Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Proyecto de tecnológico

Estudio, implementación y análisis computacional del algoritmo de primalidad AKS

Trimestre 2018 Invierno

Katia Cecilia Chávez Rodríguez
Matrícula: 2123033387

Asesor:
Víctor Cuauhtemoc García Hernández
Doctor en Ciencias
Profesor asociado
Departamento de Ciencias Básicas

24 de abril de 2018

Declaratoria

Yo, Victor Cuahutémoc García Hernández, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dr. Víctor Cuauhtemoc García Hernández

Yo, Katia Cecilia Chávez Rodríguez, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Katia Cecilia Chávez Rodríguez

Resumen

Se presenta una implementación del algoritmo AKS en Python, sustituyendo la aritmética de polinomios por arimética modular para el cálculo eficiente de potencias modular así como el cálculo del máximo común divisor de dos números. Se realiza una lista de cincuenta números, primos y compuestos, de entre 30 y 44 bits, ejecutando el algoritmo con cada uno de ellos para comprobar que el programa implementado identifica correctamente los números primos de la lista. Finalmente se hace una comparación de los tiempos de ejecucion con la complejidad del algoritmo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Un número primo p es un entero mayor o igual a 2 que sólo admite a 1 y él mismo como divisores. Hace más de 2000 años, Euclides de Alejandría probó que existe una infinidad de números primos [1]. En 1801 Gauss demostró que todo entero mayor que 2 admite una descomposición como producto de primos que es única salvo el orden [2], hoy en día este hecho se conoce como el *Teorema Fundamental de la Aritmética*. En este contexto, los números primos pueden entenderse como las unidades indivisibles del sistema multiplicativo de los enteros.

Decidir si un entero dado es primo es un problema clásico en las Ciencias de la Computación con aplicaciones en criptografía y en el cómputo científico [3]. Hasta 2004 no se sabía de una prueba de primalidad de tiempo polinomial, es decir, no se conocia un algoritmo que determinara si un entero n es primo o no con un número de operaciones menor que $(\log n)^A$, para alguna constante positiva A. En 2004 Manindra Agrawal, Neeraj Kayal y Nitin Saxena [4] finalmente probaron que se puede determinar en tiempo polinomial si un entero es primo.

En este proyecto se desea presentar un estudio del algoritmo AKS con las siguientes caracteristicas.

- Implementación del algoritmo AKS y su ejecución en al menos cincuenta enteros de entre 30 y 40 bits.
- Comparación de los tiempos de ejecución obtenidos teoricamente y mediante el algoritmo implementado.

1.1. Antecedentes

Proyectos de Integración o Terminales.

- **Diseño, implementación y comparación de los métodos de encriptación RSA en aritmética modular y Massey-Omura en curvas elípticas** [5].
  Presenta un estudio e implementa el algoritmo de encriptación de llave pública RSA en $\mathbb{F}_p$ y el algoritmo de Massey–Omura en curvas elípticas en $\mathbb{F}_p$. El algoritmo AKS también está

basado en la aritmética de $\mathbb{F}_p$. En particular, en el criptosistema RSA la determinación de números primos es fundamental para la construcción de la llave privada.

- **Implementación en software-hardware de aritmética sobre campos finitos binarios $\mathbb{F}_{2^m}$ en curvas elípticas para aplicaciones criptográficas de llave pública** [6].
  Presenta un estudio e implementación de la generación de llaves públicas mediante la utilización de curvas elípticas. Se menciona la generación de llaves públicas en el algoritmo RSA, en el que la determinación de números primos es esencial.

Artículos y Software

- **PRIMES is in P** [4].
  En este artículo se presenta el algoritmo AKS que es el test de primalidad en tiempo polinomial. La diferencia con este proyecto, es que el algoritmo no sé implementó.
- **On distinguishing prime numbers from composite numbers** [7].
  Se presenta un algoritmo que decide si un número es compuesto o primo en un tiempo casi polinomial. También se presentan dos versiones del algoritmo, uno determinístico y otro probabilistico. En este proyecto se estudia e implementa el test de primalidad AKS, el cual es cudeterminístico en tiempo polinomial.
- **Probabilistic Algorithm for Testing Primality** [8].
  Presenta un algoritmo probabilístico para probar la primalidad de números grandes. Este test es determinístico en cuanto a números compuestos. Al igual que el trabajo de Rabin, el algoritmo AKS tampoco asume como verdadera la *ERH* (Hipótesis Extendida de Riemann). La diferencia es que el test de primalidad AKS es determinístico en cuanto a primalidad y composición.
- **Implementation of the AKS algorithm** [9].
  Se implementa el algoritmo AKS en lenguaje C++, empleando la biblioteca GMP. La diferencia es que en este proyecto se implementa el algoritmo AKS, sustituyendo la aritmética de polinomios por aritmética modular, además se diseñan e implementan módulos para calcular el máximo común divisor de dos números empleando el *Algoritmo de Euclides Extendido*. Así mismo se realizará una tabla y una gráfica comparativa de los tiempos estimados respecto al número de cifras y los obtenidos de la ejecución del algoritmo AKS.

1.2. Justificación

Las propiedades de los números primos han sido un tema de intenso estudio en las Matemáticas y de gran impacto en las Ciencias de la Computación. Los números primos son base de estructuras aritméticas conocidas como campos finitos y curvas elípticas donde se han desarrollado criptosistemas exitosos tales como RSA, ElGamal y Massey–Omura. En este contexto, determinar números primos grandes y de características particulares es fundamental ya que juegan un papel fundamental en la seguridad [10], [3], [11].

La búsqueda de números primos es complicada y por sí misma un tema de estudio. Por ejemplo, dado un entero n, no se conoce un algoritmo eficiente y explícito que devuelva al n-ésimo número

primo. Por otra parte, un reto clásico en la materia consiste en "cazar" primos tan grandes como sea posible, es decir, mostrarlos explícitamente. La EFF -Electronic Frontier Foundation- ofrece premios por encontrar primos grandes [12], por ejemplo a quién exhiba un primo con un mil millones de cifras.

El trabajo de Agrawal-Kayal-Saxena[4] respondió una de las preguntas célebres en Ciencias de la Computación. Algunos de los trabajos más importantes que le anteceden son.

- *La criba de Eratóstenes* creada alrededor de 300 años A.C. [13], determina si el entero N es primo con una complejidad del orden $\sqrt{N}$ y asumiendo que se conocen los primos hasta $[\sqrt{N}] + 1$.
- En 1975 Miller [14] probó que existe un algoritmo polinomial para decidir si un entero es primo. Sin embargo, su prueba es de tipo condicional ya que asume la veracidad de la llamada *Hipótesis extendida de Riemann.*
- En 1983 Adleman-Pomerance-Rumely [7] establecieron de manera incondicional que existe una prueba de primalidad de orden

  $$(\log N)^{C(\log\log\log N)},$$

  donde $C > 0$ es una constante que no depende del parámetro N y N es suficientemente grande.
- En 2004 Agrawal-Kayal-Saxena [4] finalmente presentaron un algoritmo del orden

  $$(\log N)^{15/2}(\log\log N)^{A},$$

  donde $A > 0$ es una constante que no depende de N y $N > 0$ es suficientemente grande.
- Lenstra hizo algunas modificaciones al algoritmo AKS, logrando tiempos de ejecución de orden $(\log N)^{9/2}$. Por otra parte, Agrawal, Kayal y Saxena observan en su artículo [4] que se puede obtener un algoritmo del orden

  $$(\log N)^{3}(\log\log\log N)^{A},$$

  con $A > 0$, si se asumen verdaderas ciertas congruencias.

Capítulo 2

Objetivos

2.1. Objetivo General

Implementar el test de primalidad en tiempo polinomial AKS con el fin de realizar pruebas con al menos cincuenta números enteros positivos de entre treinta y cuarenta bits y posteriormente hacer una tabla comparativa de los tiempos estimados y los tiempos obtenidos en la ejecución del programa.

2.2. Objetivos Específicos

- Diseñar e implementar un módulo para la captura y validación de datos, en el cual se descarten enteros negativos, el cero y potencias perfectas.
- Diseñar e implementar un módulo tal que dado un entero n, permita encontrar al mínimo entero positivo r tal que $\text{ord}_r(n) > \log^2 n$.
- Realizar un modulo para implementar el algoritmo extendido de la división y además calcular el máximo común divisor (k, n), para todos los enteros $k \leq n$.
- Elaborar una tabla con cincuenta números positivos de entre treinta y cuarenta bits, entre los cuales se sabrá de antemano quiénes son primos.
- Implementar un módulo en Phyton que utilice el algoritmo AKS para ejecutarlo con cada uno de los números con el fin de elaborar una tabla con los tiempos de ejecución y otra tabla para verificar si se ejecutó correctamente.
- Elaborar una tabla que muestre el orden del algoritmo para cada entero de la lista.
- Elaborar una tabla y una gráfica comparativa de los tiempos estimados y los obtenidos en la ejecución del algoritmo AKS para cada uno de los enteros de la lista.

Capítulo 3

Marco teórico

3.1. Divisibilidad

Esta sección, así como las secciones 3.2 y 3.3, tienen como objetivo presentar una serie de definiciones y resultados que conforman la base para construir el algoritmo AKS.

Los teoremas se presentan sin demostraciones, las cuales se pueden consultar en los libros de Koblitz [3], Buchmann [10] y Vinogradov [15].

Definición 3.1.1. Sean a y b enteros. Se dice que b divide a a si existe un entero q tal que $a = bq$, y se denota por $b \mid a$. Dado un entero a, un divisor es un entero $b \geq 1$ tal que $b \mid a$.

Teorema 3.1.1 (Algoritmo de la división de Euclides)**.** *Si a, b son enteros y $b > 0$, entonces existe una pareja única de enteros, q y r que satisfacen*

$$a = qb + r \qquad \textit{tal que} \quad 0 \leq r < b.$$

Si $r = 0$, entonces b divide exactamente a a y se denota como $b|a$. En caso contrario $b \nmid a$, es decir b no divide a a, siendo q el cociente y r el residuo de dicha división.

Teorema 3.1.2. *Sean a, b y c enteros. Entonces tienen lugar las siguientes propiedades.*

- *Si* $c \mid b$ *y* $b \mid a$, *entonces* $c \mid a$.
- *Si* $b \mid a$, *entonces* $cb \mid ca$ *para todo entero c distinto de cero.*
- *Si* $c \mid b$ *y* $c \mid a$, *entonces $c \mid ax + by$, para cualesquiera enteros x y y.*
- *Si* $b \mid a$ *y* $a \neq 0$, *entonces* $|b| \leq |a|$.
- *Si* $b \mid a$ *y* $a \mid b$, *entonces* $|b| = |a|$.

3.1.1. Máximo común divisor

Definición 3.1.2. Se dice que c es divisor común de a y b si $c \mid a$ y $c \mid b$.

Observe que 1 es común divisor de cualquier par de enteros a y b.

Definición 3.1.3. Sean a, b enteros. El máximo común divisor de a y b se denota por $gcd(a,b)$ y satisface:

- El entero $gcd(a,b)$ es divisor común de a y b; $gcd(a,b) \mid a$ y $gcd(a,b) \mid b$.
- Si existe un entero k tal que $k \mid a$ y $k \mid b$, entonces $k \leq gcd(a,b)$.

El máximo común divisor también se denota por (a,b).

