El número exacto de primos menores que x

Violeta Gracia Aguilar

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Directores del trabajo:
Julio Bernués, José Carlos Ciria

Junio de 2019

Summary

Prime numbers have been studied since the Greek ages. The seventh book of Euclid's *Elementa* (Euclid 300 B.C.), includes the distinction between prime and composite integers and in the ninth book the existence of infinite prime numbers is stated.

The aim of this work is to study the distribution of prime numbers. First of all, we will focus on the methods that provide the value of $\pi(n)$, the number of primes not exceeding n, from the Greeks to the beginning of the 20th century. Those methods were initiated by Eratosthenes (276 B.C.-197 B.C.), developed by Legendre in 1830, simplified by Meissel in 1870 and improved in 1949 by Lehmer. In addition, we implement an algorithm to compute $\pi(n)$ for each method studied. At the same time, the asymptotic behaviour of $\pi(n)$ has been of historical interest. In particular we are going to study the celebrated result due to Chebyshev, with real methods and real functions, as well as the Bertrand's Postulate.

The oldest procedure for calculating the value of $\pi(n)$ dates back to Eratosthenes, commonly known as the sieve of Eratosthenes, where all primes not exceeding n are explicitly found and then counted. This is essentially a method of exclusion, by which all composite numbers are successively erased from the natural numbers, leaving the primes. Despite the laborious procedure, this sieve has been for a very long time the practical way to compute $\pi(n)$.

In 1830 Legendre gave a quantitative form of the sieve of Eratosthenes, the simplest formula, but unfortunately labour-consuming. The idea is, having the prime numbers in $[2,\ \sqrt{n}]$, delete all their multiples not exceeding n thus obtaining the amount of primes in $(\sqrt{n},\ n]$. In doing so we get the value of $\pi(n)$, however *we renounce to know all the prime numbers*. Legendre's formula is not so convenient for computation, for this reason, in 1870 Meissel developed an efficient modification of it and in 1959 Lehmer improved Meissel's method.

Meissel's formula and its generalizations, in particular Lehmer's formula, are based on having the first a prime numbers, $\{p_1,\ p_2,\ \ldots,\ p_a\}$, get the amount of the remaining primes not exceeding n.
Meissel provides $\pi(n)$ from the first $a = \pi(\sqrt[3]{n})$ primes, and Lehmer from $a = \pi(\sqrt[4]{n})$. As we can notice, Legendre's method is the particular case where $a = \pi(\sqrt{n})$.

In the foregoing pages, Chapter 2 and 3, we are going to present the preceding four methods and implement their algorithms in Java. Since for each method there are different possible algorithms, our objective is develop the most efficient one among them. That is, the one that uses less resources, essentially computation-time and memory.

When possible, we will present directly the implemented algorithm and study its complexity. In other cases we will introduce an intuitive version of the algorithm and discuss some improvements to optimise its computational cost. This procedure will led us to the final algorithm.

We will prove that the four proposed algorithms (i.e. the sieve of Eratosthenes, Legendre, Meissel and Lehmer) are increasingly efficient. This result will be confirmed by numerical computations which will allow us to quantify the improvement. In particular, the dependence between time and n fits lineal

models, and is compatible with the predicted formulas.

Other methods have been subsequently discovered, like Mapes in 1963, and recently, in 1985, the Lagarias, Miller and Odlyzko method.

As we have seen previously, the interest of counting prime numbers reaches dates back to the Greek ages, but it was in the 18th when it started to be an interesing question. The apparently random distribution of the prime numbers makes the function $\pi(n)$ difficult to determinate.

Euler was the first to intuit, in 1762, that $\pi(n)$ approximates to $\frac{n}{\ln n}$. In the following years other outcomes related were developed by Gauss, 1791, and Legendre, 1798. Both tried to prove that

$$\pi(n) \sim \frac{n}{\ln n},\ n \to \infty, \tag{1}$$

unsuccessfully. The first result concerning the asymptotic behaviour of $\pi(n)$ was obtained by Chebyshev:

$$\lim_{n\to\infty} \frac{\pi(n)\ln n}{n} = 1,$$

if this limit exists. Chebyshev made an important step in the proof of (1) but could not prove the existence of this limit. In 1896 this hypothesis was finally proved as the commonly known The Prime Number Theorem.

In the last chapter we will study the Chebyshev's functions and, making use of his methods, we will study the asymptotic behaviour of $\pi(n)$ and get sharp upper and lower bounds of $\pi(n)$. With elementary methods we can obtain concrete values of the bounds. In addition, using the Chebyshev's functions we will deduce the Bertrnad Postulate too, "There exists a prime number between x and $2x\ \forall\ x > 3$."

Also in the proof of Chebyshev's Theorem we use a simpler method which in turn lead us to compute the corresponding numerical constants.

As we shall see, part of this work contains original material, for instance, in Appendix D, we will calculate a table with different values of $\pi(n)$, using the algorithms that we have implemented. Moreover Appendix F includes all these algorithms developed in the work.

Índice general

Summary III

1. Introducción 1
1.1. Resultados previos . . . 1
1.2. Marco histórico . . . 1

2. La cantidad de primos hasta n 5
2.0.1. Legendre . . . 5
2.0.2. Meissel-Lehmer . . . 6

3. Algoritmos 9
3.1. Criba de Eratóstenes . . . 9
3.1.1. Descripción del algoritmo . . . 9
3.1.2. Complejidad . . . 9
3.1.3. Mejoras . . . 10
3.2. Suma de Legendre, $\phi(n,a)$ . . . 12
3.2.1. Descripción del algoritmo . . . 12
3.2.2. Complejidad . . . 13
3.2.3. Mejoras . . . 14
3.3. Legendre . . . 14
3.3.1. Complejidad . . . 14
3.4. $P_2(n,a)$ . . . 15
3.4.1. Descripción del algoritmo . . . 15
3.4.2. Complejidad . . . 15
3.5. $P_3(n,a)$ . . . 16
3.5.1. Descripción del algoritmo . . . 16
3.5.2. Complejidad . . . 17
3.6. Meissel . . . 18
3.6.1. Complejidad . . . 18
3.7. Lehmer . . . 18
3.7.1. Complejidad . . . 18
3.8. Conclusiones y resultados . . . 19

4. Distribución de los números primos. Métodos de Chebyshev 21
4.0.1. Definición funciones de Chebyshev . . . 21
4.0.2. Acotaciones asintóticas para las funciones de Chebyshev . . . 22
4.0.3. Acotaciones asintóticas históricas . . . 24
4.0.4. Teorema de Chebyshev . . . 25

Bibliografía 27

Anexos 28

A. Nociones de complejidad **29**

B. Cantidad de primos menores que n. Anexo del Capítulo 2 **31**

C. Algoritmos. Anexo del Capítulo 3 **33**

D. Tabla con valores de $\pi(n)$ **35**

E. Distribución de los números primos. Métodos de Chebyshev. Anexo del Capítulo 4 **37**
E.1. Postulado de Bertrand 37
E.2. Teorema de Chebyshev 38

F. Programas **41**
F.1. *cribaEratóstenes* 41
F.2. *cadenaPrimos* 42
F.3. *subCribaIntervalo* 43
F.4. *cribaIntervalo* 44
F.5. *piEratóstenes* 45
F.6. *sumaPhi* 45
F.7. *piLegendre* 47
F.8. $P_2(n,a)$ 48
F.9. $P_3(n,a)$ 49
F.10. *piMeissel* 50
F.11. *piLehmer* 50
F.12. *postuladoBertrand* 51

Capítulo 1

Introducción

Para la elaboración de este capítulo hemos consultado [1], [3] y [7].

