CRIPTOGRAFÍA Y COMPUTACIÓN

GRADO EN INGENIERÍA INFORMÁTICA

Logaritmo Discreto y Factorización

PRÁCTICA 3

Autores

Roberto García Pérez

Vladislav Nikolov Vasilev

Víctor Alejandro Vargas Pérez

Fecha de Entrega

Granada, 14 de abril de 2020

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

CURSO 2019-2020

1. Introducción

En esta práctica se han implementado diferentes algoritmos para dos problemas: el logaritmo discreto y la factorización. La implementación de dichos algoritmos se puede ver en el fichero **p3.py**, que a su vez hace uso de **Utilidades.py**, fuente con funciones auxiliares aportadas para realizar esta práctica, así como otras realizadas en prácticas anteriores.

El fichero **p3.py** también contiene unas funciones usadas para obtener los tiempos de ejecución de los algoritmos, que se han usado para realizar el análisis de los tiempos que se presenta a continuación.

2. Problema Logaritmo Discreto

Se mide el tiempo de cada algoritmo para diferentes tamaños del problema, siendo este determinado por el número de bits de la entrada. Para cada tamaño, se genera un primo fuerte con el número de bits correspondiente (p), así como un elemento primitivo de este (a). Con esto garantizamos que existe el logaritmo discreto. A continuación, se generan tantos enteros aleatorios entre p y p-2 (b) como repeticiones se hayan prefijado, y se ejecuta el algoritmo con cada uno de ellos, midiendo el tiempo. De esta forma, se realiza finalmente una media de los tiempos, con lo que tendremos el tiempo del algoritmo con el tamaño indicado.

Concretamente, hemos comenzado a ejecutar todos los algoritmos con números primos (p) de 5 bits, incrementado el tamaño en 2 bits cada vez y terminando cuando en media el tiempo rondaba los 100 segundos. Estos son los resultados:

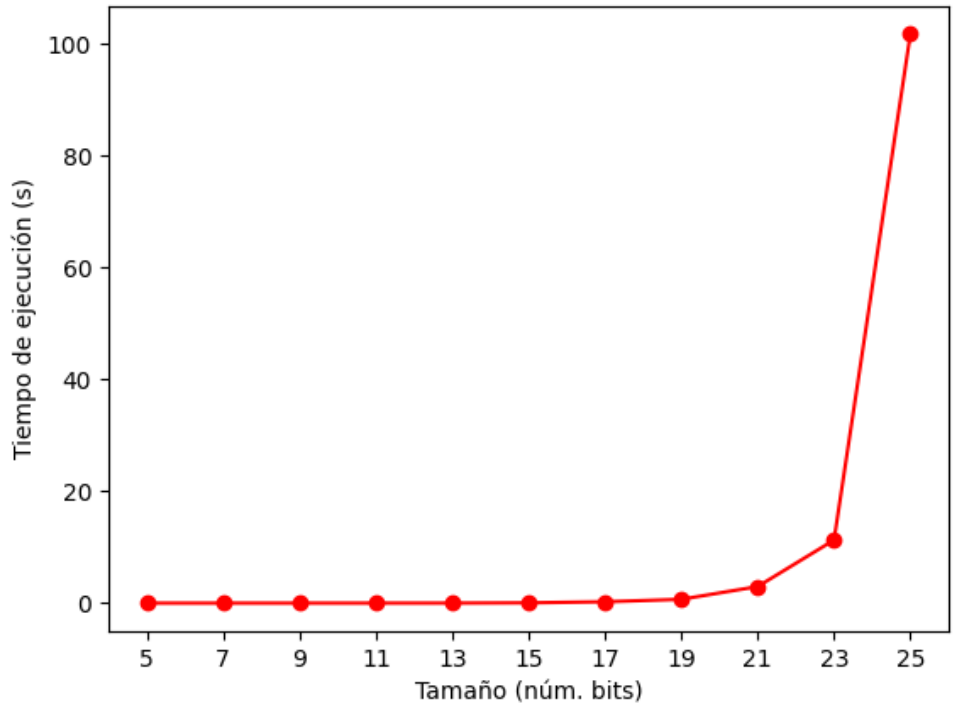

Figura 1: Logaritmo Discreto con **Fuerza Bruta**

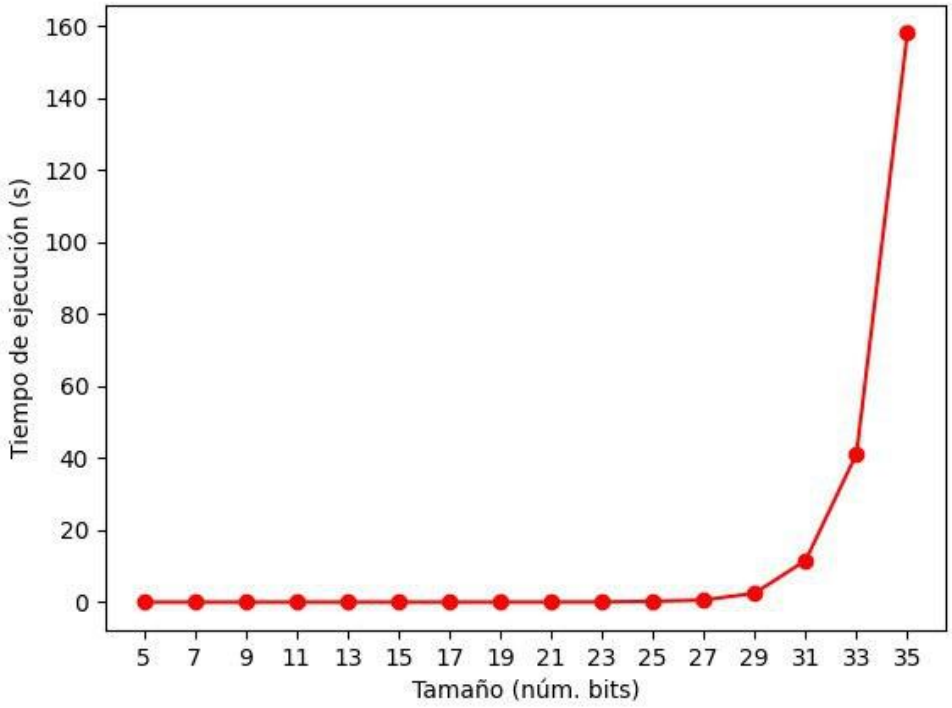

Figura 2: Logaritmo Discreto con **Paso enano - Paso gigante**

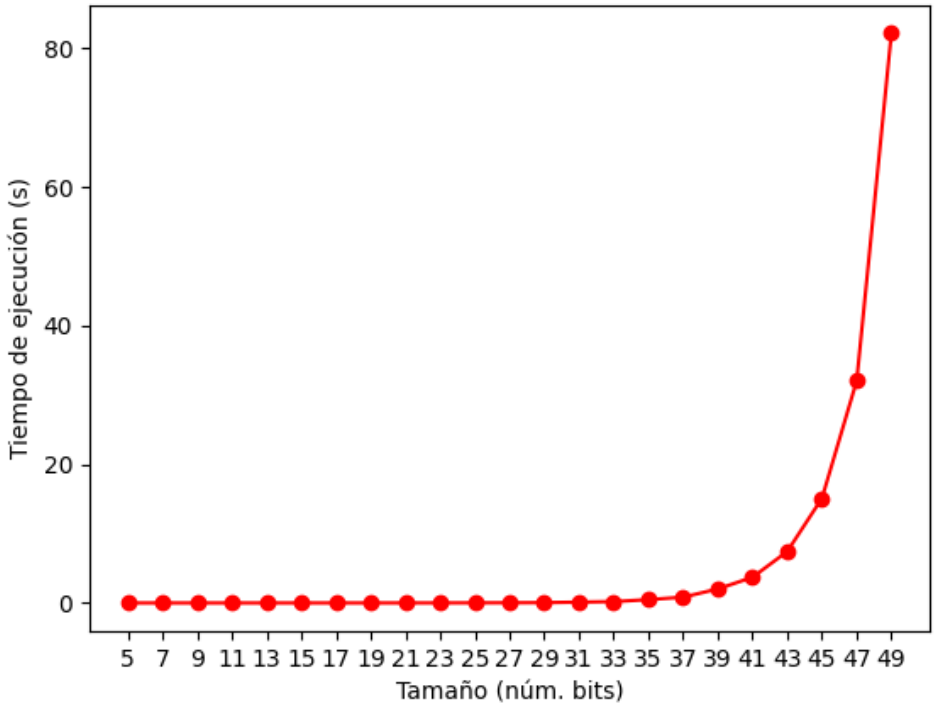

Figura 3: Logaritmo Discreto con ρ **de pollard**

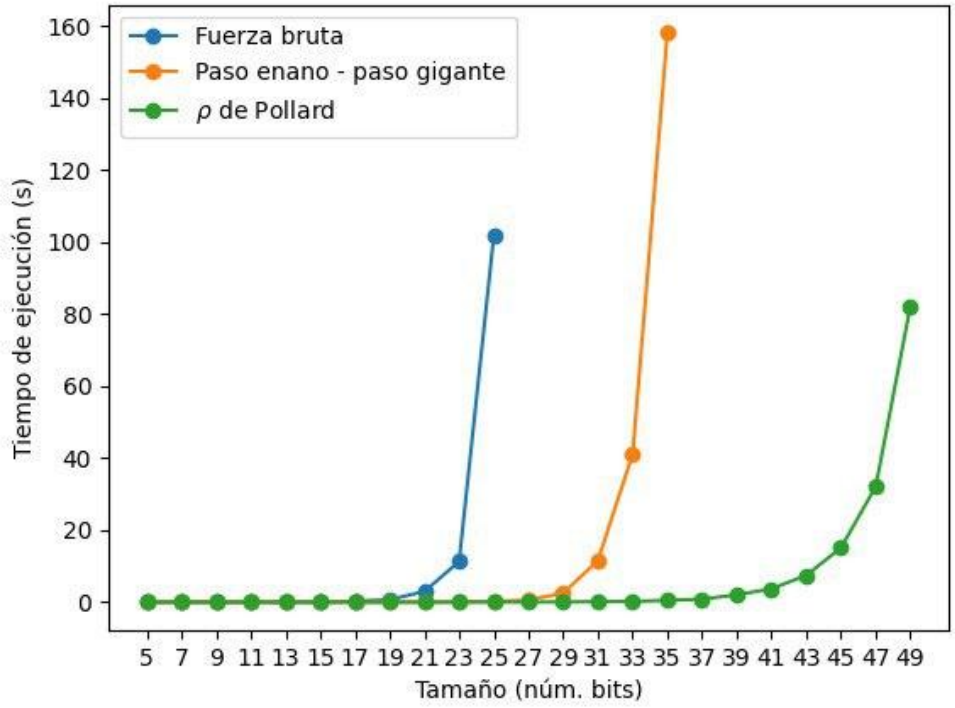

Figura 4: Logaritmo Discreto con los 3 algoritmos

Si observamos los tres algoritmos de forma conjunta, vemos que cuando se utilizan números relativamente pequeños de entre 5 y 17 bits, los resultados son más o menos parecidos. Sin embargo, a partir de este número de bits los resultados difieren. Los tiempos parecen tener un crecimiento exponencial en función del número de bits, aunque difieren en cuanto a la tasa de crecimiento.

Por ejemplo, en el caso del logaritmo por fuerza bruta vemos que los tiempos se disparan mucho antes que en los otro casos. En este caso, con 25 bits, el tiempo medio de ejecución supera levemente los 100 segundos. Al pasar de 23 bits a 25 parece que el tiempo medio de ejecución se ha visto multiplicado por 10 aproximadamente (de aproximadamente unos 10 segundos a unos 100). Por tanto, si seguimos ejecutando el algoritmo, con 27 bits puede que tarde aproximadamente 1000 segundos de media, con 29, unos 10000, y así sucesivamente.

Si ahora nos fijamos en el método de paso enano - paso gigante, vemos que puede calcular rápidamente el algoritmo discreto para números de hasta unos 29 bits. A partir de 31, los tiempos empiezan a incrementarse. Este crecimiento se parece al que teníamos anteriormente aunque con una tasa de crecimiento menor. Aquí parece que el tiempo se va multiplicando por 4 cada vez que se incrementa el número de bits, de forma que si por ejemplo para 31 bits tenemos un tiempo de ejecución medio de unos 10 segundos, para 33 bits el tiempo será de aproximadamente 40 segundos. En el caso de que por ejemplo quisiéramos calcular el logaritmo con un número p de 37 bits, probablemente el tiempo de ejecución sería de unos 640 segundos.