Teorema 3.1.3. *El conjunto de todas las combinaciones lineales de enteros de a y b, es el conjunto de todos los enteros múltiplos de* $gcd(a,b)$.

$$\{ax + by \ : \ a,b \ \in \mathbb{Z}\} = \{c\boldsymbol{gcd}(a,b) \ : \ c \in \mathbb{Z}\}.$$

Corolario 3.1.3.1. *Existen enteros* $x \ y \ y$ *que satisfacen* $gcd(a,b) = ax + by$. *Más aún,* $gcd(a,b)$ *es la mínima combinación lineal positiva de a y b.*

3.1.2. Números primos

Definición 3.1.4. Un número primo es todo entero $p \geq 2$ que admite únicamente dos divisores, 1 y p.

Teorema 3.1.4. *Todo entero* $n > 1$ *tiene un divisor primo.*

Teorema 3.1.5 (Teorema fundamental de la aritmética)**.** *Todo entero* $n > 1$ *puede ser escrito de manera única como producto de números primos, salvo el orden.*

Teorema 3.1.6. *Existe una infinidad de números primos.*

3.2. Algoritmo de Euclides

El algoritmo de Euclides permite calcular de manera eficiente el máximo común divisor de dos números. A continuación se da una breve descripción.

Lema 3.2.1. *Si* $a = qb + r$ *entonces* $gcd(a,b) = gcd(b,r)$.

De esta forma se reducen el tamaño de los enteros dados, a y b, sin alterar su máximo común divisor. Sean a, b dos enteros dados, al menos uno distinto de 0, se desea determinar $gcd(a,b)$.

$$gcd(a,b) = |a| \qquad si \quad b = 0,$$

$$gcd(a,b) = |b| \qquad si \quad a = 0.$$

Ahora suponga que $a \geq b$, entonces $gcd(a,b) = a$ si $a = b$. Se puede asumir que

$$a \ > \ b \ > \ 0.$$

Empleando el teorema 3.1.1, se divide a entre b

$$a = q_1 b + r_1 \qquad con \qquad 0 \leq r_1 < b.$$

Si $r_1 = 0$, $gcd(m,n) = n$. Por otro lado, si $r_1 \neq 0$, se dividide r_1 entre n

$$b = q_2 r_1 + r_2 \qquad con \qquad 0 \leq r_2 < r_1.$$

Si $r_2 = 0$, por el lema 3.2.1 se tiene que $gcd(a,b) = gcd(b,r_1) = r_1$. Si $r_2 \neq 0$, entonces

$$r_1 = q_3 r_2 + r_3 \qquad con \qquad 0 \leq r_3 < r_2.$$

Se continúa hasta obtener un $r_k = 0$. En las dos últimas iteraciones se obtiene

$$r_{k-3} = q_{k-1} r_{k-2} + r_{k-1} \qquad con \qquad 0 < r_{k-1} < r_{k-2}$$

$$r_{k-2} = q_k r_{k-1} + r_k \qquad con \qquad r_k = 0.$$

Teorema 3.2.2. *El máximo común divisor de a y b es el último residuo diferente de cero, r_{k-1}.*

3.2.1. Algoritmo de Euclides Extendido

El Algoritmo de Euclides Extendido permite encontrar los enteros x y y que satisfacen $gcd(a,b) = ax + by$ (corolario 3.1.3.1).

Teorema 3.2.3. *Es posible representar el máximo común divisor de a y b como una combinación lineal con coeficientes x y y tales que* $x = (-1)^k x_k$ *y* $y = (-1)^{k+1} y_k$. *Teniendo*

$$r_i = (-1)^i x_i a + (-1)^{i+1} y_i b \qquad para \quad 0 \leq i \leq k+1.$$

3.3. Aritmética modular

Definición 3.3.1. Dados tres enteros, a, b y q, se dice que a es congruente con b módulo q, denotado como $a \equiv b(mod q)$, si q divide a $b - a$.

Teorema 3.3.1. *La congruencia es una relación de equivalencia en los enteros.*

1. $a \equiv b(mod\ q)$ (reflexiva).
2. $a \equiv b(mod\ q)$ implica $b \equiv a(mod\ q)$ (simétrica).
3. $a \equiv b(mod\ q)$ y $b \equiv c(mod\ q)$ implica que $a \equiv c(mod\ q)$ (transitiva).

Lema 3.3.2. *Las expresiones siguientes son equivalentes.*

1. $a \equiv b(mod\ q)$.

2. Existe un entero k tal que $a = b + kq$.
3. Al dividir a y b por q, ambos tienen el mismo residuo.

La clase de equivalencia de a consiste en todos los enteros obtenidos de la suma de a y los enteros múltiples de q.

$$\{b : b \equiv a(mod\ q)\} = a + q\mathbb{Z}.$$

Esta clase de equivalencia es la *clase residual* de a (mod q). El conjunto de clases residuales módulo q se denota como $\mathbb{Z}/q\mathbb{Z}$.

Lema 3.3.3. *Todo entero es congruente módulo q, a un sólo y único entero entre 0 y $q-1$. Es decir, todo entero pertenece a una sola clase residual módulo q.*

Teorema 3.3.4. $a \equiv b\ (mod\ q) \quad y \quad c \equiv d\ (mod\ q)$ *implican:*

- $-a \equiv -b\ (mod\ q)$.
- $a + c \equiv b + d\ (mod\ q)$.
- $ac \equiv bd\ (mod\ q)$.

Teorema 3.3.5 (Pequeño Teorema de Fermat)**.** *El pequeño teorema de Fermat se enuncia en la siguiente forma. Sea p un número primo, entonces para cualquier entero a tal que $(a, p) = 1$ se tiene*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Definición 3.3.2. Para todo entero $(a, p) = 1$, se define el orden de a módulo p como

$$\text{ord}_p(a) = \text{mín}\{1 \leq k \leq p-1 \ :\ a^k \equiv 1 \pmod{p}\}.$$

Corolario 3.3.5.1. *Para todo entero $(a, p) = 1$ se tiene $ord_p(a)|p-1$.*

3.4. Algoritmo AKS

El algoritmo AKS se basa en una generalización del *Pequeño Teorema de Fermat*. Sea $a \in \mathbb{Z}$, $n \in \mathbb{N}$, con $n \geq 2$ y $gcd(a, n) = 1$. Entonces n es primo si y sólo sí se cumple

$$(X + a)^n = X^n + a\ (mod\ n).$$

Sin embargo, esta prueba no es óptima ya que falla en ciertos números conocidos como *pseudoprimos* y tiene una complejidad $O(n)$, teniendo que evaluar con n coeficientes en el lado izquierdo de la ecuación. Por ello Agrawal, Kayal y Saxena hicieron una modificación, de modo que se reduce el número de coeficientes evaluando ambos lados con un polinomio de la forma $X^r - 1$, para una r pequeña.

$$(X + a)^n \equiv X^n + a\ (mod\ X^r - 1,\ n).$$

Los valores de a y r están limitados por un polinomio en $\log n$. Así, se tiene un algoritmo deterministico en tiempo polinomial.

En terminos generales, el algoritmo AKS se puede plantear de la manera siguiente [16]. Dado un entero $n \geq 2$, r un entero positivo menor que n, para el cual el orden de $n > (\log n)^2$ módulo r. Entonces n es primo únicamente si

- n no es potencia perfecta,
- n no tiene ningún factor primo menor o igual a r,
- $(X + a)^n \equiv X^n + a \ (mod\ X^r - 1,\ n)$.

Lenstra y Pomerance hicieron algunas mejoras al algoritmo de Agrawal, Kayal y Saxena [7]:

Entrada: entero $n > 1$.
1: Si ($n = a^b$ para $a \in N$ y $b > 1$), salida COMPUESTO.
2: Encontrar el mínimo r tal que $O_r(n) > \log^2 n$.
3: Si $gcd(a, n) \neq 1$ para toda $a \leq r$, salida COMPUESTO.
4: Desde $a = 1$ hasta $\lfloor \sqrt{r} \log n \rfloor$ hacer
Si $((X + a)^n \equiv X^n + a (\mathrm{mod} X^r - 1, n))$, salida PRIMO;

Figura 3.1: Algoritmo AKS

Una forma de calcular r es la siguiente.

- $q > \lfloor (\log^2\ n) \rfloor$.
- Calcular $n^j\ mod\ q$ para j=1,2,...,$\lfloor (\log^2\ n) \rfloor$.
- Si el residuo es igual a 1, entonces incrementar q en uno.
- De otra forma, r es igual a q.

En este proyecto se sustituye la aritmética de polinomios por arimética modular. Así, se propone trabajar con una constante auxiliar A, lo suficientemente grande para que a cada valor de a le corresponda única y exclusivamente una clase residual

$$(A + a)^n \equiv A^n + a \ (mod\ A^r - 1,\ n).$$

Sea $p(x)$ un polinomio, cuyo grado sea menor que r

$$p(x) = a_0 + a_1 x + \cdots + a_{r-1} x^{r-1}.$$

1. Considerar $p(x) = a_0 + a_1 x + \cdots + a_{r-1} x^{r-1}$, con $a_i \in \mathbb{Z}/n\mathbb{Z}$.
2. Calcular $p^2(x)$ y reducir módulo $(X^r - 1,\ n)$.
3. $p^2(x)\ (mod\ X^r - 1,\ n)$, repetir los pasos 1 y 2 hasta calcular $(X + a)^n$.

Dado $p(x) = a_0 + a_1 x + \cdots + a_{r-1}x^{r-1}$

$$p^2(x) = C_0 + C_1 x + \cdots + C_{2r-2}x^{2r-2},$$

$$C_k = \sum a_i a_{k-i} \qquad donde \quad max\{(2r-2)-k\} \leq i \leq min\{k, 2r-2\},$$

$$|C_k| \leq \sum |a_i||a_{k-i}| \leq (n)^2(r) = A,$$

$$p(x) \rightarrow \mathbb{Z}/(2A)^{2r-1}\mathbb{Z}, \qquad (2A)^{2r-2} = (2n^2r)^{2r-2}.$$

Para fines prácticos, sea $A = 2n^2r$.

$$p(x) \rightarrow p(A) = a_0 + a_1(A) + a_2(A)^2 + \cdots a_{r-1}(A)^{r-1}.$$

Una manera óptima de calcular el lado izuiqerdo de la expresión $(A + a)^n \equiv A^n + a \ (mod\ A^r - 1,\ n)$ es empleando el *Algoritmo de Euclides Extendido*, escribiendo la potencia n en su forma binaria y reduciendo módulo $A^r - 1$, módulo n en cada iteración.

$$n = \delta_0 + 2\delta_1 + ... + 2^\ell \delta_\ell, \qquad \delta_i = 0, 1,$$

$$f = (A + a),$$

$$f^n = (A + a)^n = (A + a)^{\delta_0 + 2\delta_1 + ... + 2^\ell \delta_\ell},$$

$$f^n = (A + a)^n = (A + a)^{\delta_0}(A + a)^{2\delta_1}(A + a)^{2^2\delta_2} \cdots (A + a)^{2^\ell \delta_\ell},$$

$$f^n = (A + a)^n = ((A + a)^{2^0})^{\delta_0}((A + a)^{2^1})^{\delta_1}((A + a)^{2^2})^{\delta_2} \cdots ((A + a)^{2^\ell})^{\delta_\ell},$$

$$f^n = ((f)^{2^0})^{\delta_0}((f)^{2^1})^{\delta_1}((f)^{2^2})^{\delta_2} \cdots ((f)^{2^\ell})^{\delta_\ell}.$$