1.1. Resultados previos

A lo largo de este trabajo utilizaremos la letra p para caracterizar a un primo genérico y emplearemos p_i para denotar al i-ésimo primo: $p_1 = 2,\ p_2 = 3,\ p_3 = 5,\ \ldots$

Definición 1.1. Dado un número natural n, la función $\pi(n)$ es la cantidad de primos menores o iguales que n:

$$\pi(n) = |\{\ p\ \mid\ \text{p primo, } p \leq n\ \}|$$

Donde aquí y a lo largo del trabajo, $|\cdot|$ indica el cardinal de un conjunto.
Así, por ejemplo, $\pi(2) = 1,\ \pi(3) = 2,\ \pi(4) = 2,\ \pi(5) = 3,\ \ldots$ Es claro que $\pi(n) < n$.

Enunciamos a continuación ciertos resultados que van a ser utilizados a lo largo de todo el trabajo.

Teorema 1.1 (Teorema fundamental de la Aritmética, TFA)**.**
Cada entero $n > 1$ se puede representar como un producto de factores primos de forma única, salvo en el orden de los factores.

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_r^{\alpha_r} = \prod_{i=1}^{r} p_i^{\alpha_i}.$$

Observación 1.2. *Sea un entero $n > 1$, $\forall\ b \leq n$, si b no es divisible por ningún primo $\leq \sqrt{n}\ \Rightarrow\ b$ es un número primo en $(\sqrt{n},\ n]$.*

Demostración. Si $b \leq n$ es un número compuesto tal que $b = rq$ con $\sqrt{n} > r,\ q\ \Rightarrow\ b = rq > n$; llegamos a contradicción. □

1.2. Marco histórico

El primer resultado tratando la distribución de los números primos aparece en la Proposición 20 del libro 9 de los *Elementos* de Euclides donde se demuestra la existencia de infinitos números primos; de modo que $\pi(n) \to \infty$ cuando $n \to \infty$.

Teorema 1.3 (Euclides)**.** *Hay infinitos números primos.*

En 1737, Euler desmuestra que la suma de los inversos de los primos diverge. De donde también se obtiene la existencia de infinitos números primos.

Teorema 1.4 (Euler)**.** *La serie*

$$\sum_p \frac{1}{p}$$

donde p recorre todos los primos, es divergente.

A partir del siglo XVIII empezó a interesar $\pi(n)$ como función. Esta función es difícil de determinar ya que desconocemos la relación entre un primo y su siguiente. Históricamente, se han intentado obtener cotas superiores e inferiores de $\pi(n)$ y simultáneamente, una fórmula que nos diera la cantidad exacta de primos menores que n.

Euler fue el primero en intuir que $\pi(n)$ se aproxima a $\frac{n}{\ln n}$, 1762/63. Años más tarde, en 1798, Legendre publicó en su obra *Essai sur la Théorie des nombres* que una buena aproximación para $\pi(n)$ venía dada por

$$\frac{n}{A\ln n + B}, \text{ con } A,\ B \text{ constantes.}$$

Simultáneamente en 1791 Gauss, con el estudio de las tablas de todos los primos hasta $3\cdot 10^6$, observó que $\pi(n)$ se aproximaba a $\frac{n}{\ln n}$ y, en una carta a su alumno Enke en 1849, propuso la siguiente aproximación

$$\pi(n) \sim \int_2^n \frac{dt}{\ln t} \sim \frac{n}{\ln n},\ n\to\infty, \tag{1.1}$$

aunque dicho resultado no fue publicado hasta 1863.

Ambos intentaron demostrar estas ideas sin éxito, hasta que en el año 1848, Chebyshev demostró que

$$\lim_{n\to\infty} \frac{\pi(n)\ln n}{n} = 1, \tag{1.2}$$

si este límite existe.

Chebyshev dio un paso importante para demostrar la *Ley asintótica de distribución de números primos* aunque no consiguió demostrar que dicho cociente tenía límite. Años más tarde, en 1896, esta conjetura fue finalmente demostrada de manera independiente por Hadamard y Vallée Poussin. Actualmente este resultado es conocido como el Teorema del Número Primo:

Teorema 1.5 (Teorema del número primo, TNP)**.**

$$\lim_{n\to\infty} \frac{\pi(n)\ln n}{n} = 1,\ \textit{es decir,}\ \pi(n) \sim \frac{n}{\ln n},\ n\to\infty. \tag{1.3}$$

En el Capítulo 4 de este trabajo estudiamos las denominadas funciones de Chebyshev y a partir de ellas obtendremos una acotación superior e inferior para $\pi(n)$. Por otro lado, también deduciremos otros resultados interesantes como el postulado de Bertrand.

Como se ha comentado al inicio desde que en la antigua Grecia se demostrara la existencia de infinitos números primos surgió interés por conocer la cantidad exacta de primos menores que n. El procedimiento más antiguo que localiza los números primos es desarrollado por Eratóstenes (276 A.C.-197 A.C., Director de la Biblioteca de Alejandría) y es conocido como la criba de Eratóstenes. Esencialmente es un método de exclusión donde los números compuestos son sucesivamente eliminados, obteniendo así los primos.

La idea es, empezando por $p_1 = 2$, tomar los primos menores o iguales que $\sqrt{n}$ y eliminar sus múltiplos menores o iguales que n : $\{\ p_i^2,\ p_i^2 + p_i,\ p_i^2 + 2\,p_i,\ \ldots \le n\ \}$, siendo p_i^2 el primer múltiplo no eliminado por los primos precedentes. Notar que se estudia hasta $p_i \le \sqrt{n}$, por la Observación 1.2.

Por ejemplo, los primos menores o iguales que 25 son $\{2, 3, 5, 7, 11, 13, 17, 19, 23\}$ y $\pi(25) = 9$, para ello eliminamos los múltiplos de los primos $\leq \sqrt{25} = 5$, veámoslo:

$$\mathbf{2}, \underline{\mathbf{3}}, 4, \underline{\mathbf{5}}, \| \ 6, \underline{7}, 8, 9, 10, \underline{11}, 12, \underline{13}, 14, 15, 16, \underline{17}, 18, \underline{19}, 20, 21, 22, \underline{23}, 24, 25.$$

En 1830 Legendre desarrolló una fórmula que expresaba dicha criba donde se daba el valor de $\pi(n)$ aunque se renunciaba a conocer los propios primos. Éste método requería de una gran cantidad de cuentas y para valores grandes de n era complejo, por ello, en 1870 el matemático Meissel presentó una mejora con la que se podía calcular más cómodamente $\pi(n)$ para valores grandes de n. Sin embargo, hasta la llegada de los ordenadores no se realizaron mejoras significativas. En 1959 Lehmer realiza una nueva mejora para el método de Meissel obteniendo el método conocido por Meissel-Lehmer. Comparando la complejidad obtenida y con experimentos numéricos se puede observar fácilmente el ahorro en tiempo y memoria que se produce entre métodos. Posteriormente se han ido desarrollando métodos más eficientes y, con ello, más complejos.

