 y nada mas.

Una de las opciones para poder obtener un resultado satisfactorio, al igual que el caso anterior, fue imaginar una comparacion de todos contra todos, y si las comparaciones no arrojaban resultados satisfactorios, entonces, la conjetura era falsa para ese numero. Bueno, dado que estamos hablando de solo dos combinaciones, para rangos pequeños como 10^4 la ejecucion es medianamente lenta.

Sin embargo, para numeros mayores la ejecucion es realmente lenta. Entonces, surge la nececidad de refinar el algoritmo aplicando ciertas reglas para poder reducir el espacio de soluciones. Por ejemplo, tomando en cuenta que usamos dos numeros para partir la busqueda i y j, i recorriendo desde la posicion 0 hasta j, y j recorriendo desde el primo directamente igual o menor a j.

Se podria imaginar que en este caso las combinatorias podrian ser bastante tediosas, ya que pide comprobar todos los numeros disponibles de i antes de cambiar el valor de j. Por ende es nesesaria algunas refinacion.

3.1.3. Refinaciones

La primera refinacion a aplicar esta basada en el principio que si la suma de i y j es mayor a n (el numero buscado), entonces no tiene caso seguir incrementando j ya que siempre sea un numero superior.
La segunda refinacion al algoritmo esta basada en la mencionada anteriormente. Dado que es inutil siempre partir buscando desde el numero mas pequeño, es mas factible, comenzar a buscar desde el primo mas cercano a la diferencia de i y n, y desde ahi comenzar a ejecutar una regresion, ya que hay mas posibilidades que el primo mas cercano que haga la suma exacta sea el que es directamente el menor que la diferencia entre i y n.
Tambien se sabe que si la combinatoria es llevada a cabo solo por dos numeros, ninguno de ellos puede ser menor que $n/2$, dada esta condicion, tambien los dos no pueden ser al mismo tiempo mayores que $n/2$, dado que entregaria un numero mayor siempre. Por ende, una buena opcion es interrumpir el ciclado cuando alguna de las condiciones expuestas anteriormente se da.
Otro caso partcular que se puede dar es que justo $j = n$, entonces, en ese caso, el ciclado retorna que ese numero cumple la conjetura inmediatamente. Esto si bien, rompe con la conjetura de Goldbach, en la tarea no fue especificado, de modo que es totalmente valido hacer ese cambio.

3.1.4. Implementacion sobre las conjeturas

El algoritmo anterior es totalmente valido para cuando es nesesario dos niveles de conbinatoria. Para mas niveles, se implementa de forma recursiva, reemplazando n por j, y asi susesivamente. Pero el hacer una implementacion recursiva, pone en riesgo que la pila de operaciones se vuelque por accidente debido a que podria haber un numero extremadamente grande, que nececite demasiadas recursiones. Por ende, esa implementacion fue migrada a ciclos.

La estructura básica de cada ciclo es:

```
elejir i y j prometedores
mientras i > (n-1/n)i/n
. si i = n, retorna verdadero
. si i > n, rompe ciclo
. mientras(j < (n-2/2)j/n)
. . -implementarNuevamente-
. i <- el primo anterior a i
.
```

Este tipo de estructura asegura dos cosas. Que las soluciones no den falsos positivos ni negativos, ya que la condicion de exito, solo se da cuando la combinacion da n. Tambien asegura que las refinaciones anteriormente tratadas se cumplan.

3.2. Diagrama de Estados

Pendiente

3.3. Diseño

Pendiente

3.4. Implementación

Algoritmo 1 principal

Entrada: Parametros de entrada validos, con $n \in [1, +\text{ínf}]$ y $conjetura = 1, 2, 3$
Salida: 0 si la ejecucion fue exitosa, otro valor en caso de haber fallado.

```
1: prosesarArgumentos()
2: v ← arreglo de tamaño n
3: generarPrimos(v)
4: numeros ← 0
5: para i = 0..n hacer
6:    si V[i] = cierto  entonces
7:       numeros ← numeros + 1
8:    fin si
9: fin para
10: t ← arreglo de tamaño numeros
11: para i = 0, j = 0..n hacer
12:    si v[i] = cierto  entonces
13:       t[j] ← i
14:    fin si
15: fin para
16: switch (conjetura)
17: caso 1:
18: conjetura1(v,t,numeros,n)
19: caso 2:
20: conjetura2(v,t,numeros,n)
21: caso 3:
22: conjetura3(v,t,numeros,n)
23: fin switch
```