Una desventaja de emplear la constante A es que no es posible calcular $(A + a)^n \ (mod\ n)$ coeficiente a coeficiente de forma directa.
A continuación se presenta la idea general para calcular $(A + a)^n \ (mod\ A^r - 1,\ n)$, recuperando los coeficientes módulo A para $(A + a)^2$.

$$X + a \rightarrow A + a \quad \in \mathbb{Z}/A^{2r-1}\mathbb{Z}.$$

1. Sea $p(x) = X + a$ el polinomio.
2. Entonces $p^2(A) = (A + a)^2 = a^2 + 2A + A^2$.
3. Reducir el número de coeficientes haciendo módulo $A^r - 1$.

$$p^2(A) \ (mod\ A^r - 1).$$

4. Se obtiene la siguiente expresión

$$(p_{mod\ r})^2(A) \ = \ C_0 \ + \ C_1 A \ + \ \cdots \ + \ C_{r-1}A^{r-1}.$$

5. Recuperar el primer coeficiente C_0

$$C_0 \ = \ (p_{mod\ r})^2 \ (mod \ A).$$

6. Ajustar $(p_{mod\ r})^2$, para calcular el siguiente coeficiente, empleando una variable auxiliar p_3

$$p_3 \ = \ \lfloor \frac{(p_{mod\ r})^2 \ - \ C_0}{A} \rfloor.$$

7. Recalcular el coeficiente C_0 haciendo módulo n.

$$C_0 \ = \ C_0 \ (mod \ n).$$

8. De esta forma es posible reconstruir un polinomio $(mod \ A^r \ - 1, \ n)$.

$$square \ = \ C_0 A^0.$$

9. Se sigue el mismo procedimiento para calcular los siguientes coeficientes, hasta calcular $square \ = \ C_0A^0 \ + \ C_1A^1 \ + \ C_2A^2 \ + \ \cdots \ + \ C_{r-1}A^{r-1}$.

$$C_1 \ = \ p_3 \ (mod \ A),$$

$$p_3 \ = \ \lfloor \frac{p_3 \ - \ C_1}{A} \rfloor,$$

$$C_1 \ = \ C_1 \ (mod \ n),$$

$$square \ = \ square \ + \ C_1A^1$$

$$\cdots$$

Hasta ahora sólo se ha visto lo que sucede con el lado izquierdo de la expresión $(X \ + \ a)^n \ \equiv \ X^n \ + \ a \ (mod \ X^r \ - 1 \ , \ n)$. En el lado derecho de la misma, se observa que se debe calcular $X^n \ + \ a \ (mod \ X^r \ - \ 1, \ n)$, por lo que nuevamente se debe encontrar una forma óptima de hacer $X^n \ (mod \ X^r \ - \ 1)$.

$$X^n \ = \ (X^r \ - \ 1)t(x) \ + \ s(x), \qquad deg \ s < r$$

$$n = qr \ + \ k, \qquad deg \ k < r$$

$$
\begin{aligned}
X^n &= X^{qr+k} \\
&= X^r X^{r(q-1)+k} \\
&= (X^r - 1)X^{r(q-1)+k} \ + \ X^{r(q-1)+k} \\
&= (X^r - 1)X^{r(q-1)+k} \ + \ X^r X^{r(q-2)+k} \\
&= (X^r - 1)X^{r(q-1)+k} \ + \ (X^r - 1)X^{r(q-2)+k} \ + \ X^{r(q-2)+k} \\
&= (X^r - 1)\left(X^{r(q-1)+k} \ + \ X^{r(q-2)+k}\right) + X^{r(q-2)+k} \\
&= \cdots = (X^r - 1)t(x) + X^k
\end{aligned}
\tag{3.1}
$$

Así se tiene que $X^n \ \equiv \ X^k \ (mod \ X^r \ - \ 1)$, y al evaluar en A:

$$A^n \ \equiv \ A^k \ (mod \ A^r \ - \ 1).$$

Capítulo 4

Desarrollo del proyecto

4.1. Instalación de biblioteca GMPY2

GMPY2 es un módulo de extensión de Python que proporciona funciones aritméticas de múltiple precisión. Entre las funciones utilizadas en este proyecto están `mul()` y `is_power()`, para determinar si un número es potencia perfecta en la validación de datos. Para su instalación se ejecutaron los siguientes comandos desde la terminal.

```
sudo apt-get update
sudo apt-get install python-gmpy2
```

4.2. Módulo para determinar el parámetro auxiliar r

El primer paso importante para implementar el algoritmo AKS, fue determinar el parámetro auxilar r tal que $ord_r(n) > \log^2(n)$. El algoritmo 1 determina dicho parámetro auxiliar r, empleando la función `modulo(n,j,q)`, mostrada en el algoritmo 2, la cual calcula n^j mód q mediante reducción de potencias módulo q.

Algoritmo 1 $ord_r(n) > \log^2(n)$

```
1: q0 ← ⌊log² n⌋ + 1
2: q ← q0
3: while 1 do
4:     flag ← True
5:     for j ∈ {1, ..., q0} do
6:         res ← modulo(n, j, q)
7:         if res == 1 then
8:             q ← q + 1
9:             flag ← False
10:            break
11:        end if
12:    end for
13:    if flag then
14:        r ← q
15:        break
16:    end if
17: end while
```

Algoritmo 2 Cálculo de potencias módulo q

```
1: procedure MODULO(n, j, q)
2:     k ← j
3:     pow1 ← 1
4:     pow2 ← 1
5:     while k > 0 do
6:         d ← k %2
7:         s ← pow(n, d * pow2)
8:         pow1 ← (pow1 * s) %p
9:         pow2 ← pow2 * 2
10:        k ← (k − d)/2
11:    end while
12:    return pow1
13: end procedure
```

4.3. Módulo prueba de composición

El siguiente paso es hacer una prueba de composición. Se tiene que verificar que n no tenga ningún factor primo menor o igual que r; de este modo es necesario calcular el máximo común divisor de n y todos los $k \leq r$, tal como se muestra en el algoritmo 3. Se implementó esta función en el algoritmo 4 para determinar si n es compuesto o es candidato a ser un número primo.

Algoritmo 3 Máximo común divisor

```
1:  procedure MY_GCD(m, n)
2:      t ← [[1, 0], [0, 1]]
3:      tk ← [[0, 0], [0, 0]]
4:      e ← [[0, 1], [1, 0]]
5:      k ← 0
6:      r0 ← m
7:      r1 ← n
8:      r2 ← 1
9:      while r2 ≠ 0 do
10:         q ← (r0/r1)
11:         e[0][0] ← q
12:         for i ∈ {0, ..., 2} do
13:             for j ∈ {0, ..., 2} do
14:                 for w ∈ {0, ..., 2} do
15:                     tk[i][j] ← tk[i][j] + (t[i][w] * e[w][j])
16:                 end for
17:             end for
18:         end for
19:         for i ∈ {0, ..., 2} do
20:             for j ∈ {0, ..., 2} do
21:                 t[i][j] ← tk[i][j]
22:                 tk[i][j] ← 0
23:             end for
24:         end for
25:         x ← t[1][0]
26:         y ← t[0][0]
27:         r2 ← (−1)^k * x * m + (−1)^(k+1) * y * n
28:         r0 ← r1
29:         r1 ← r2
30:         k ← k + 1
31:     end while
32:     return r0
33: end procedure
```

Algoritmo 4 Prueba de composición

```
1: primo ← False
2: flag ← True
3: a ← 1
4: while flag do
5:     mcd ← my_gcd(n, a)
6:     if mcd > 1 then
7:         flag ← False                          ▷ n es compuesto
8:     else
9:         if mcd == 1 then
10:            a ← a + 1
11:        end if
12:    end if
13:    if a ≥ r then
14:        primo ← True                          ▷ n es candidato a número primo
15:        break
16:    end if
17: end while
```

4.4. Módulo para prueba de primalidad

En este módulo se realiza una última prueba para determinar si el número es primo o compuesto. En los algoritmos 5 y 6 se muestra cómo se verifica $(A+a)^n \neq A^n + a(\mathrm{mod} X^r - 1, n)$ para todas las a tal que $1 \leq a \leq \sqrt{r}\log n$, para determinar si n es primo o compuesto.

Algoritmo 5 Prueba de primalidad

```
1: if primo then
2:     n1 ← n
3:     parar ← √r * log n
4:     A ← 2 * n^2 * r
5:     g ← A^r − 1
6:     k ← n mód r
7:     flag ← True
8:     for a ∈ {1, ..., parar} do
9:         f ← A + a
10:        d ← n1 mód 2
11:        pow0 ← f^d
12:        pow2 ← f
13:        n1 ← (n1 − d)/2
```

Algoritmo 6 Prueba de primalidad (continuación)

```
        while n1>0 do
            pow2 ← pow2^2 mód g
            p3 ← pow2
            j, square ← 0
            while p3>0 do
                C1 ← p3 mód A
                p3 ← (p3 − C1)/A
                C2 ← C1 mód n
                square ← square + C2 ∗ A^j
                j ← j + 1
            end while
            pow2 ← square
            d ← n1 mód 2
            pow0 ← pow0 ∗ pow2^d
            pow0 ← pow0 mód g
            p3 ← pow0
            l, square ← 0
            while p3>0 do
                C1 ← p3 mód A
                p3 ← (p3 − C1)/A
                C2 ← C1 mód n
                square ← square + C2 ∗ A^l
                l ← l + 1
            end while
            pow0 ← square
            n1 ← (n1 − d)/2
        end while
        xn_a ← A^k + a
        if (square − xn_a) ≠ 0 then
            flag ← False                                  ▷ n es COMPUESTO
            break
        end if
        n1 ← n
    end for
    if flag then
        primo ← True                                      ▷ n es PRIMO
    end if
end if
```

4.5. Lista de números

Para obtener la lista de números se consultó *The Nth Prime Page* [17]. Se eligieron números de entre 2^{30} y 2^{40} bits. En la tabla 4.1 se muestran los números primos y en la tabla 4.2 se listan los números compuestos.

Número	Primo
1073676287	Sí
1088888881	Sí
1095912793	Sí
2599291451	Sí
3211891519	Sí
4076863487	Sí
8192454631	Sí
10000000019	Sí
15984784979	Sí
24571243009	Sí
32212254719	Sí
50006393431	Sí
99999199999	Sí
100123456789	Sí
260389232731	Sí
393142151459	Sí
555555577777	Sí
753352617167	Sí
999998999999	Sí
999999000001	Sí
2999999899999	Sí
3664533202453	Sí
6056529316217	Sí
7142857142857	Sí
8284590452353	Sí
9999999998987	Sí

Tabla 4.1: Tabla números primos.

Número	Primo
1117175145	No
1123456987	No
1136972771	No
2754425310	No
3215031751	No
4294967295	No
10123456897	No
17392546081	No
25505418241	No
32149366346	No
54047322253	No
97524222465	No
111111111111	No
123455554321	No
334826628433	No
743008370701	No
774630686237	No
799980626860	No
2935561623745	No
3465253618401	No
6252893229398	No
7625597484960	No
8284590452382	No
9999999998980	No

Tabla 4.2: Tabla números compuestos.