Estos cuatro métodos son estudiados en los Capítulos 2 y 3. En primer lugar desarrollaremos sus fórmulas y, seguidamente, estudiaremos los diferentes programas que hemos implementado para cada método. Notemos que cada método puede ser resuelto por diversos algoritmos. Es por ello que nos interesa seleccionar aquel que haga un uso más eficiente de los recursos, esencialmente, el tiempo de ejecución y la memoria de almacenamiento. Finalmente, comparando la complejidad obtenida con cada programa, se verá un ahorro en tiempo y memoria entre los cuatro métodos.

Como se puede observar, en este trabajo son necesarias ciertas nociones básicas sobre complejidad de algoritmos, por ello, todas ellas han sido introducidas en Apéndice A.

Capítulo 2

La cantidad de primos hasta n

En este capítulo se estudian diferentes métodos que desarrollan una formula para obtener el valor exacto de $\pi(n)$ para cualquier natural n. En primer lugar, la criba de Eratóstenes, introducida en el Capítulo 1 y, por otro lado, los métodos de Legendre, Meissel y Lehmer. Para ello se han consultado los libros [7], [9] y hemos desarrollado las demostraciones de manera similar a la que se presenta en [8].

2.0.1. Legendre

En 1830 Legendre desarrolló una fórmula, basándose en la criba de Eratóstenes, para obtener la cantidad de primos hasta un número natural n, $\pi(n)$. La idea consiste en buscar explícitamente los primos en $[1,\ \sqrt{n}]$ y a partir de ellos calcular cuántos primos hay en $(\sqrt{n},\ n]$. Destacar que solo interesa saber cuántos primos hay, se renuncia a conocer cuáles son los primos en $(\sqrt{n},\ n]$; para ello se apoya en la siguiente observación:

Observación 2.1. *Por la Observación 1.2 se deduce que el conjunto de primos en $(\sqrt{n},\ n]$, es el conjunto de enteros $\leq n$ no divisibles por ningún primo $\leq \sqrt{n}$.*

Para unificar notación con las fórmulas que estudiaremos posteriormente introducimos $\phi(n,a)$.

Definición 2.1. Sean dos naturales $n,\ a$, definimos $\phi(n,a)$ como la cantidad de enteros positivos menores o iguales que n y no divisibles por ninguno de los primeros a primos:

$$\phi(n,a) = |\{\text{ enteros x} \leq n \text{ no divisible por } p_1, \ldots, p_a\ \}|. \tag{2.1}$$

A continuación presentamos un resultado donde se desarrolla la fórmula general de $\phi(n,a)$ aplicando el Principio de Inclusión-Exclusión seguidamente enunciado.

Teorema 2.2 (Principio de Inclusión-Exclusión)**.**
Sean $S_1,\ \ldots, S_a$ conjuntos finitos, entonces:

$$\left|\bigcup_{i=1}^{a} S_i\right| = \sum_{1\leq i\leq a} |S_i| - \sum_{1\leq i<j\leq a} |S_i\cap S_j| + \sum_{1\leq i<j<k\leq a} |S_i\cap S_j\cap S_k| + \cdots + (-1)^a\, |S_1\cap S_2\cap\cdots\cap S_a|.$$

Teorema 2.3. *Sean $a,\ n\in\mathbb{N}$,*

$$\phi(n,a) = n - \sum_{1\leq i\leq a}\left[\frac{n}{p_i}\right] + \sum_{1\leq i<j\leq a}\left[\frac{n}{p_i p_j}\right] - \sum_{1\leq i<j<k\leq a}\left[\frac{n}{p_i p_j p_k}\right] + \cdots + (-1)^a\left[\frac{n}{p_1 p_2 \cdots p_a}\right]. \tag{2.2}$$

Demostración. Consideramos el conjunto $S = \{1,\ 2, \ldots, n\}$, con cardinalidad $|\ S\ | = n$, y los siguientes subconjuntos:

- $S_i = \{$ múltiplos de $p_i \leq n \}\ \forall\, i = 1, \ldots, a$, con cardinalidad:

$$| S_i | = |\{ \text{ múltiplos de } p_i \leq n \}| = \left| \left\{ p_i, 2\,p_i, \ldots, \left[\frac{n}{p_i}\right] p_i \right\} \right| = \left[\frac{n}{p_i}\right].$$

- Notar que $S_{i_1} \cap S_{i_2} \cap \cdots \cap S_{i_k} = \{$ múltiplos de $p_{i_1}\, p_{i_2} \cdots p_{i_k} \leq n \}\ \forall\, i_1 < i_2 < \cdots < i_k \leq a$.
  La cardinalidad de la intersección es:

$$|S_{i_1} \cap S_{i_2} \cap \cdots \cap S_{i_k}| = |\{ \text{ múltiplos de } p_{i_1}\, p_{i_2} \cdots p_{i_k} \leq n \}| = \left[\frac{n}{p_{i_1} \cdots p_{i_k}}\right].$$

De la definición de $\phi(n,a)$, tenemos

$$\phi(n,a) = |\{ \text{ enteros} \leq n \text{ no divisibles por } p_1, \ldots, p_a \}| = \left| S \setminus \bigcup_{i=1}^{a} S_i \right| = |S| - \left| \bigcup_{i=1}^{a} S_i \right|.$$

El resultado se obtiene aplicando el Principio de Inclusión-Exclusión, Teorema 2.2, a los subconjuntos finitos de $S,\ S_1,\ \ldots, S_a$. □

Observación 2.4. *En particular, aplicando la Observación 2.1 y la Definición de $\phi(n,a)$, resulta*

$$\phi(n, \pi(\sqrt{n})) = \left| \left\{ \textit{ enteros} \leq n \textit{ no divisibles por } p_1, \ldots, p_{\pi(\sqrt{n})} \right\} \right| = \left| \{ \textit{ primos} \in (\sqrt{n},\ n] \} \right|,$$

luego es claro que $\pi(n) = \pi(\sqrt{n}) - 1 + \phi(n,\ \pi(\sqrt{n}))$.

Como hemos indicado al principio de la sección la idea de Legendre es obtener la cantidad de primos en $(\sqrt{n},\ n]$ a partir de los primos hasta $\sqrt{n}$, luego queda demostrado el siguiente Teorema.

Teorema 2.5 (Teorema de Legendre)**.** *Sea $n \in \mathbb{N}$,*

$$\pi(n) = \pi(\sqrt{n}) - 1 + \phi(n, \pi(\sqrt{n})) = \pi(\sqrt{n}) - 1 + n - \sum_{1 \leq i \leq \pi(\sqrt{n})} \left[\frac{n}{p_i}\right] + \sum_{1 \leq i < j \leq \pi(\sqrt{n})} \left[\frac{n}{p_i\, p_j}\right] -$$
$$- \sum_{1 \leq i < j < k \leq \pi(\sqrt{n})} \left[\frac{n}{p_i\, p_j\, p_k}\right] + \cdots + (-1)^{\pi(\sqrt{n})} \left[\frac{n}{p_1\, p_2 \cdots p_{\pi(\sqrt{n})}}\right].$$

2.0.2. Meissel-Lehmer

Legendre llegó a calcular $\pi(10^6)$ aunque erróneamente; un ejemplo sencillo de la fórmula que se presenta, para $n = 100$ en Apéndice B, nos da una idea de la dificultad que conllevan las cuentas de su fórmula. Como razonaremos en el Capítulo 3, el coste computacional de su método es apenas ligeramente mejor que el de la criba de Eratóstenes. En 1870 Meissel realizó una eficiente modificación a la fórmula de Legendre con la que consiguió calcular, en 1885, $\pi(10^9) = 50847478$ (aunque erróneamente, valor 56 unidades menor que el real).
Hasta la era de los ordenadores no se realizaron más avances. En 1959 Lehmer modificó el método de Meissel y, con la ayuda de un ordenador, consiguió corregir el error de Meissel, hasta entonces inadvertido, y calculó $\pi(10^{10})$ (aunque nuevamente erróneo por excederse en una unidad).