Algoritmo 2 conjetura$[1,2,3](v, size, n)$

Entrada: v, t son arreglos definidos, con v de tamaño n y t de tamaño $size$ y$size < n$.
Salida: **cierto** si la conjetura es valida para el universo de numeros, de lo contrario **falso**

```
1: cumplida ← cierto
2: para i = 3..n hacer
3:    si _conjetura[1,2,3](v, size, n) = falso  entonces
4:       cumplida ← falso
5:    fin si
6: fin para
7: si cumplida = falso  entonces
8:    imprimir  conjetura no cumplida para numeros...
9: fin si
10: imprimir  conjetura cumplida
```

Algoritmo 3 $\_conjetura1(v, size, n)$

Entrada: v,es un arreglo definido, con v de tamaño n y t de tamaño $size$ y $size < n$.
Salida: **cierto** si la conjetura es valida para el universo de numeros, de lo contrario **falso**

```
1: primo1 ← primoAnteriorIgual(v, n)
2: primo2 ← 0
3: mientras primo1 ≥ n/2 hacer
4:    si primo1 = n entonces
5:       devolver  cierto
6:    fin si
7:    primo2 ← primoSiguenteIgual(v, size, (n − primo1))
8:    mientras primo2 ≤ n/2 hacer
9:       si primo1 + primo2 = n entonces
10:         devolver  cierto
11:      fin si
12:      si primo1 + primo2 > n entonces
13:         break
14:      fin si
15:      primo2 ← primoSiguente(v, primo1, primo2)
16:   fin mientras
17:   primo1 ← primoAnterior(v, primo1)
18: fin mientras
19: devolver  falso
```

Algoritmo 4 $\_conjetura2(v, size, n)$

Entrada: v,es un arreglo definido, con v de tamaño n y t de tamaño $size$ y $size < n$.
Salida: **cierto** si la conjetura es valida para el universo de numeros, de lo contrario **falso**

```
1: primo1 ← primoAnteriorIgual(v, n)
2: primo2 ← 0
3: primo3 ← 0
4: mientras primo1 ≥ 2n/3 hacer
5:    si primo1 = n entonces
6:       devolver cierto
7:    fin si
8:    primo2 ← primoSiguenteIgual(v, size, (n − primo1))
9:    mientras primo2 ≤ 2 ∗ (n/3) hacer
10:      si primo1 + primo2 = n entonces
11:         devolver cierto
12:      fin si
13:      si primo1 + primo2 > n entonces
14:         break
15:         primo3 ← primoSiguenteIgual(v, size, (n − primo1 − primo2))
16:      fin si
17:      mientras primo3 ≤ n/3 hacer
18:         si primo1 + primo2 + primo3 = n entonces
19:            devolver cierto
20:         fin si
21:         si primo1 + primo2 + primo3 > n entonces
22:            break
23:         fin si
24:         primo3 ← primoSiguenteIgual(v, primo2, primo3)
25:      fin mientras
26:      primo2 ← primoSiguente(v, primo1, primo2)
27:   fin mientras
28:   primo1 ← primoAnterior(v, primo1)
29: fin mientras
30: devolver falso
```