Capítulo 5

Resultados

La tabla 5.1 muestra la respuesta del algoritmo con los veintiséis números primos y sus tiempos de ejecución en minutos. El programa logró determinar correctamente que los números son primos.

Número	Primo	Ejecución	Tiempo (minutos)
1073676287	Sí	Correcto	2.7555319
1088888881	Sí	Correcto	2.6355870
1095912793	Sí	Correcto	2.7527394
2599291451	Sí	Correcto	4.1320456
3211891519	Sí	Correcto	4.6077504
4076863487	Sí	Correcto	4.8848695
8192454631	Sí	Correcto	5.7853457
10000000019	Sí	Correcto	6.9305827
15984784979	Sí	Correcto	8.1431382
24571243009	Sí	Correcto	9.9463035
32212254719	Sí	Correcto	11.0341768
50006393431	Sí	Correcto	11.5024914
99999199999	Sí	Correcto	14.4309558
100123456789	Sí	Correcto	14.2850574
260389232731	Sí	Correcto	22.7061393
393142151459	Sí	Correcto	25.0834130
555555577777	Sí	Correcto	27.7959947
753352617167	Sí	Correcto	29.7498304
999998999999	Sí	Correcto	25.6455209
999999000001	Sí	Correcto	31.0190203
2999999899999	Sí	Correcto	36.7226778
3664533202453	Sí	Correcto	37.7533985
6056529316217	Sí	Correcto	45.2790234
7142857142857	Sí	Correcto	45.3615262

Continúa en página siguiente

Número	Primo	Ejecución	Tiempo (minutos)
8284590452353	Sí	Correcto	62.2013448
9999999998987	Sí	Correcto	66.3672222

Tabla 5.1: Resultados de ejecución con números primos

La grafica 5.1 muestra el tiempo de ejecución en minutos del programa para cada uno de los veinteséis números primos. El tiempo de ejecución está entre los 2 y 66 minutos aproximadamente; se observa que el programa comenzó a tardar más a partir de los números primos de doce dígitos.

Figura 5.1: Gráfica de tiempos de ejecución con números primos

En la tabla 5.2 se muestra la respuesta del algoritmo con los veinticuatro números compuestos y se observa que también determinó correctamente que los números no son primos; la tabla también muestra sus tiempos de ejecución en minutos.

Número	Primo	Ejecución	Tiempo (minutos)
1117175145	No	Correcto	0.0001761
1123456987	No	Correcto	0.0063331
1136972771	No	Correcto	0.0002774
2754425310	No	Correcto	0.0005646
3215031751	No	Correcto	0.0009849
4294967295	No	Correcto	0.0002930
10123456897	No	Correcto	0.0014348
17392546081	No	Correcto	0.0008311
25505418241	No	Correcto	0.0174213
32149366346	No	Correcto	0.0006434
54047322253	No	Correcto	0.0013920
97524222465	No	Correcto	0.0005384
111111111111	No	Correcto	0.0011370
123455554321	No	Correcto	0.0006584
334826628433	No	Correcto	0.0006694
743008370701	No	Correcto	0.0012439
774630686237	No	Correcto	0.0013340
799980626860	No	Correcto	0.0009013
2935561623745	No	Correcto	0.0010258
3465253618401	No	Correcto	0.0020959
6252893229398	No	Correcto	0.0013447
7625597484960	No	Correcto	0.0011260
8284590452382	No	Correcto	0.0011889
9999999998980	No	Correcto	0.0018245

Tabla 5.2: Tabla números compuestos.

La grafica 5.2 muestra el tiempo de ejecución en minutos del programa para cada uno de los veinticuatro números compuestos. El tiempo de ejecución está entre los 0.0001761 y 0.0018245 minutos aproximadamente; se observan dos picos en los que el tiempo fue mayor. Para el número compuesto 1123456987 el tiempo de ejecución es 0.0063331 minutos, mientras que para el número compuesto 25505418241 fue de 0.0174213 minutos. Esto se debe a que dichos números poseen factores primos mayores a la r determinada para cada uno, por lo que la prueba de composición dio como resultado que son "candidatos a ser números primos". Sin embargo, en la última prueba, de primalidad, el programa determinó que son números compuestos.

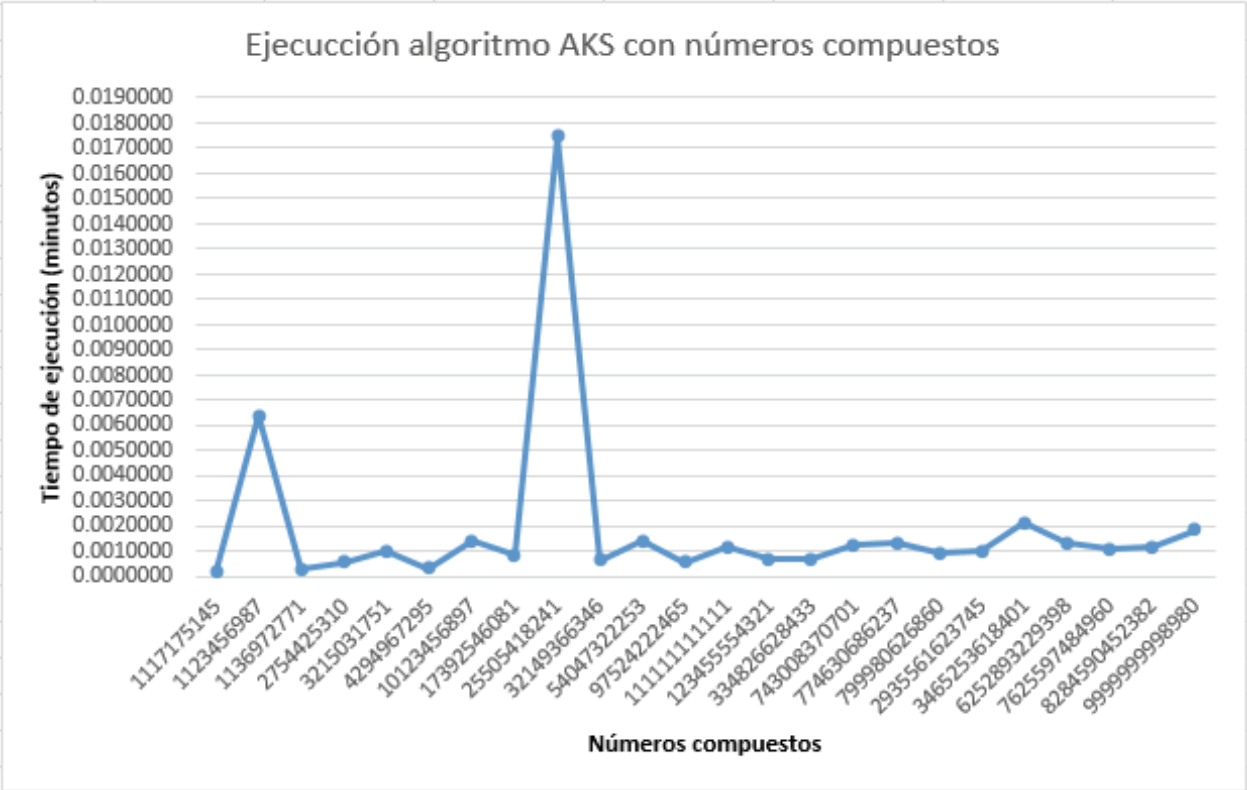

Figura 5.2: Gráfica de tiempos de ejecución con números compuestos

En la tabla 5.3 se registraron los cincuenta números con los resultados de la ejecución del algoritmo AKS. En esta tabla también se puede observar la relación $\frac{t}{d^{7.5}}$, donde t es el tiempo de ejecución para cada número, medido en minutos, y d es el número de dígitos correspondiente a n escrito en binario.

Número	d	Primo	Ejecución	Tiempo (minutos)	$\frac{t}{d^{7.5}}$
1073676287	30	Sí	Correcto	2.7555319	2.30036E-11
1088888881	31	Sí	Correcto	2.6355870	1.72054E-11
1095912793	31	Sí	Correcto	2.7527394	1.79702E-11
1117175145	31	No	Correcto	0.0001761	1.14927E-15
1123456987	31	No	Correcto	0.0063331	4.13429E-14
1136972771	31	No	Correcto	0.0002774	1.81079E-15
2599291451	32	Sí	Correcto	4.1320456	2.12589E-11
2754425310	32	No	Correcto	0.0005646	2.90480E-15
3211891519	32	Sí	Correcto	4.6077504	2.37063E-11
3215031751	32	No	Correcto	0.0009849	5.06728E-15
4076863487	32	Sí	Correcto	4.8848695	2.51321E-11
4294967295	32	No	Correcto	0.0002930	1.50762E-15
8192454631	33	Sí	Correcto	5.7853457	2.36306E-11
10000000019	34	Sí	Correcto	6.9305827	2.26297E-11
10123456897	34	No	Correcto	0.0014348	4.68500E-15
15984784979	34	Sí	Correcto	8.1431382	2.65889E-11
17392546081	35	No	Correcto	0.0008311	2.18349E-15
24571243009	35	Sí	Correcto	9.9463035	2.61307E-11
25505418241	35	No	Correcto	0.0174213	4.57687E-14
32149366346	35	No	Correcto	0.0006434	1.69028E-15
32212254719	35	Sí	Correcto	11.0341768	2.89888E-11
50006393431	36	Sí	Correcto	11.5024914	2.44638E-11
54047322253	36	No	Correcto	0.0013920	2.96050E-15
97524222465	37	No	Correcto	0.0005384	9.32436E-16
99999199999	37	Sí	Correcto	14.4309558	2.49909E-11
100123456789	37	Sí	Correcto	14.2850574	2.47383E-11
111111111111	37	No	Correcto	0.0011370	1.96892E-15
123455554321	37	No	Correcto	0.0006584	1.14019E-15
260389232731	38	Sí	Correcto	22.7061393	3.21934E-11
334826628433	39	No	Correcto	0.0006694	7.81032E-16
393142151459	39	Sí	Correcto	25.0834130	2.92686E-11
555555577777	40	Sí	Correcto	27.7959947	2.68245E-11
743008370701	40	No	Correcto	0.0012439	1.20039E-15
753352617167	40	Sí	Correcto	29.7498304	2.87101E-11
774630686237	40	No	Correcto	0.0013340	1.28733E-15
799980626860	40	No	Correcto	0.0009013	8.69768E-16

Continúa en página siguiente

Número	d	Primo	Ejecución	Tiempo (minutos)	$\frac{t}{d^{7.5}}$
999998999999	40	Sí	Correcto	25.6455209	2.47492E-11
999999000001	40	Sí	Correcto	31.0190203	2.99349E-11
2935561623745	42	No	Correcto	0.0010258	6.86594E-16
2999999899999	42	Sí	Correcto	36.7226778	2.45790E-11
3465253618401	42	No	Correcto	0.0020959	1.40282E-15
3664533202453	42	Sí	Correcto	37.7533985	2.52689E-11
6056529316217	43	Sí	Correcto	45.2790234	2.54029E-11
6252893229398	43	No	Correcto	0.0013447	7.54418E-16
7142857142857	43	Sí	Correcto	45.3615262	2.54492E-11
7625597484960	43	No	Correcto	0.0011260	6.31730E-16
8284590452353	43	Sí	Correcto	62.2013448	3.48969E-11
8284590452382	43	No	Correcto	0.0011889	6.67019E-16
9999999998980	44	No	Correcto	0.0018245	8.61503E-16
9999999998987	44	Sí	Correcto	66.3672222	3.13371E-11

Tabla 5.3: Tiempo de ejecución de los 50 números

En la tabla 5.4 se puede observar la relación $\frac{t}{d^{7.5}}$ para los veintiséis números primos.