Como veremos en el estudio de la complejidad de Legendre, Meissel y Lehmer, en el Capítulo 3, la parte más costosa de la fórmula para $\pi(n)$ es obtener $\phi(n,a)$; este coste disminuirá cuanto menor sea el valor de a.
La idea de Meissel es identificar los primos hasta $\sqrt[3]{n}$ y contar cuántos hay en $(\sqrt[3]{n},\ n]$, para ello calculamos $\phi(n, \pi(\sqrt[3]{n}))$, con $a = \pi(\sqrt[3]{n})$. Siguiendo el mismo razonamiento, Lehmer reduce el valor de a

hasta $\pi(\sqrt[4]{n})$.

Si reducimos el valor de a se simplifican las cuentas de $\phi(n,a)$ y, al mismo tiempo, para obtener $\pi(n)$ es necesario introducir nuevos factores. Por ello la fórmula de Meissel y sus generalizaciones introducen y estudian los conjuntos $P_k(n,a)$.

Definición 2.2. Dados tres naturales, $n,\ a,\ k$, $P_k(n,a)$ es la cantidad de enteros menores o iguales que n que se obtienen como producto de k primos, no necesariamente distintos, mayores que p_a :

$$P_k(n,a) = |\{ \text{ enteros } x = p_{i_1} \cdots p_{i_k} \leq n \text{ tq } p_{a+1} \leq p_{i_1} \leq \ldots \leq p_{i_k} \}|, \text{ con } k = 1, 2, \ldots \tag{2.3}$$

En particular, $P_1(n,a) = |\{ \text{ enteros } x = p_i \leq n \text{ tq } p_{a+1} \leq p_i \}| = |\{ \text{ primos en } (p_a,\ n] \}| = \pi(n) - a$.

Una vez presentados los términos $P_k(n,a)$, y conociendo la definición de $\phi(n,a)$, vamos a razonar una fórmula para $\pi(n)$.

Teorema 2.6. *Sean $a,\ n \in \mathbb{N}$,*

$$\pi(n) = a - 1 + \phi(n,a) - P_2(n,a) - P_3(n,a) - \cdots \tag{2.4}$$

Demostración. El único entero positivo menor que $p_a\ \forall\ a$ y que no sea múltiplo de ninguno de los a primeros primos es el 1. El resto de números $\leq n$ que no son múltiplos de los primeros a primos, por el TFA, son compuestos de los primos en $[p_{a+1},\ n]$, o bien son los propios primos. Es decir,

$$\phi(n,a) = 1 + P_1(n,a) + P_2(n,a) + P_3(n,a) + \cdots \Rightarrow \pi(n) = a - 1 + \phi(n,a) - P_2(n,a) - P_3(n,a) - \cdots \quad \square$$

La expresión anterior tiene una cantidad finita de sumandos, veamos ahora cuántos términos $P_k(n,a)$ son necesarios. Esto va a depender únicamente de a.

Proposición 2.7. *Sean $a,\ n \in \mathbb{N}$,*

$$\textit{si } a \textit{ verifica, } n^{\frac{1}{k}} < p_{a+1} \leq n^{\frac{1}{k-1}}, \textit{ entonces } P_{k-1}(n,a) \neq 0 \textit{ y } P_m(n,a) = 0\ \forall\ m \geq k.$$

Demostración.

- Sea $n^{\frac{1}{k}} < p_{a+1}$, todos los enteros $x = p_{i_1} \cdots p_{i_k}$ tq $p_{a+1} \leq p_{i_1} \leq \cdots \leq p_{i_k}$ verifican que $x = p_{i_1} \cdots p_{i_k} > n$. Luego por definición $P_k(n,a) = 0$.

- Sea ahora $p_{a+1} \leq n^{\frac{1}{k-1}}$. Entre los primos $p_{a+1} \leq p_{i_1} \leq \cdots \leq p_{i_{k-1}}$ al menos existe uno verificando $p_{i_j} = p_{a+1} \geq n^{\frac{1}{k-1}}$, luego al menos existe el entero $x = (p_{a+1})^{k-1} \leq n$. Así, por definición, tenemos que $P_{k-1}(n,a) \neq 0$. $\square$

Observación 2.8. *La fórmula* (2.4) *para $\pi(n)$ incluye al método de Legendre. Este se corresponde con el caso particular en que $a = \pi(\sqrt{n})$. En este caso se tiene $\sqrt{n} < p_{a+1} \leq n$, por el resultado anterior, Proposición 2.7, $P_k(n, \pi(\sqrt{n})) = 0\ \forall\ k \geq 2$, luego $\pi(n) = a - 1 + \phi(n,a)$.*

Proposición 2.9. *Sean $a,\ n \in \mathbb{N}$ y sea $k \geq 2$. $P_k(n,a)$ puede calcularse mediante la fórmula de recurrencia:*

$$P_k(n,a) = \sum_{i=a+1}^{\pi(n^{\frac{1}{k}})} P_{k-1}\left(\frac{n}{p_i}, i-1\right).$$

Demostración.

$$P_k(n,a) = |\{ \text{ enteros } x = p_{i_1} \cdots p_{i_k} \leq n \text{ tq } p_{a+1} \leq p_{i_1} \leq \cdots \leq p_{i_k} \}| =$$
$$= \left| \left\{ \text{ enteros } x = p_{i_2} \ldots p_{i_k} \leq \frac{n}{p_{a+1}} \text{ tq } p_{a+1} \leq p_{i_2} \leq \cdots \leq p_{i_k} \right\} \right| +$$
$$+ \left| \left\{ \text{ enteros } x = p_{i_2} \cdots p_{i_k} \leq \frac{n}{p_{a+2}} \text{ tq } p_{a+2} \leq p_{i_2} \leq \cdots \leq p_{i_k} \right\} \right| + \cdots = \sum_{i \geq a+1} P_{k-1}\left(\frac{n}{p_i}, i-1\right).$$

Para que P_k solo contenga sumandos > 0, el sumatorio llega hasta el índice i tal que $p_{a+1} \leq p_i \leq n^{\frac{1}{k}}$. En particular, vemos que tenemos $a < \pi(n^{\frac{1}{k}})$. □

Ahora ya podemos enunciar y demostrar los Teoremas de Meissel y Lehmer.

Teorema 2.10 (Teorema de Meissel)**.** *Sea $n \in \mathbb{N}$, $a = \pi(n^{\frac{1}{3}})$, $b = \pi(n^{\frac{1}{2}})$*

$$\pi(n) = a - 1 + \phi(n,a) - P_2(n,a) = \phi(n,a) + \frac{(b+a-2)(b-a+1)}{2} - \sum_{i=a+1}^{b} \pi\left(\frac{n}{p_i}\right).$$

Demostración. Meissel desarrolló la fórmula (2.4) de $\pi(n)$ para $a = \pi(n^{\frac{1}{3}})$. A continuación vamos a desarrollarla, para ello es necesario estudiar los términos $P_k(n,a)$ que aparecen en ella.