Algoritmo 5 $\_conjetura3(v, size, n)$

Entrada: v,es un arreglo definido, con v de tamaño n y t de tamaño $size$ y $size < n$.
Salida: **cierto** si la conjetura es valida para el universo de numeros, de lo contrario **falso**

```
1: primo1 ← primoAnteriorIgual(v, n)
2: primo2 ← 0
3: primo3 ← 0
4: mientras primo1 ≥ 4n/5 hacer
5:   si primo1 = n entonces
6:     devolver cierto
7:   fin si
8:   primo2 ← primoSiguenteIgual(v, size, (n − primo1))
9:   mientras primo2 ≤ 4n/5 hacer
10:    si primo1 + primo2 = n entonces
11:      devolver cierto
12:    fin si
13:    si primo1 + primo2 > n y primo1 ≠ primo2 entonces
14:      break
15:      primo3 ← primoSiguenteIgual(v, size, (n − primo1 − primo2))
16:    fin si
17:    mientras primo3 ≤ 3n/5 hacer
18:      si primo1 + primo2 + primo3 = n entonces
19:        devolver cierto
20:      fin si
21:      si primo1 + primo2 + primo3 > n y primo1 ≠ primo2 ≠ primo3 entonces
22:        break
23:        primo4 ← primoSiguenteIgual(v, size, (n − primo1 − primo2 − primo3))
24:      fin si
25:      mientras primo4 ≤ 2n/5 hacer
26:        si primo1 + primo2 + primo3 + primo4 = n entonces
27:          devolver cierto
28:        fin si
29:        si primo1 + primo2 + primo3 + primo4 > n y primo1 ≠ primo2 ≠ primo3 ≠ primo4
           entonces
30:          break
31:          primo5 ← primoSiguenteIgual(v, size, (n − primo1 − primo2 − primo3 − primo4))
32:        fin si
33:        mientras primo5 ≤ n/5 hacer
34:          si primo1 + primo2 + primo3 + primo4 + primo5 = n entonces
35:            devolver cierto
36:          fin si
37:          si primo1 + primo2 + primo3 + primo4 + primo5 > n y primo1 ≠ primo2 ≠
             primo3 ≠ primo4 ≠ primo5 entonces
38:            break
39:          fin si
40:          primo5 ← primoSiguenteIgual(v, primo4, primo5)
41:        fin mientras
42:        primo4 ← primoSiguente(v, primo3, primo4)
43:      fin mientras
44:      primo3 ← primoSiguente(v, primo2, primo3)
45:    fin mientras
46:    primo2 ← primoSiguente(v, primo1, primo2)
```

3.5. Modo de uso

Dentro de el directorio src, existe un archivo llamado “run.sh” el cual automaticamente compila el programa y entrega un ejecutable llamado “tarea1”, si es que se esta dentro de un entorno linux.

A modo de requerimiento, solo esta el del sistema operativo, el cual debe ser alguna distribucion linux con compilador g++ $\geq$ 4.6.3

Respecto a los parametros, para asegurar la correcta ejecucion del programa se sugiere no incluir valores negativos o caracteres como parametros.

4. Pruebas, resultados y anexos

Se ejecutaron pruebas para todas las decadas solicitadas, desde 10^1 hasta 10^9, para las tres conjeturas. Dentro de las estadisticas de rendimiento, se pueden calsificar en