Por último tenemos el algoritmo ρ de Pollard, el cuál ha mostrado un mejor comportamiento que los dos anteriores, ya que ha permitido calcular el logaritmo de números de hasta 39 bits de forma relativamente rápida. A partir de los 41 bits, el algoritmo empieza a tardar más tiempo y empieza a tener un comportamiento que nos recuerda a los dos casos anteriores. Aquí, sin

embargo, parece que el tiempo se duplica al pasar de un número de bits al siguiente. De esta forma, si por ejemplo decidiésemos calcular el logaritmo discreto con un número p de 51 bits, el tiempo medio aproximado de ejecución sería de unos 160 segundos. Es decir, tardaría lo mismo que el de paso enano - paso gigante con 35 bits. Si decidiésemos utilizar 53 bits, el tiempo sería aproximadamente de unos 320 segundos, y así sucesivamente.

En resumen, el mejor algoritmo parece ser el ρ de Pollard, ya que es el que permite calcular el logaritmo para números más grandes en un menor tiempo. El segundo mejor es el de paso enano - paso gigante, mientras que el peor es, obviamente, el de fuerza bruta.

3. Problema Factorización

De nuevo, se mide el tiempo de cada algoritmo para diferentes tamaños, en función del número de bits del entero a factorizar. En esta ocasión, también se realizan varias repeticiones por tamaño, sacando el tiempo medio y máximo de estas, pero la diferencia está en la generación de cada número n a factorizar. De esta forma, para cada tamaño:

- Se genera un número n como el producto de números primos pequeños (de 137 en adelante), de tantos como sean necesarios para tener el número de bits oportuno.
- Se genera un segundo número n como el producto de dos primos fuertes de la mitad de bits del número a factorizar.
- El resto de número se generan como impares aleatorios no primos del número de bits correspondientes, sin más restricciones.

El motivo por el que se genera uno de los números como el producto de dos primos grandes es que así nos aseguramos de que la factorización sea complicada, pues solo habrá dos divisores y serán de gran tamaño. Por esta razón, es interesante obtener el máximo además de la media de los tiempos, pues el máximo estará ligado a esta versión más complicada.

Se realizan 10 repeticiones por tamaño (incluyendo los dos casos especiales), incrementándose este en un bit cada vez. Los resultados se muestran a continuación.