- $P_2(n,a) \neq 0$ y $P_k(n,a) = 0 \; \forall \, k \geq 3$ por la Proposición 2.7, ya que $a = \pi(n^{\frac{1}{3}}) \Rightarrow n^{\frac{1}{3}} < p_{a+1} \leq n^{\frac{1}{2}}$.
- Por la Proposición 2.9, nos queda la siguiente expresión:

$$P_2(n,a) = \sum_{i=a+1}^{\pi(n^{\frac{1}{2}})} P_1\left(\frac{n}{p_i}, i-1\right) = \sum_{i=a+1}^{b} \left[\pi\left(\frac{n}{p_i}\right) - (i-1)\right] = \sum_{i=a+1}^{b} \pi\left(\frac{n}{p_i}\right) - \sum_{i=a+1}^{b} (i-1) =$$
$$= \sum_{i=a+1}^{b} \pi\left(\frac{n}{p_i}\right) - \frac{(b-a)(b+a+1)}{2}, \; \forall \, a. \tag{2.5}$$

Luego, tenemos:

$$\pi(n) = a - 1 + \phi(n,a) - P_2(n,a) = \phi(n,a) + \frac{(b+a-2)(b-a+1)}{2} - \sum_{i=a+1}^{b} \pi\left(\frac{n}{p_i}\right). \qquad \square$$

Teorema 2.11 (Teorema de Lehmer)**.** *Sea $n \in \mathbb{N}$,*

$$\pi(n) = a - 1 + \phi(n,a) - P_2(n,a) - P_3(n,a) =$$
$$= \phi(n,a) + \frac{(b+a-2)(b-a+1)}{2} - \sum_{i=a+1}^{b} \pi\left(\frac{n}{p_i}\right) - \sum_{i=a+1}^{c} \sum_{j=1}^{b_i} \left[\pi\left(\frac{n}{p_i p_j}\right) - (j-1)\right],$$
$$\text{con } a = \pi(n^{\frac{1}{4}}),\; c = \pi(n^{\frac{1}{3}}),\; b = \pi(n^{\frac{1}{2}}),\; b_i = \pi\left(\sqrt{\frac{n}{p_j}}\right).$$

Demostración. La idea de Lehmer es una generalización del método de Meissel. Desarrolló la fórmula (2.4) de $\pi(n)$ para $a = \pi(n^{\frac{1}{4}})$. Vamos a estudiar los términos $P_k(n,a)$ que aparecen en esta fórmula.

- $P_3(n,a) \neq 0$ y $P_k(n,a) = 0 \; \forall \, k \geq 4$ por la Proposición 2.7, ya que $a = \pi(n^{\frac{1}{4}}) \Rightarrow n^{\frac{1}{4}} < p_{a+1} \leq n^{\frac{1}{3}}$.
- P_2 y P_3 quedan determinados $\forall \, a$ en (2.5) y en la Proposición 2.9, respectivemente.

Obtenemos el resultado deseado operando $\pi(n) = a - 1 + \phi(n,a) - P_2(n,a) - P_3(n,a)$. □

Capítulo 3

Algoritmos

Hemos estudiado diferentes métodos que proporcionan el valor exacto de $\pi(n)$, la criba de Eratóstenes, Legendre, Meissel y Lehmer; una vez introducidos, en este capítulo vamos a crear un programa para cada uno de ellos. Cada método tiene diversos algoritmos que pueden implementarlo, el objetivo es obtener aquel que haga un uso más eficiente del tiempo de ejecución y del espacio de almacenamiento. Finalmente, compararemos la complejidad obtenida por cada programa, tanto explícitamente como con ejemplos numéricos, observando una clara mejoría en tiempo y memoria. Los algoritmos a los que vamos a referirnos a lo largo de este capítulo se encuentran en el Apéndice F.

La notación (habitual) básica del capítulo se recuerda en el Apéndice A.

3.1. Criba de Eratóstenes

3.1.1. Descripción del algoritmo

En esta sección desarrollamos un algoritmo para la criba de Eratóstenes. Para ello empezaremos describiendo el algoritmo más intuitivo para este método, el Algoritmo 1 presentado en la página 10, y comentaremos las mejoras que se han llevado a cabo para obtener el algoritmo finalmente implementado.

3.1.2. Complejidad

Para estudiar la complejidad del algoritmo será necesario presentar los siguientes resultados preliminares, los cuales se encuentran demostrados en Apéndice C.

Proposición 3.1.

$$\sum_{i=2}^{a} \frac{1}{i \ln i} \in \Theta(\ln\ln a).$$

Proposición 3.2.

$$\sum_{i=1}^{a} \frac{1}{p_i} \in \Theta(\ln\ln a).$$

Veámos a continuación la complejidad del Algoritmo 1, *criba de Eratóstenes*, es decir, la estimación del tiempo de ejecución, $t(n)$, y de la memoria de almacenamiento necesaria, $M(n)$.

Proposición 3.3. *Para el Algoritmo 1* $t(n) \in \mathcal{O}(n \ln\ln n)$ *y* $M(n) \in \mathcal{O}(n)$.

Demostración. Para calcular $t(n)$ identificamos una sentencia barómetro que se encuentra en la línea 13 del Algoritmo 1, donde se criban los múltiplos de los primos menores o iguales que $\sqrt{n}$: $p_1, \ldots, p_{\pi(\sqrt{n})}$.

Algoritmo 1 criba de Eratóstenes

Entrada: Un número entero positivo n.

Salida: Secuencia de $n+1$ bits donde el índice i corresponde al número i.
El i-ésimo bit es 1 sii i es primo, y es 0 si no lo es.

```
1: listado ← secuencia de bits [0, ..., n], inicializados a 1;
2: listado[0] ← 0;
3: listado[1] ← 0;
4: i ← 2;
5: //Buscamos los primos hasta √n
6: mientras i ≤ √n hacer
7:    mientras listado[i] == 0 hacer
8:       i ← i+1;    //Si es 0 buscamos el siguiente primo
9:    fin mientras
10:   //Cribamos los múltiplos de i que son ≤ n
11:   j ← i*i;        //Primer múltiplo de i aún no cribado por primos anteriores
12:   mientras j ≤ n hacer
13:      listado[j] = 0;
14:      j ← j+i;  //Toma el siguiente múltiplo
15:   fin mientras
16:   i ← i+1;
17: fin mientras
18: devolver listado;
```

Sabemos que $\forall\, d \leq n,\ |\{\text{ múltiplos de } d \leq n\ \}| = \left[\frac{n}{d}\right]$, por lo tanto, el número de veces que se ejecuta esta sentencia está acotado superiormente por

$$\frac{n}{2}+\frac{n}{3}+\cdots+\frac{n}{p_{\pi(\sqrt{n})}} = n\sum_{i=1}^{\pi(\sqrt{n})}\frac{1}{p_i}. \tag{3.1}$$

Por la Proposición 3.2 y aplicando el Teorema de Chebyshev, como se verá más adelante en 4.10:

$$n\sum_{i=1}^{\pi(\sqrt{n})}\frac{1}{p_i} \leq nD\ln\ln(\pi(\sqrt{n})) < nD\ln\ln\left(\frac{B\sqrt{n}}{\ln\sqrt{n}}\right) = nD\ln\ln n,\ \text{si}\ \frac{B}{\ln\sqrt{n}} \leq 1 \Leftrightarrow n \geq e^{2B}.$$

siendo D una constante y $B > 0$ un real.