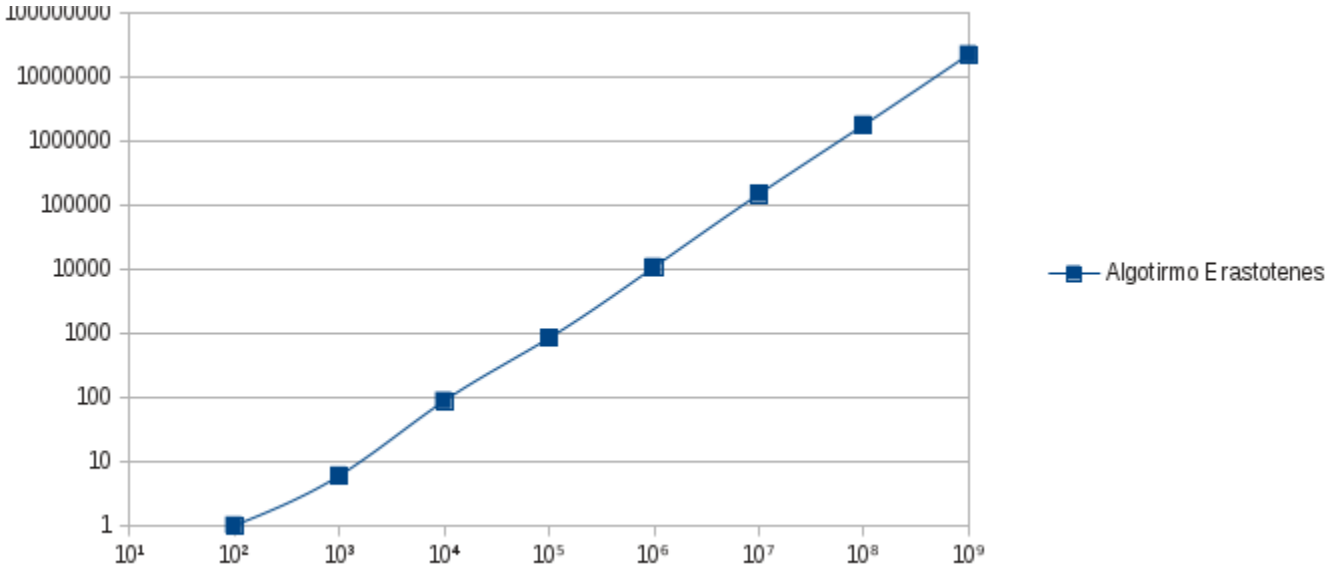

Figura 1: Rendimiento de la implementacion del algoritmo de Erastótenes

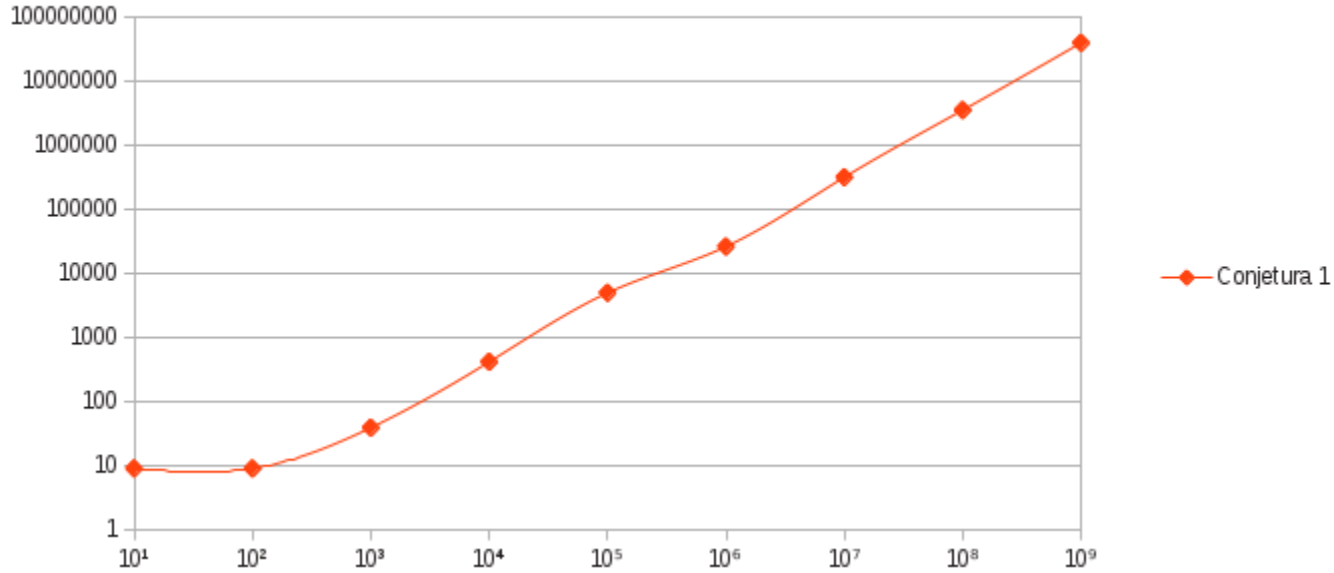