Número	d	Primo	Ejecución	Tiempo (minutos)	$\frac{t}{d^{7.5}}$
1073676287	30	Sí	Correcto	2.7555319	2.30036E-11
1088888881	31	Sí	Correcto	2.6355870	1.72054E-11
1095912793	31	Sí	Correcto	2.7527394	1.79702E-11
2599291451	32	Sí	Correcto	4.1320456	2.12589E-11
3211891519	32	Sí	Correcto	4.6077504	2.37063E-11
4076863487	32	Sí	Correcto	4.8848695	2.51321E-11
8192454631	33	Sí	Correcto	5.7853457	2.36306E-11
10000000019	34	Sí	Correcto	6.9305827	2.26297E-11
15984784979	34	Sí	Correcto	8.1431382	2.65889E-11
24571243009	35	Sí	Correcto	9.9463035	2.61307E-11
32212254719	35	Sí	Correcto	11.0341768	2.89888E-11
50006393431	36	Sí	Correcto	11.5024914	2.44638E-11
99999199999	37	Sí	Correcto	14.4309558	2.49909E-11
100123456789	37	Sí	Correcto	14.2850574	2.47383E-11
260389232731	38	Sí	Correcto	22.7061393	3.21934E-11
393142151459	39	Sí	Correcto	25.0834130	2.92686E-11
555555577777	40	Sí	Correcto	27.7959947	2.68245E-11
753352617167	40	Sí	Correcto	29.7498304	2.87101E-11
999998999999	40	Sí	Correcto	25.6455209	2.47492E-11
999999000001	40	Sí	Correcto	31.0190203	2.99349E-11
2999999899999	42	Sí	Correcto	36.7226778	2.45790E-11
3664533202453	42	Sí	Correcto	37.7533985	2.52689E-11
6056529316217	43	Sí	Correcto	45.2790234	2.54029E-11
7142857142857	43	Sí	Correcto	45.3615262	2.54492E-11
8284590452353	43	Sí	Correcto	62.2013448	3.48969E-11
9999999998987	44	Sí	Correcto	66.3672222	3.13371E-11

Tabla 5.4: Tabla números primos.

La grafica 5.3 muestra la relación entre el tiempo de ejecución y la cantidad de dígitos de cada número primo. En general, se observa que los tiempos se distribuyen de sobre una recta. Es decir, el tiempo es constante.

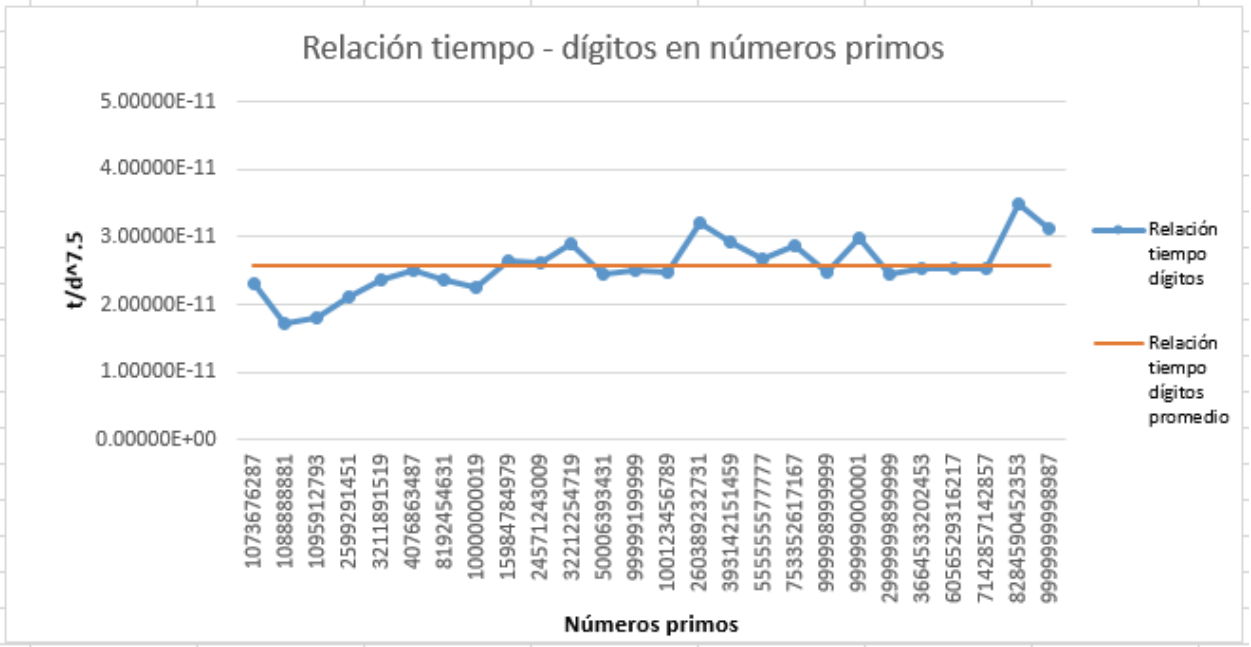

Figura 5.3: Gráfica de relción tiempo de ejecución - dígitos de números primos

En la tabla 5.5 se puede observar la relación $\frac{t}{d^{7.5}}$ para los veinticuatro números compuestos.

Número	d	Primo	Ejecución	Tiempo (minutos)	$\frac{t}{d^{7.5}}$
1117175145	31	No	Correcto	0.0001761	1.14927E-15
1123456987	31	No	Correcto	0.0063331	4.13429E-14
1136972771	31	No	Correcto	0.0002774	1.81079E-15
2754425310	32	No	Correcto	0.0005646	2.90480E-15
3215031751	32	No	Correcto	0.0009849	5.06728E-15
4294967295	32	No	Correcto	0.0002930	1.50762E-15
10123456897	34	No	Correcto	0.0014348	4.68500E-15
17392546081	35	No	Correcto	0.0008311	2.18349E-15
25505418241	35	No	Correcto	0.0174213	4.57687E-14
32149366346	35	No	Correcto	0.0006434	1.69028E-15
54047322253	36	No	Correcto	0.0013920	2.96050E-15
97524222465	37	No	Correcto	0.0005384	9.32436E-16
111111111111	37	No	Correcto	0.0011370	1.96892E-15
123455554321	37	No	Correcto	0.0006584	1.14019E-15
334826628433	39	No	Correcto	0.0006694	7.81032E-16
743008370701	40	No	Correcto	0.0012439	1.20039E-15
774630686237	40	No	Correcto	0.0013340	1.28733E-15
799980626860	40	No	Correcto	0.0009013	8.69768E-16
2935561623745	42	No	Correcto	0.0010258	6.86594E-16
3465253618401	42	No	Correcto	0.0020959	1.40282E-15
6252893229398	43	No	Correcto	0.0013447	7.54418E-16
7625597484960	43	No	Correcto	0.0011260	6.31730E-16
8284590452382	43	No	Correcto	0.0011889	6.67019E-16
9999999998980	44	No	Correcto	0.0018245	8.61503E-16

Tabla 5.5: Tabla números compuestos.

La grafica 5.4 muestra la relación entre el tiempo de ejecución y la cantidad de dígitos de cada número número compuesto. Al igual que en los números primos, se observa que los tiempos se distribuyen de sobre una recta, es decir, el tiempo es constante, pese a que existen dos números, 1123456987 y 25505418241, que fueron candidatos a ser números primos.

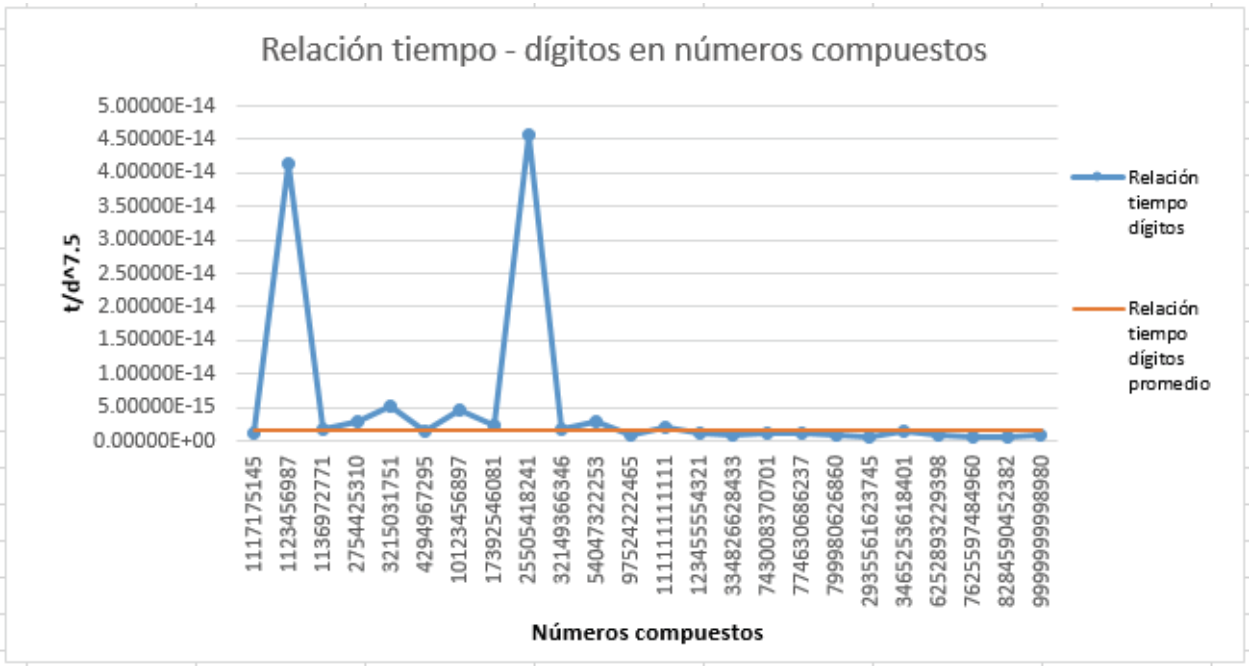

Figura 5.4: Gráfica de relción tiempo de ejecución - dígitos de números compuestos

Capítulo 6

Conclusiones

De los resultados obtenidos en este proyecto, se puede concluir que el algoritmo AKS funciona adecuadamente incluso sustituyendo el empleo de polinomios por aritmética modular, empleando una constante lo suficientemente grande para tener distintas clases residuales. Se resolvió uno de los problemas que representa este algoritmo, la equivalencia modular de dos números elevado a la potencia n, implementando módulos propios para el cálculo eficiente de potencias modular y el máximo común divisor empleando el *Algoritmo de Euclides Extendido*. Para reducir el tiempo de ejecución se utilizó la biblioteca GMPY2 para realizar las multiplicaciones, ya que Python no implementa un método eficiente para multiplicación de números grandes. Por último, el tiempo de ejecución respecto a la cantidad de dígitos, tanto de números primos como de números compuestos, es constante, por ello se puede concluir que se logró implementar de manera exitosa el alrgoritmo AKS.