Por otro lado, en el estudio de la complejidad también nos encargamos del espacio de almacenamiento necesario. Para calcular el listado de primos hasta n necesitamos una cadena de longitud $n+1$, es decir, $n+1$ bits de memoria. □

3.1.3. Mejoras

A continuación vamos a presentar las mejoras que hay entre el algoritmo más intuitivo, el Algoritmo 1 anteriormente presentado, y *piEratostenes*, ver Sección F.5, el algoritmo finalmente implementado que es más eficiente para obtener $\pi(n)$ aplicando la criba de Eratóstenes.
Tal y como está descrito el Algoritmo 1 la función necesita $n+1$ bits de memoria que corresponden al tamaño de la secuencia utilizada. Es posible reducir memoria de dos formas:

1. Podemos prescindir del estudio de todos los números pares recordando que el único primo par es el 2. Vamos a eliminar la representación de los pares en las cadenas de bits que creemos. De esta forma, en el estudio de cualquier intervalo $[a,b]$ será necesaria una cadena de longitud $\frac{b-a}{2}$ en vez de $b-a$.

2. Basta con usar dos secuencias de bits, cada una con longitud $\frac{\sqrt{n}}{2}$; ya que conociendo el funcionamiento de la criba solo es estrictamente necesario obtener los primos hasta $\sqrt{n}$. Para llevar a cabo esta mejora hemos desarrollado dos funciones donde cada una creará una de estas secuencias.

   - En la primera secuencia almacenamos en memoria los primos menores o iguales que $\sqrt{n}$, la obtenemos con el programa *cribaEratóstenes*, ver Sección F.1, para $\sqrt{n}$.
     En esta primera secuencia codificamos los enteros impares en el intervalo $[3,\ \sqrt{n}]$. De este modo, la posición i-ésima representa el número $3+2i$. Inicialmente la secuencia estará inicializada a 1; luego se irá cribando siguiendo el método de Eratóstenes: identificando los primos hasta $\sqrt[4]{n}$ y eliminando sus múltiplos hasta $\sqrt{n}$. Indicaremos con un 1 los que son primos y con 0 los que no. De este modo tenemos almacenados los primos en $[2,\ \sqrt{n}]$.
   - En la segunda secuencia codificamos sucesivamente los enteros impares en los intervalos $[i\sqrt{n}+1,\ (i+1)\sqrt{n}]$ con $i \in [1,\ \sqrt{n}-1]$, para ello aplicaremos el programa *cribaIntervalo*, ver Sección F.4, sucesivamente en cada uno de estos intervalos.
     Veámoslo con más detalle. Queremos cribar el intervalo $(\sqrt{n},\ n]$, para ello lo vamos a estudiar por tramos: lo descomponemos en subintervalos de longitud $\sqrt{n}$, lo que equivale a secuencias de bits de longitud $\frac{\sqrt{n}}{2}$ que se criban de forma independiente; donde cada subintervalo se va a ir sobrescribiendo sobre la misma secuencia de bits. Como hemos dicho, la secuencia i-ésima representa los números impares en el intervalo $[i\sqrt{n}+1,\ (i+1)\sqrt{n}]$; inicialmente cada secuencia estará inicializada a 1, y se van a ir cribando siguiendo nuevamente el método de Eratóstenes, indicando con 1 los primos y con 0 los compuestos; para ello utilizamos los primos memorizados en la primera secuencia, $\{p_1,\ldots,p_{\pi(\sqrt{n})}\}$.
     Hacemos notar que para cribar un intervalo cualquiera denotado como $[n0,\ nF]$, basta trabajar con los primos $\leq \sqrt{nF} \leq \sqrt{n}$, lo cual supone un ahorro adicional de tiempo.

De este modo hemos pasado de necesitar una secuencia de longitud n a dos de longitud $\frac{\sqrt{n}}{2}$, por lo tanto el programa *piEratostenes* tiene la memoria en $\mathcal{O}(\sqrt{n})$.
Finalmente tenemos que $\pi(n)$ es el número de primos hasta $\sqrt{n}$, es decir, la cantidad de 1's en la primera secuencia, más la cantidad de primos en cada subintervalo de $(\sqrt{n},\ n]$, es decir, la cantidad de 1's de sus correspondientes secuencias.

Por ejemplo, para $n = 100$. En primer lugar guardamos en memoria los primos en $[2,\ 10]$, la secuencia de bits que codifica los enteros impares en este intervalo es 1110. A continuación cribamos $(10,\ 100]$, es decir, cribamos los subintervalos $[11,\ 20]$, $[21,\ 30]$, $\ldots$, $[91,\ 100]$, la secuencia de bits que codifica los impares de este primer tramo sería 11011. Como solo consideramos los impares en dichos subintervalos, basta con 5 bits para representarlos ya que los diferentes tramos se van sobrescribiendo sobre la misma secuencia. Necesitamos pues: 9 bits, 4 para los primeros primos y 5 para analizar los tramos. Con el Algoritmo 1 hubiéramos necesitado una única cadena de 100 bits. Luego, hemos pasado a tener una memoria en $\mathcal{O}(\sqrt{100})$ en vez de en $\mathcal{O}(100)$.

De la Proposición 3.3, y de los comentarios previos, se sigue:

Teorema 3.4. *Para el algoritmo piEratóstenes, Sección F.5, $t(n) \in \mathcal{O}(n \ln\ln n)$ y $M(n) \in \mathcal{O}(\sqrt{n})$.*

Observación 3.5. *Para cribar el intervalo $[n0,\ nF]$ aplicamos la criba de Eratóstenes. Siguiendo el mismo procedimiento que en cribaEratóstenes y atendiendo a las mejoras que se han efectuado para llegar a un algoritmo más eficiente, como piEratóstenes, hemos desarrollado el programa cribaIntervalo, Sección F.4.*
El programa cribaIntervalo da la cantidad de primos en un intervalo $[n0,\ nF]$ con $t(n) \in \mathcal{O}((nF - n0)\ln\ln nF)$ y $M(n) \in \mathcal{O}(min\{(nF-n0),\ \sqrt{nF}\})$.

3.2. Suma de Legendre, $\phi(n,a)$

3.2.1. Descripción del algoritmo

En el Teorema 2.3 hemos visto la definición y desarrollo de $\phi(n,a)$. Veámos ahora uno de los algoritmos que obtienen $\phi(n,a)$, *sumaLegendre*, Algoritmo 2.

Algoritmo 2 *sumaLegendre*

Entrada: Una secuencia de N enteros $\{p_1,\ldots,p_N\}$, llamada *listaPrimos*.
Cuatro enteros positivos: a, *sumaAcumulada*, valor, i.

Salida: Un entero.