Figura 2: Rendimiento de la implementacion de la primera conjetura

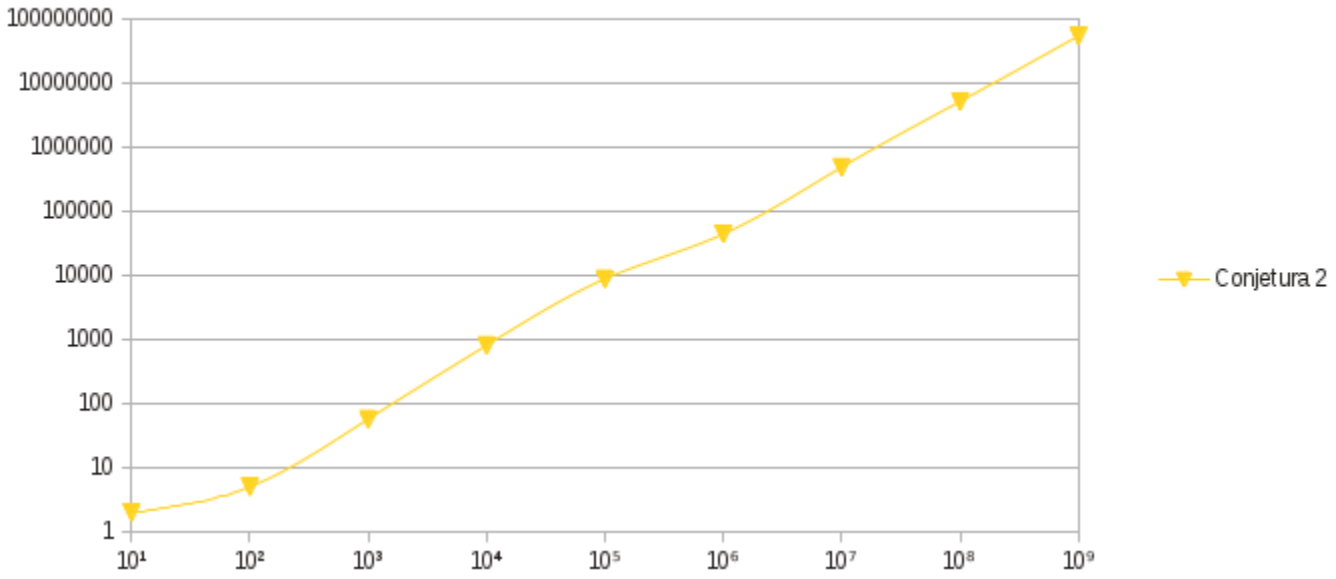

Figura 3: Rendimiento de la implementacion de la segunda conjetura

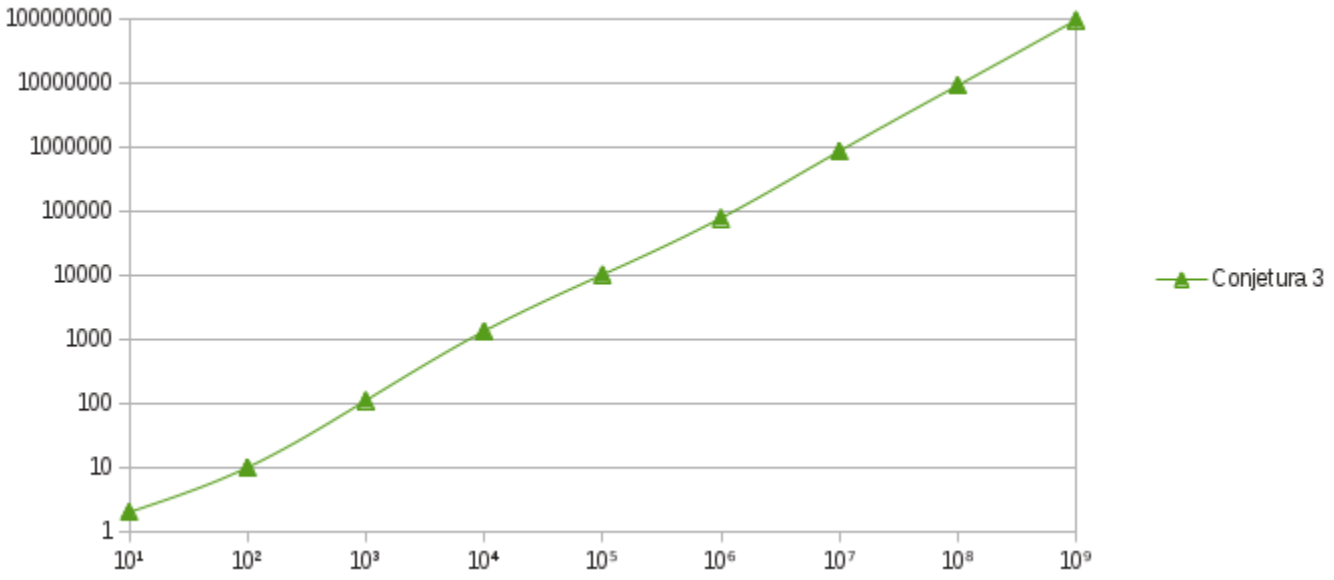