Apéndice A

Código fuente del algoritmo AKS, implementado en Python

```
import math
import time
from ctypes import *
from gmpy2 import *

def modulo(n, j, q):
    k=j
    pow2=1 #potencias de 2
    pow1=1 #potencia del algoritmo

    while(k>0):
        d = k%2
        s = pow(n, d*pow2)
        pow1=(pow1*s)%q #se calculan las potencias mod q
        pow2=2*pow2 #se calculas las potencias de 2, es decir 2^i
        k =long (k-d)/2

    return pow1



def my_gcd(m, n):
    t = [[1,0],[0,1]]
    tk = [[0,0],[0,0]]
    e = [[0,1],[1,0]]

    #inicializo k, r0, r1 y r2
    k=0
    r0=m
    r1=n
    r2=1

    while(r2!=0):
        q=int(r0/r1) #Se calcula qk = parte entera (r0/r1)
```

```
        e[0][0]=q #Se cambia la entrada de
        #e[0][0] para obtener la matriz ek

        for i in range(0,2,1):
            for j in range(0,2,1):
                for w in range(0,2,1):
                    tk[i][j]=tk[i][j]+(t[i][w]*e[w][j])

        for i in range(0,2,1):
            for j in range(0,2,1):
                t[i][j]=tk[i][j]
                tk[i][j]=0

        #obtengo xk & yk de la matriz tk+1
        x=t[1][0]
        y=t[0][0]

        #Recalcular r2 = (-1)^k(x)(m) + (-1)^(k+1)(y)
        r2=((pow(-1,k))*x*m) + ((pow(-1,k+1))*y*n)

        #Recalcular r0 y r1
        r0=r1
        r1=r2

        k=k+1

    return int(r0)

while(1):
    n = input("Dame n: ")
    if(type(n)!=int):
        continue
    if(is_power(n)):
        continue
    if((n>0) & (type(n)==int)):
        break

ttotal0=time.clock()
log_n = math.log(n)
print "log(n):", log_n
log2_n = math.pow(log_n,2)
print "log2(n): ", log2_n
tlog2n = math.trunc(log2_n)
q0 = tlog2n+1
q = q0
print "\nq: ",q
flag=True

t0=time.clock()
while(1):
    flag=True
    for j in range(1,q0,1):
        res = modulo(n,j,q)
```

```
            if(res==1):
                q=q+1
                flag=False
                break
        if(flag):
            r=q
            break

t1=time.clock()

print "\nLa r encontrada es: ",r,"\nEl tiempo para determinar r es: ", t1-t0," segundos."

primo=False
flag=True
a=1
t2=time.clock()
while(flag):
    mcd=my_gcd(n, a)
    if(mcd>1):
        print "\nEl numero ",n," es COMPUESTO"
        flag=False
    else:
        if(mcd==1):
            a=a+1
    if(a>=r):
        print "\nEl numero ",n," es candidato a ser PRIMO"
        primo=True
        break
t3=time.clock()

print "El tiempo empleado es: ", t3-t2," segundos.\n"

t4=time.clock()
if(primo):
    n1=n
    t0=time.clock()
    parar=mul(sqrt(r), (log(n)))
    A=2*mul((n**2),r)    #Constante A auxiliar
    print "\nA: ", A

    g=(A**r)-1  #x^r - 1 evaluado en A

    k=n%r
    flag=True

    for a in range (1, parar, 1):
        f=A+a                #x + a evaluado en A
        d=n1%2               #delta
        pow0=pow(f,d)        #f^d
        pow2=f               #inicializo pow2 = f

        n1=long((n1-d)/2)
```

```
        #calculo de f^n(mod A^r - 1, n)
        while(n1>0):
            pow2 = pow(pow2,2,g)     #pow2 = (pow2^2) (mod A^r - 1)
            p3=pow2
            j=0
            square=0
            #Calculo de pow2 (mod n)
            while(p3>0):
                #Calculo de los coeficientes de  pow2
                C1=p3%A
                p3=long((p3 - C1)/A)
                #Calculo de los coeficientes de pow2 (mod n)
                C2=C1%n
                #Recalculo el pow2 con los coeficientes obtenidos (ya con mod n)
                square=square + mul(C2,(A**j))
                j=j+1
            pow2=square      #pow2 es f^2i(mod A^r-1, n)
            d=n1%2
            pow0= mul(pow0, pow(pow2,d))
            pow0 = pow0%g

            p3=pow0
            l=0
            square=0
            while(p3>0):
                C1=p3%A
                p3=long((p3 - C1)/A)
                C2=C1%n
                square=square + mul(C2,(A**l))
                l=l+1

            pow0=square
            n1=long((n1-d)/2)


        xn_a=(A**k)+a

        if((square-xn_a)!=0):
            print"\nEl numero ",n," es COMPUESTO."
            flag=False
            break
        n1=n

    t5=time.clock()
    if(flag):
        print"\nEl numero ",n," es PRIMO"
    print "Tiempo empleado (segundos): ",t5-t4
    print "Tiempo empleado (minutos): ", (t5-t4)/60

ttotal1=time.clock()
print "\n\nEl tiempo total (en minutos) del algoritmo fue: ", (ttotal1-ttotal0)/60
```

Apéndice B

Capturas de pantalla de ejecución con números primos

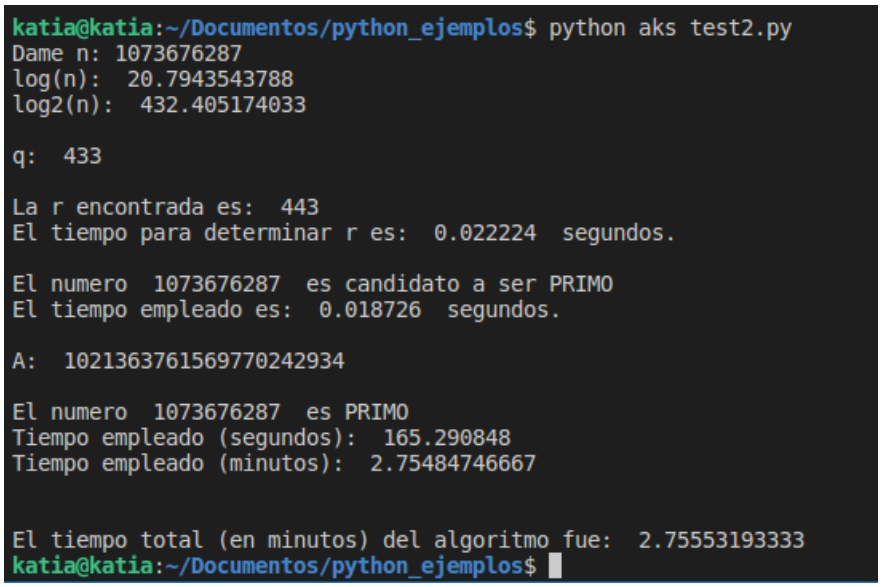

Figura B.1: Ejecución con número primo 1073676287

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 1088888881
log(n):  20.808423638
log2(n):  432.9904943

q:  433

La r encontrada es:  449
El tiempo para determinar r es:  0.028351  segundos.

El numero  1088888881  es candidato a ser PRIMO
El tiempo empleado es:  0.022579  segundos.

A:  1064739737658558080578

El numero  1088888881  es PRIMO
Tiempo empleado (segundos):  158.084132
Tiempo empleado (minutos):  2.63473553333


El tiempo total (en minutos) del algoritmo fue:  2.63558703333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.2: Ejecución con número primo 1088888881

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 1095912793
log(n):  20.8148534539
log2(n):  433.258124306

q:  434

La r encontrada es:  439
El tiempo para determinar r es:  0.012653  segundos.

El numero  1095912793  es candidato a ser PRIMO
El tiempo empleado es:  0.015903  segundos.

A:  1054499818178011425422

El numero  1095912793  es PRIMO
Tiempo empleado (segundos):  165.135638
Tiempo empleado (minutos):  2.75226063333


El tiempo total (en minutos) del algoritmo fue:  2.75273941667
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
```

Figura B.3: Ejecución con número primo 1095912793

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 2599291451
log(n):  21.678504726
log2(n):  469.957567155

q:  470

La r encontrada es:  491
El tiempo para determinar r es:  0.03408  segundos.

El numero  2599291451  es candidato a ser PRIMO
El tiempo empleado es:  0.021287  segundos.

A:  6634702358391335063782

El numero  2599291451  es PRIMO
Tiempo empleado (segundos):  247.8672
Tiempo empleado (minutos):  4.13112


El tiempo total (en minutos) del algoritmo fue:  4.13204555
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.4: Ejecución con número primo 2599291451

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 3211891519
log(n):  21.8901258588
log2(n):  479.177610114

q:  480

La r encontrada es:  503
El tiempo para determinar r es:  0.0269  segundos.

El numero  3211891519  es candidato a ser PRIMO
El tiempo empleado es:  0.022378  segundos.

A:  10378144612603072125166

El numero  3211891519  es PRIMO
Tiempo empleado (segundos):  276.415612
Tiempo empleado (minutos):  4.60692686667


El tiempo total (en minutos) del algoritmo fue:  4.6077504
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.5: Ejecución con número primo 3211891519

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 4076863487
log(n):  22.1285937765
log2(n):  489.674662527

q:  490

La r encontrada es:  509
El tiempo para determinar r es:  0.037996  segundos.

El numero  4076863487  es candidato a ser PRIMO
El tiempo empleado es:  0.017094  segundos.

A:  16919990577683207554042

El numero  4076863487  es PRIMO
Tiempo empleado (segundos):  293.036886
Tiempo empleado (minutos):  4.8839481


El tiempo total (en minutos) del algoritmo fue:  4.88486945
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.6: Ejecución con número primo 4076863487

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 8192454631
log(n):  22.8264794007
log2(n):  521.048161828

q:  522

La r encontrada es:  523
El tiempo para determinar r es:  0.018752  segundos.

El numero  8192454631  es candidato a ser PRIMO
El tiempo empleado es:  0.02653  segundos.

A:  70203663273519040084406

El numero  8192454631  es PRIMO
Tiempo empleado (segundos):  347.075323
Tiempo empleado (minutos):  5.78458871667


El tiempo total (en minutos) del algoritmo fue:  5.78534568333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.7: Ejecución con número primo 8192454631

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 10000000019
log(n):  23.0258509318
log2(n):  530.189811135

q:  531

La r encontrada es:  557
El tiempo para determinar r es:  0.036822  segundos.

El numero  10000000019  es candidato a ser PRIMO
El tiempo empleado es:  0.019512  segundos.

A:  111400000423320000402154

El numero  10000000019  es PRIMO
Tiempo empleado (segundos):  415.77851
Tiempo empleado (minutos):  6.92964183333


El tiempo total (en minutos) del algoritmo fue:  6.93058273333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.8: Ejecución con número primo 10000000019

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 15984784979
log(n):  23.4949031679
log2(n):  552.010474871

q:  553

La r encontrada es:  577
El tiempo para determinar r es:  0.059182  segundos.

El numero  15984784979  es candidato a ser PRIMO
El tiempo empleado es:  0.024856  segundos.

A:  294862406851893091128914

El numero  15984784979  es PRIMO
Tiempo empleado (segundos):  488.504108
Tiempo empleado (minutos):  8.14173513333


El tiempo total (en minutos) del algoritmo fue:  8.14313816667
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.9: Ejecución con número primo 15984784979

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 24571243009
log(n):  23.9248426127
log2(n):  572.398094043

q:  573

La r encontrada es:  607
El tiempo para determinar r es:  0.071389  segundos.

El numero  24571243009  es candidato a ser PRIMO
El tiempo empleado es:  0.021569  segundos.

A:  732947623370900288134334

El numero  24571243009  es PRIMO
Tiempo empleado (segundos):  596.685114
Tiempo empleado (minutos):  9.9447519


El tiempo total (en minutos) del algoritmo fue:  9.94630351667
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.10: Ejecución con número primo 24571243009

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 32212254719
log(n):  24.1956127984
log2(n):  585.427678692

q:  586

La r encontrada es:  617
El tiempo para determinar r es:  0.104997  segundos.

El numero  32212254719  es candidato a ser PRIMO
El tiempo empleado es:  0.025841  segundos.

A:  1280434622936864406897874

El numero  32212254719  es PRIMO
Tiempo empleado (segundos):  661.919574
Tiempo empleado (minutos):  11.0319929


El tiempo total (en minutos) del algoritmo fue:  11.0341768333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.11: Ejecución con número primo 32212254719

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 50006393431
log(n):  24.6354167028
log2(n):  606.903756122

q:  607

La r encontrada es:  617
El tiempo para determinar r es:  0.061862  segundos.

El numero  50006393431  es candidato a ser PRIMO
El tiempo empleado es:  0.023565  segundos.

A:  308578899982633458047307 4

El numero  50006393431  es PRIMO
Tiempo empleado (segundos):  690.063885
Tiempo empleado (minutos):  11.50106475


El tiempo total (en minutos) del algoritmo fue:  11.5024914
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.12: Ejecución con número primo 50006393431

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 99999199999
log(n):  25.3284280229
log2(n):  641.529266111

q:  642

La r encontrada es:  643
El tiempo para determinar r es:  0.031983  segundos.

El numero  99999199999  es candidato a ser PRIMO
El tiempo empleado es:  0.023304  segundos.

A:  1285979424056584205760 1286

El numero  99999199999  es PRIMO
Tiempo empleado (segundos):  865.801898
Tiempo empleado (minutos):  14.4300316333


El tiempo total (en minutos) del algoritmo fue:  14.4309558167
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.13: Ejecución con número primo 99999199999

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 100123456789
log(n):  25.3296698294
log2(n):  641.592173665

q:  642

La r encontrada es:  647
El tiempo para determinar r es:  0.072516  segundos.

El numero  100123456789  es candidato a ser PRIMO
El tiempo empleado es:  0.03316  segundos.

A:  1297197033959610274653417 4

El numero  100123456789  es PRIMO
Tiempo empleado (segundos):  856.997555
Tiempo empleado (minutos):  14.2832925833


El tiempo total (en minutos) del algoritmo fue:  14.2850574
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
```