1: **si** $i<0$ OR $i\geq N$ OR $p_{i+1}>[valor]$ **entonces**
2: **devolver** *sumaAcumulada*;
3: **si no**
4: **para** $j=i+1;\ j\leq a;\ j=j+1$ **hacer**
5: $factor \leftarrow -\left[\frac{valor}{p_j}\right]$;
6: $sumaAcumulada \leftarrow sumaAcumulada +$
$factor + sumaLegendre(listaPrimos,\ a,\ sumaAcumulada,\ \text{factor},\ j+1)$;
7: **fin para**
8: **devolver** *sumaAcumulada*;
9: **fin si**

A continuación probaremos que el Algoritmo 2 calcula el valor correcto de $\phi(n,a)$. Para ello utilizaremos los siguientes resultados previos:

Lema 3.6. *Sean tres enteros positivos $m,\ n,\ N$ con $m<n$. Entonces, $\left[\frac{N}{mn}\right]=\left[\left[\frac{N}{m}\right]/n\right]$.*

Demostración. Podemos expresar

$$N=\left[\frac{N}{mn}\right]mn+o=\left[\frac{N}{mn}\right]mn+\left[\frac{o}{m}\right]m+r,\ \text{donde } r<m,\ o<mn,\ \text{y por tanto } \left[\frac{o}{m}\right]\leq\frac{o}{m}<n.$$

$$\text{Es inmediato que } \left[\frac{N}{m}\right]=\left[\frac{N}{mn}\right]n+\left[\frac{o}{m}\right] \text{ y por tanto } \left[\left[\frac{N}{m}\right]/n\right]=\left[\frac{N}{mn}\right]. \qquad \square$$

Teorema 3.7. *Sea $lP\equiv\{p_i\}$ la secuencia de los N primeros números primos.*

i) Si $0\leq i<a\leq N$, sumaLegendre(lP, a, sumaAcumulada, valor, i), devuelve

$$sumaAcumulada+\sum_{l=1}^{a-i}(-1)^l\left(\sum_{i<j_1<\cdots<j_l\leq a}\left[\frac{valor}{\Pi_{k=1}^{l}p_{j_k}}\right]\right)=$$
$$=sumaAcumulada+\sum_{l=1}^{a-i}(-1)^l\left(\sum_{i<j_1<\cdots<j_l\leq a}\left[\frac{valor}{p_{j_1}\cdots p_{j_l}}\right]\right). \qquad (3.2)$$

ii) Si $a\leq N$, sumaLegendre(lP, a, n, n, 0) devuelve el valor $\phi(n,a)$.

Demostración. i) Por inducción. Si $i\geq a$ o si $p_{i+1}>valor$, el algoritmo no modifica el valor de *sumaAcumulada*. Eso es precisamente lo que ocurre con (3.2) en esos casos, el sumatorio del miembro derecho es cero. Si $i\geq a$, el sumatorio consta de cero términos. Por otro lado, si $p_{i+1}>valor$, dado que lP es una secuencia ordenada, tenemos que para todo $j>i$ sucede que $[valor/p_j]\leq[valor/p_{i+1}]=0$. Por tanto, *sumaAcumulada* no se modifica.

Tomemos ahora la hipótesis de que la condición *i*) se verifica para todo $i > k$. Veremos que también se cumple para $i = k$. Por hipótesis de inducción, la iteración j-ésima del bucle incrementa el valor de *sumaAcumulada* en una cantidad:

$$-\left[\frac{valor}{p_j}\right]+\sum_{l=1}^{a-j}(-1)^l\left(\sum_{j<j_1<\cdots<j_l\leq a}-\left[\frac{valor}{p_j p_{j_1}\cdots p_{j_l}}\right]\right). \tag{3.3}$$

Al sumar las contribuciones de todas las iteraciones, $j \in [k+1,\ \ldots,\ a]$, obtenemos el resultado.

ii) Se sigue de *i*) y del desarrollo de $\phi(n,a)$, Teorema 2.3. □

3.2.2. Complejidad

A continuación vamos a ver la complejidad de *sumaLegendre*, tiempo de ejecución y espacio de almacenamiento. Para el estudio del tiempo es necesario introducir el siguiente resultado fácilmente demostrable, ver Apéndice C:

Resultado 3.8.

$$Si\ a > \frac{m^2}{m-1}\ \Rightarrow\ \binom{a}{m} > \frac{a^m}{m!\ m^m}.$$

Teorema 3.9. *Para el algoritmo de $\phi(n,a)$, Algoritmo 2, $t(n) \in \mathcal{O}(n)$ y $M(n) \in \mathcal{O}(a+\ln n)$.*

En el caso particular $a = \pi(n^{\frac{1}{m}})$, $t(n) \in \mathcal{O}(n) \cap\ \Omega\left(\frac{n}{(\ln n)^m}\right)$.

Demostración. En este algoritmo localizamos la sentencia barómetro en la línea 5. Cada vez que se ejecuta se calcula $\left[\frac{valor}{p_j}\right]$, es decir, un término de la forma $\left[\frac{n}{p_{j_1}\cdots p_{j_k}}\right]$; la cantidad de estos términos es igual al número de posibles enteros, producto de primos, de la siguiente forma:

$$x = \prod_{p_i=p_1}^{p_a} p_i^{\alpha_i} \leq n \text{ con } \alpha_i \in \{0,1\};\ \ x \leq n \text{ ya que si no } \left[\frac{n}{x}\right] = 0. \tag{3.4}$$

Por lo tanto, cada vez que se calcula un término de $\phi(n,a)$ se genera uno de los posibles números de la forma (3.4) y se divide n entre dicho número; siendo cada vez el número generado distinto. Podemos intuir fácilmente que la cantidad de tales números es menor que n, ya que están en el intervalo $[2,n]$. Luego tenemos que, $\forall\ n\ y\ \forall\ a$, el tiempo de ejecución está acotado superiormente por n, $t(n) \in \mathcal{O}(n)$.

Consideramos el caso particular en que $a = \pi(n^{\frac{1}{m}})$, con $m \geq 2$.

Existen $\binom{a}{m}$ posibles productos de m primos distintos tal que $p_{i_1}\cdots p_{i_m} \leq n$, con $\{p_{i_1},\ldots,p_{i_m}\} \subset \{p_1,\ldots,p_a\}$. Esto es, cada producto corresponde a un término distinto de los calculados por nuestro algoritmo, $\binom{a}{m}$ es pues, una cota inferior del número de dichos términos.

Aplicando el Resultado 3.8 y el TNP a $a = \pi(n^{\frac{1}{m}})$, vamos a acotar inferiormente $\binom{a}{m}$. Si $a > \frac{m^2}{m-1}$, tenemos:

$$\text{la cantidad de términos a calcular en } \phi(n,a) > \binom{a}{m} > \frac{a^m}{m!\,m^m} = \frac{1}{m!\,m^m}\left(\frac{n^{\frac{1}{m}}}{(\ln n)^{\frac{1}{m}}}\right)^m = \frac{1}{m!}\,\frac{n}{(\ln n)^m}.$$

Vamos a estudiar ahora el espacio de almacenamiento. Para el estudio de la memoria, en este algoritmo recursivo, debemos conocer cuantas instancias de *sumaLegendre* se ejecutan simultáneamente. El número máximo de instancias es la cantidad máxima de primos distintos cuyo producto es $\leq n$, por lo

tanto, queremos conocer el valor máximo de k tal que $p_{i_1} p_{i_2} \cdots p_{i_k} \leq n$, con $\{p_{i_1}, \ldots, p_{i_k}\} \subset \{p_1, \ldots, p_a\}$, veámoslo:

$$n \geq p_{i_1} \cdots p_{i_k} \geq p_1 \cdots p_k > 1 \cdots k = k! \sim \sqrt{2\pi n} \left(\frac{k}{e}\right)^k > \left(\frac{k}{e}\right)^k \Rightarrow \left(\frac{k}{e}\right)^k < n \Rightarrow k < \frac{\ln n}{\ln\left(\frac{k}{e}\right)} < \ln n.$$