Figura 4: Rendimiento de la implementacion de la tercera conjetura

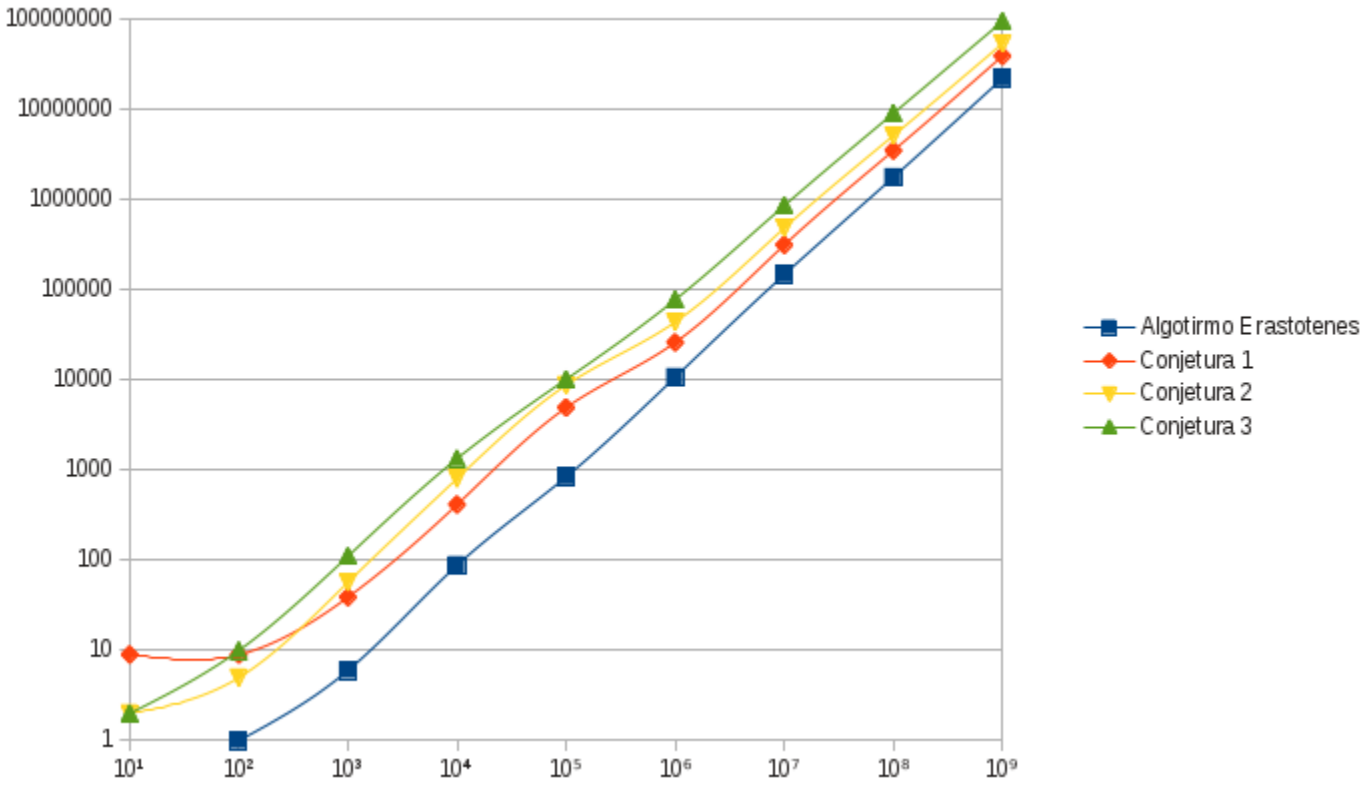

Figura 5: Interpolacion de rendimiento de los algoritmos descritos