Figura B.14: Ejecución con número primo 100123456789

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 260389232731
log(n):  26.2854433975
log2(n):  690.924534601

q:  691

La r encontrada es:  751
El tiempo para determinar r es:  0.222343  segundos.

El numero  260389232731  es candidato a ser PRIMO
El tiempo empleado es:  0.031557  segundos.

A:  10183943388840280034097822 2

El numero  260389232731  es PRIMO
Tiempo empleado (segundos):  1362.114236
Tiempo empleado (minutos):  22.7019039333


El tiempo total (en minutos) del algoritmo fue:  22.7061393333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.15: Ejecución con número primo 260389232731

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 393142151459
log(n):  26.697437092
log2(n):  712.753147279

q:  713

La r encontrada es:  743
El tiempo para determinar r es:  0.157775  segundos.

El numero  393142151459  es candidato a ser PRIMO
El tiempo empleado es:  0.02908  segundos.

A:  22967727636316358560l419966

El numero  393142151459  es PRIMO
Tiempo empleado (segundos):  1504.817752
Tiempo empleado (minutos):  25.0802958667


El tiempo total (en minutos) del algoritmo fue:  25.0834130333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.16: Ejecución con número primo 393142151459

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 555555577777
log(n):  27.043234491
log2(n):  731.336531737

q:  732

La r encontrada es:  751
El tiempo para determinar r es:  0.080962  segundos.

El numero  555555577777  es candidato a ser PRIMO
El tiempo empleado es:  0.031027  segundos.

A:  463580283998702716985116958

El numero  555555577777  es PRIMO
Tiempo empleado (segundos):  1667.647558
Tiempo empleado (minutos):  27.7941259667


El tiempo total (en minutos) del algoritmo fue:  27.7959947
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
```

Figura B.17: Ejecución con número primo 555555577777

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 753352617167
log(n):  27.3477992382
log2(n):  747.902123174

q:  748

La r encontrada es:  821
El tiempo para determinar r es:  0.193327  segundos.

El numero  753352617167  es candidato a ser PRIMO
El tiempo empleado es:  0.032  segundos.

A:  931900952231069016419869738

El numero  753352617167  es PRIMO
Tiempo empleado (segundos):  2137.66746
Tiempo empleado (minutos):  35.627791


El tiempo total (en minutos) del algoritmo fue:  35.6315485333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.18: Ejecución con número primo 753352617167

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 999999000001
log(n):  27.6310201159
log2(n):  763.473272647

q:  764

La r encontrada es:  827
El tiempo para determinar r es:  0.152846  segundos.

El numero  999999000001  es candidato a ser PRIMO
El tiempo empleado es:  0.027248  segundos.

A:  1653996692004961996692001654

El numero  999999000001  es PRIMO
Tiempo empleado (segundos):  1860.96099
Tiempo empleado (minutos):  31.0160165


El tiempo total (en minutos) del algoritmo fue:  31.0190202833
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.19: Ejecución con número primo 999999000001

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 2999999899999
log(n):  28.7296333713
log2(n):  825.391833647

q:  826

La r encontrada es:  839
El tiempo para determinar r es:  0.080767  segundos.

El numero  2999999899999  es candidato a ser PRIMO
El tiempo empleado es:  0.031367  segundos.

A:  151019989931899487803356016 78

El numero  2999999899999  es PRIMO
Tiempo empleado (segundos):  2203.248384
Tiempo empleado (minutos):  36.7208064


El tiempo total (en minutos) del algoritmo fue:  36.7226778167
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.20: Ejecución con número primo 2999999899999

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 3664533202453
log(n):  28.9297220768
log2(n):  836.928819443

q:  837

La r encontrada es:  839
El tiempo para determinar r es:  0.064817  segundos.

El numero  3664533202453  es candidato a ser PRIMO
El tiempo empleado es:  0.031917  segundos.

A:  225335324271753781273944767 02

El numero  3664533202453  es PRIMO
Tiempo empleado (segundos):  2265.107049
Tiempo empleado (minutos):  37.75178415


El tiempo total (en minutos) del algoritmo fue:  37.7533985333
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.21: Ejecución con número primo 3664533202453

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 6056529316217
log(n):  29.4321580318
log2(n):  866.251926412

q:  867

La r encontrada es:  877
El tiempo para determinar r es:  0.092832  segundos.

El numero  6056529316217  es candidato a ser PRIMO
El tiempo empleado es:  0.037392  segundos.

A:  643394340662757166099011701 06

El numero  6056529316217  es PRIMO
Tiempo empleado (segundos):  2716.61104
Tiempo empleado (minutos):  45.2768506667


El tiempo total (en minutos) del algoritmo fue:  45.2790234
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.22: Ejecución con número primo 6056529316217

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 7142857142857
log(n):  29.5971339723
log2(n):  875.990339374

q:  876

La r encontrada es:  877
El tiempo para determinar r es:  0.109271  segundos.

El numero  7142857142857  es candidato a ser PRIMO
El tiempo empleado es:  0.038996  segundos.

A:  894897959183637673469387755 46

El numero  7142857142857  es PRIMO
Tiempo empleado (segundos):  2721.543124
Tiempo empleado (minutos):  45.3590520667


El tiempo total (en minutos) del algoritmo fue:  45.36152615
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.23: Ejecución con número primo 7142857142857

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 8284590452353
log(n):  29.7454183332
log2(n):  884.789911815

q:  885

La r encontrada es:  911
El tiempo para determinar r es:  0.125667  segundos.

El numero  8284590452353  es candidato a ser PRIMO
El tiempo empleado es:  0.030956  segundos.

A:  125051947790984079967417101598

El numero  8284590452353  es PRIMO
Tiempo empleado (segundos):  3731.923952
Tiempo empleado (minutos):  62.1987325333


El tiempo total (en minutos) del algoritmo fue:  62.2013448167
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.24: Ejecución con número primo 8284590452353

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 9999999998987
log(n):  29.9336062088
log2(n):  896.020780665

q:  897

La r encontrada es:  919
El tiempo para determinar r es:  0.115581  segundos.

El numero  9999999998987  es candidato a ser PRIMO
El tiempo empleado es:  0.034333  segundos.

A:  183799999962762120001886098622

El numero  9999999998987  es PRIMO
Tiempo empleado (segundos):  3981.883277
Tiempo empleado (minutos):  66.3647212833


El tiempo total (en minutos) del algoritmo fue:  66.3672221667
katia@katia:~/Documentos/python_ejemplos$
```

Figura B.25: Ejecución con número primo 9999999998987

Apéndice C

Capturas de pantalla de ejecución con números compuestos

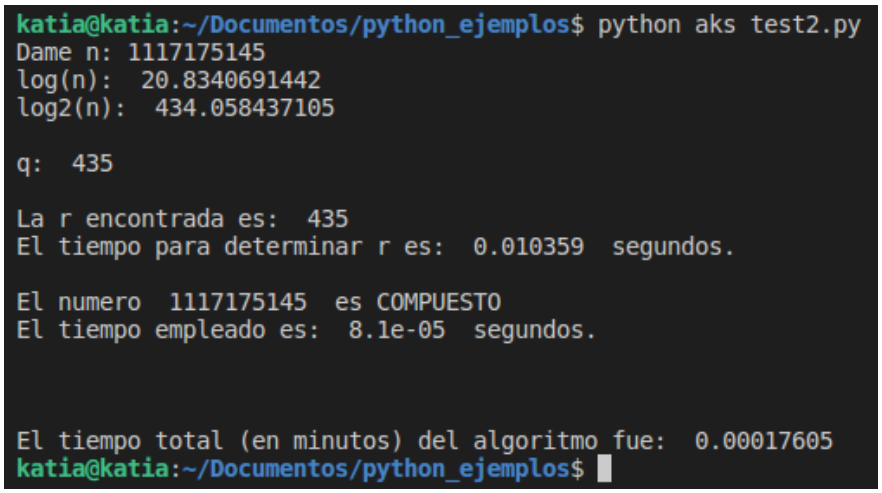

Figura C.1: Ejecución con número compuesto 1117175145

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 1123456987
log(n):  20.839676364
log2(n):  434.292110958

q:  435

La r encontrada es:  443
El tiempo para determinar r es:  0.015893  segundos.

El numero  1123456987  es candidato a ser PRIMO
El tiempo empleado es:  0.015605  segundos.