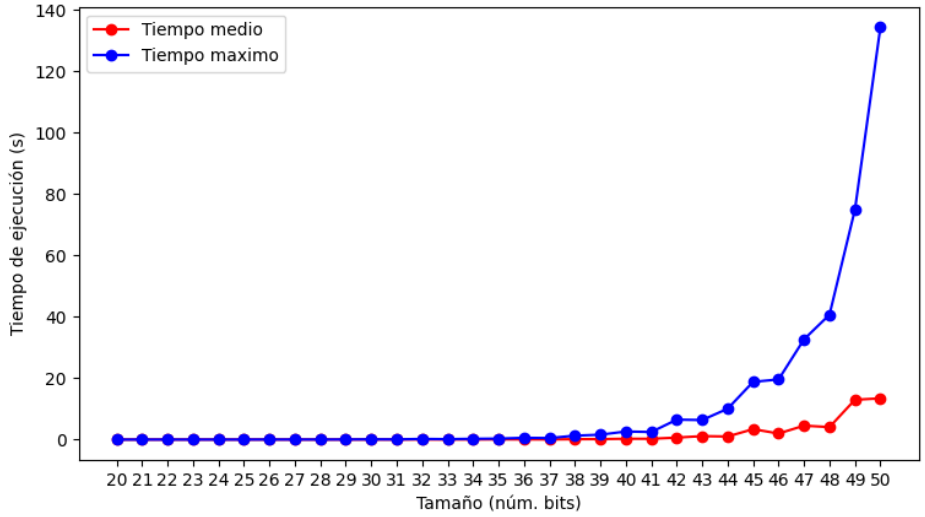

Figura 5: Factorización con **División por Tentativa**

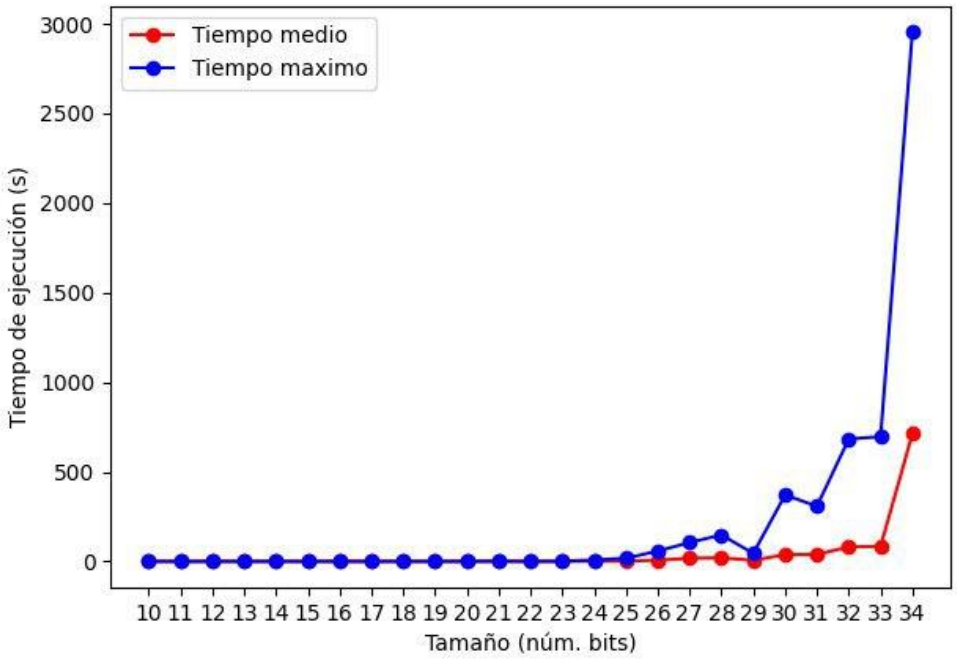

Figura 6: Factorización con **Método de Fermat**

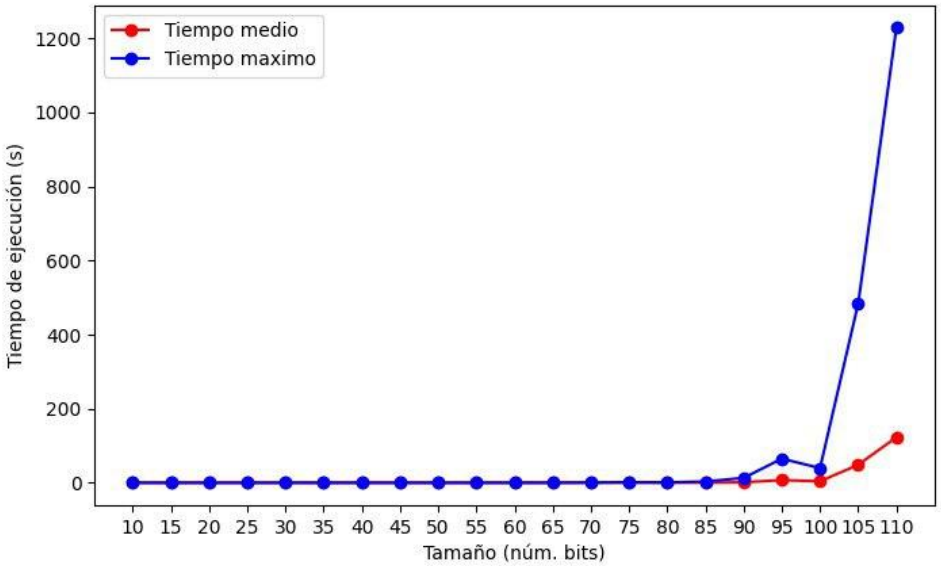

Figura 7: Factorización con ρ **de pollard**

En primer lugar, cabe destacar que, como era de esperar, los tiempos máximos son considerablemente superiores a los medios. Esta diferencia es más notoria cuanto mayor es el número, pues el máximo tiene un crecimiento más pronunciado. Esto constata la diferencia, para cualquiera de los algoritmos, de factorizar un número impar no primo cualquiera respecto a un número con únicamente dos divisores, y ambos de gran tamaño. En los 3 algoritmos parece que el crecimiento en tiempo es exponencial respecto al número de bits de la entrada, la diferencia se encuentra en la velocidad de dicho crecimiento.

El peor algoritmo con diferencia es el Método de Fermat, que con 34 bits ya tarda casi 1000 segundos en media y 3000 segundos como máximo. Le sigue la división por tentativa que en ese punto aún tiene valores cercanos al 0, pero que con 50 bits ya presenta un crecimiento notable, pasando a 140 segundos de máximo y casi 20 segundos de media.

Por su parte, ρ de pollard es el mejor de los algoritmos: no comienza un crecimiento significativo hasta los 100 bits, obteniendo con 110 bits menos de 200 segundos de media y algo más de 1200 segundos de máxima.