A:  111826986305225869773 4

El numero  1123456987  es COMPUESTO.
Tiempo empleado (segundos):  0.348301
Tiempo empleado (minutos):  0.00580501666667


El tiempo total (en minutos) del algoritmo fue:  0.00633306666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.2: Ejecución con número compuesto 1123456987

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 1136972771
log(n):  20.8516351033
log2(n):  434.790686482

q:  435

La r encontrada es:  439
El tiempo para determinar r es:  0.016159  segundos.

El numero  1136972771  es COMPUESTO
El tiempo empleado es:  0.000346  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000277383333333
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.3: Ejecución con número compuesto 1136972771

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 2754425310
log(n):  21.7364746589
log2(n):  472.474330596

q:  473

La r encontrada es:  474
El tiempo para determinar r es:  0.033542  segundos.

El numero  2754425310  es COMPUESTO
El tiempo empleado es:  8.7e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.0005646
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.4: Ejecución con número compuesto 2754425310

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 3215031751
log(n):  21.8911030705
log2(n):  479.220393642

q:  480

La r encontrada es:  499
El tiempo para determinar r es:  0.05183  segundos.

El numero  3215031751  es COMPUESTO
El tiempo empleado es:  0.007041  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000984916666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.5: Ejecución con número compuesto 3215031751

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 4294967295
log(n):  22.1807097777
log2(n):  491.983886242

q:  492

La r encontrada es:  492
El tiempo para determinar r es:  0.017248  segundos.

El numero  4294967295  es COMPUESTO
El tiempo empleado es:  0.000121  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000293033333333
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.6: Ejecución con número compuesto 4294967295

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 10123456897
log(n):  23.0381210331
log2(n):  530.755020735

q:  531

La r encontrada es:  557
El tiempo para determinar r es:  0.065977  segundos.

El numero  10123456897  es COMPUESTO
El tiempo empleado es:  0.0199  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00143483333333
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.7: Ejecución con número compuesto 10123456897

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 17392546081
log(n):  23.5793075652
log2(n):  555.983745256

q:  556

La r encontrada es:  557
El tiempo para determinar r es:  0.048122  segundos.

El numero  17392546081  es COMPUESTO
El tiempo empleado es:  0.001412  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000831116666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.8: Ejecución con número compuesto 17392546081

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 25505418241
log(n):  23.9621567466
log2(n):  574.184955948

q:  575

La r encontrada es:  617
El tiempo para determinar r es:  0.113694  segundos.

El numero  25505418241  es candidato a ser PRIMO
El tiempo empleado es:  0.027475  segundos.


A:  802749527806046049055954

El numero  25505418241  es COMPUESTO.
Tiempo empleado (segundos):  0.903855
Tiempo empleado (minutos):  0.01506425


El tiempo total (en minutos) del algoritmo fue:  0.01742125
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.9: Ejecución con número compuesto 25505418241

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 32149366346
log(n):  24.1936585781
log2(n):  585.333115395

q:  586

La r encontrada es:  586
El tiempo para determinar r es:  0.038376  segundos.

El numero  32149366346  es COMPUESTO
El tiempo empleado es:  7.3e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000643383333333
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.10: Ejecución con número compuesto 32149366346

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 54047322253
log(n):  24.7131258378
log2(n):  610.738588673

q:  611

La r encontrada es:  645
El tiempo para determinar r es:  0.082521  segundos.

El numero  54047322253  es COMPUESTO
El tiempo empleado es:  0.000857  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00139198333333
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.11: Ejecución con número compuesto 54047322253

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 97524222465
log(n):  25.3033666196
log2(n):  640.260362288

q:  641

La r encontrada es:  642
El tiempo para determinar r es:  0.032111  segundos.

El numero  97524222465  es COMPUESTO
El tiempo empleado es:  6.8e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000538433333333
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.12: Ejecución con número compuesto 97524222465

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 111111111111
log(n):  25.4337965386
log2(n):  646.878006366

q:  647

La r encontrada es:  648
El tiempo para determinar r es:  0.067991  segundos.

El numero  111111111111  es COMPUESTO
El tiempo empleado es:  6.7e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00113695
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.13: Ejecución con número compuesto 111111111111

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 123455554321
log(n):  25.5391470442
log2(n):  652.248031745

q:  653

La r encontrada es:  656
El tiempo para determinar r es:  0.03906  segundos.

El numero  123455554321  es COMPUESTO
El tiempo empleado es:  0.000302  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.0006584
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.14: Ejecución con número compuesto 123455554321

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 334826628433
log(n):  26.5368787077
log2(n):  704.205931549

q:  705

La r encontrada es:  706
El tiempo para determinar r es:  0.039803  segundos.

El numero  334826628433  es COMPUESTO
El tiempo empleado es:  0.000231  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00066935
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.15: Ejecución con número compuesto 334826628433

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 743008370701
log(n):  27.3339731477
log2(n):  747.146088038

q:  748

La r encontrada es:  769
El tiempo para determinar r es:  0.067492  segundos.

El numero  743008370701  es COMPUESTO
El tiempo empleado es:  0.007012  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00124386666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.16: Ejecución con número compuesto 743008370701

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 774630686237
log(n):  27.3756522188
log2(n):  749.426334407

q:  750

La r encontrada es:  761
El tiempo para determinar r es:  0.069387  segundos.

El numero  774630686237  es COMPUESTO
El tiempo empleado es:  0.010499  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00133395
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.17: Ejecución con número compuesto 774630686237

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 799980626860
log(n):  27.4078533479
log2(n):  751.19042514

q:  752

La r encontrada es:  752
El tiempo para determinar r es:  0.053869  segundos.

El numero  799980626860  es COMPUESTO
El tiempo empleado es:  7.3e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.000901266666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.18: Ejecución con número compuesto 799980626860

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 2935561623745
log(n):  28.7079199048
log2(n):  824.144665262

q:  825

La r encontrada es:  825
El tiempo para determinar r es:  0.061285  segundos.

El numero  2935561623745  es COMPUESTO
El tiempo empleado es:  0.000142  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00102581666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.19: Ejecución con número compuesto 2935561623745

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 3465253618401
log(n):  28.8738069402
log2(n):  833.696727222

q:  834

La r encontrada es:  834
El tiempo para determinar r es:  0.125435  segundos.

El numero  3465253618401  es COMPUESTO
El tiempo empleado es:  8.9e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.0020959
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.20: Ejecución con número compuesto 3465253618401

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 6252893229398
log(n):  29.4640653893
log2(n):  868.131149263

q:  869

La r encontrada es:  870
El tiempo para determinar r es:  0.080457  segundos.

El numero  6252893229398  es COMPUESTO
El tiempo empleado es:  5.6e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.0013447
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.21: Ejecución con número compuesto 6252893229398

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 7625597484960
log(n):  29.662531794
log2(n):  879.865792432

q:  880

La r encontrada es:  880
El tiempo para determinar r es:  0.067388  segundos.

El numero  7625597484960  es COMPUESTO
El tiempo empleado es:  4.9e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00112601666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.22: Ejecución con número compuesto 7625597484960

```
katia@katia:~/Documentos/python_ejemplos$ python aks test2.py
Dame n: 8284590452382
log(n):  29.7454183332
log2(n):  884.789911815

q:  885

La r encontrada es:  885
El tiempo para determinar r es:  0.071122  segundos.

El numero  8284590452382  es COMPUESTO
El tiempo empleado es:  9e-05  segundos.



El tiempo total (en minutos) del algoritmo fue:  0.00118891666667
katia@katia:~/Documentos/python_ejemplos$
```

Figura C.23: Ejecución con número compuesto 8284590452382

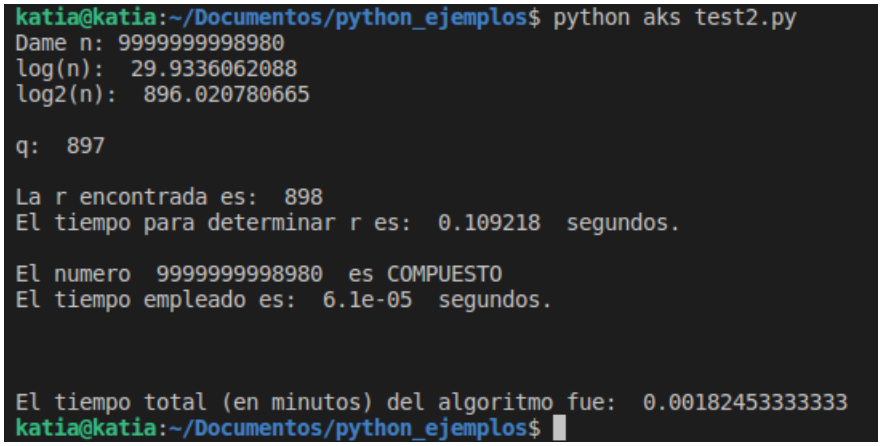

Figura C.24: Ejecución con número compuesto 9999999998980

Bibliografía

[1] Euclid, *The Elements of Euclid, With Dissertations (J. Williamson (translator))*. Clarendon Press Oxford, 1782.

[2] C. Gauss, *Disquisitiones Arithemeticae (Clark, A. (translator)*, second, corrected edition ed. New York: Springer Verlag, 1986.

[3] N. Koblitz, *A Course in Number Theory and Cryptography*, 2.ª ed. Springer, 1994.

[4] A. Manindra, K. Neeraj, y S. Nitin, "Primes is in p," *Annals of Mathematics*, vol. 160, pp. 781–793, 2004.

[5] A. H. Vázquez, "Diseño, implementación y comparación de los métodos de encriptación RSA en aritmética modular y Massey-Omura en curvas elípticas," Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, septiembre 2016.

[6] A. A. Gaona, "Implementación en software-hardware de aritmética sobre campos finitos binarios $\mathbb{F}_{2^m}$ en curvas elípticas para aplicaciones criptográficas de llave pública," Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, abril 2012.

[7] Adleman, Pomerance, y Rumely, "On distinguishing prime numbers from composite numbers," *Annals of Mathematics*, vol. 117, pp. 173–206, 1983.

[8] M. O. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory*, vol. 12, pp. 128–138, 1980.

[9] "Implementation of the aks algorithm," http://yves.gallot.pagesperso-orange.fr/src/aks_gmp.html, consultada: 2017-11-29.

[10] J. Buchmann, *Introduction to cryptography*, ser. Undergraduate Texts in Mathematics. Springer New York, 2004.

[11] S. Yan, *Number theory for computing*. New York: Springer Verlag, 2002.

[12] "Eff cooperative computing awards," https://www.eff.org/awards/coop, 2017.

[13] C. Alina y M. Ram, *An Introduction to Sieve Methods and Their Applications*, ser. London Mathematical Society St. Cambridge University Press, 2005. [En línea]. Disponible: https://books.google.com.mx/books?id=1swo9Yf3d2YC

[14] G. Miller., "Riemannn's hypothesis and test for primality," *J. Comput. Sys, Sci.*, vol. 13, pp. 300–317, 1976.

[15] I. Vinogradov, *An introduction to the theory of numbers*. London & New York: Pergamon Press, 1955.

[16] A. Granville., "It is easy to determine whether a given integer is prime," *Bulletin (New Series) of the American Mathematical Society*, vol. 42, pp. 3–38, 2004.

[17] "The nth prime page," https://primes.utm.edu/nthprime/, consultada: 2017-11-29.