Por otro lado, el algoritmo necesita acceso a los a primeros primos. □

Observación 3.10. *Con el estudio de la complejidad podemos hacer dos observaciones en relación al tiempo de ejecución para los casos particulares de Legendre* $(m = 2)$*, Meissel* $(m = 3)$ *y Lehmer* $(m = 4)$*:*

i) La complejidad será menor cuanto mayor sea m, por lo tanto, cuanto menor sea a; esto es debido a que $\frac{n}{(\ln n)^m}$ *es decreciente con m. Luego tenemos que la complejidad de* $\phi(n,\ \pi(n^{1/4}))$*, fórmula de Lehmer, es menor que la de* $\phi(n,\ \pi(n^{1/3}))$*, fórmula de Meissel, y ésta a su vez, menor que la de* $\phi(n,\ \pi(n^{1/2}))$*, Legendre.*

ii) Las acotaciones de $t(n)$ *son explícitas si* $a > \dfrac{m^2}{m-1}$*, es decir, si* $n \geq 11^2$ $(m = 2)$*,* $n \geq 11^3$ $(m = 3)$*,* $n \geq 13^4$ $(m = 4)$*.*

3.2.3. Mejoras

A continuación vamos a presentar las mejoras entre el Algoritmo 2, y el algoritmo más efectivo finalmente implementado, *sumaPhi*, ver Sección F.6.

El Algoritmo 2 es recursivo, es decir, se invoca a sí mismo. Los algoritmos recursivos son concisos e intuitivos; y en nuestro caso tenemos la ventaja de que su demostración es fácil (por inducción). Sin embargo, al ejecutarse conlleva un sobrecoste de tiempo y memoria. Cada vez que se invoca se crea una instancia del algoritmo, lo que conlleva una pérdida de tiempo en asignar memoria a esa instancia para almacenar sus variables e inicializarlas, así como en liberar dicha memoria cuando termina la ejecución de la iteración. [5].
Hemos traducido este algoritmo a otro equivalente iterativo, más efectivo, *sumaPhi*.

3.3. Legendre

3.3.1. Complejidad

El algoritmo finalmente implementado para obtener $\pi(n)$ con el método de Legendre es *piLegendre*, ver Sección F.7. Dado un parámetro de entrada, un natural n, calcula $\pi(n)$ siguiendo la fórmula del Teorema de Legendre, 2.5, es decir:

$$\pi(n) = a - 1 + \phi(n,a), \quad \text{con } a = \pi(\sqrt{n}). \tag{3.5}$$

Teorema 3.11. *Para el algoritmo piLegendre,* $t(n) \in \mathcal{O}(n) \cap \Omega\left(\dfrac{n}{(\ln n)^2}\right)$ *y* $M(n) \in \mathcal{O}(\sqrt{n})$.

Demostración. El tiempo de ejecución es el requerido por *cribaEratóstenes* para obtener el listado de los a primeros primos y por *sumaPhi*, para calcular $\phi(n,a)$, con $a = \pi(\sqrt{n})$.

Tras haber estudiado la complejidad de ambos programas en la Propiedad 3.3 y el Teorema 3.9, respectivamente, podemos asegurar que el tiempo de ejecución de *piLegendre* está en $\mathcal{O}(n + \sqrt{n}\ln\ln\sqrt{n}) \cap$

$\Omega\left(\frac{n}{(\ln n)^2}\right) = \mathcal{O}(n) \cap \Omega\left(\frac{n}{(\ln n)^2}\right)$.

Por otro lado, la memoria utilizada por *piLegendre* es la memoria necesaria por ambos programas. En este caso obtenemos la lista de primos con exactamente a primos, así por 3.3 y 3.9 la memoria está en $\mathcal{O}(\sqrt{n}+\sqrt{n}) = \mathcal{O}(\sqrt{n})$. □

3.4. $P_2(n,a)$

3.4.1. Descripción del algoritmo

En este programa obtenemos $P_2(n,a)$ siguiendo la siguiente fórmula:

$$P_2(n,a) = \sum_{i=a+1}^{b} \pi\left(\frac{n}{p_i}\right) - \frac{(b-a)(b+a+1)}{2}, \text{ con } b = \pi(n^{\frac{1}{2}}). \tag{3.6}$$

Como indica la Proposición 2.7, $P_2(n,a) \neq 0$ y $P_m(n,a) = 0 \ \forall\ m \geq 3 \ \Leftrightarrow$ a verifica, $n^{\frac{1}{3}} < p_{a+1} \leq n^{\frac{1}{2}}$. Luego es natural tomar $a < \pi(n^{\frac{1}{2}})$ para no tener ningún término de P_2 nulo.

Algoritmo 3 P_2

Entrada: Dos enteros positivos, n y a.
Una secuencia de N enteros $\{p_1,\ldots,p_N\}$, llamada *listaPrimos*.

Salida: Un entero.
En el caso particular de que *listaPrimos* contenga los N primeros primos, con $N \geq \pi(n^{\frac{1}{2}})$, la salida es el valor de $P_2(n,a)$, la cantidad de enteros positivos menores o iguales que n tal que son producto de dos primos mayores o iguales que p_{a+1} y no necesariamente diferentes

1: $b \leftarrow \pi(\sqrt{n})$;
2: //Obtenemos el sumatorio $\sum_{i=a+1}^{b} \pi(\frac{n}{p_i})$
3: itermino $\leftarrow$ b;
4: sumatorio $\leftarrow$ 0;
5: $n0 \leftarrow \sqrt{n}+1$;
6: **para** $i = b$; $i \geq a+1$; $i = i-1$ **hacer**
7: $nF \leftarrow \frac{n}{p_i}$
8: itermino $\leftarrow$ itermino $+ cribaIntervalo(litaPrimos, n0, nF)$;
9: sumatorio $\leftarrow$ sumatorio + itermino;
10: $n0 \leftarrow nF+1$;
11: **fin para**
12: **devolver** sumatorio - $\frac{(b-a)(b+a-1)}{2}$;

3.4.2. Complejidad

Teorema 3.12. *Para el algoritmo de $P_2(n,a)$, ver Sección F.8, $t(n) \in \mathcal{O}\left(\frac{n}{p_{a+1}} \ln\ln n\right)$ y $M(n) \in \mathcal{O}\left(\sqrt{\frac{n}{p_{a+1}}} + \pi(n^{\frac{1}{2}})\right)$.*

Demostración. En este programa, la sentencia que requiere de mayor coste computacional se encuentra en la línea 8, donde se calcula cada sumando: $\pi\left(\frac{n}{p_i}\right)$ para $i = a+1,\ldots,b$; dicho sumando se obtiene de la siguiente forma:

$$\begin{aligned}\pi\left(\frac{n}{p_i}\right) =& \pi\left(\frac{n}{p_{i+1}}\right) + \left|\text{ primos} \in \left[\frac{n}{p_{i+1}}+1,\ \frac{n}{p_i}\right]\right| = \\ =& \pi\left(\frac{n}{p_{i+1}}\right) + cribaIntervalo\left(listaPrimos,\ \frac{n}{p_{i+1}}+1,\ \frac{n}{p_i}\right).\end{aligned}$$

El tiempo de ejecución del programa es el de esta sentencia por el número de veces que se ejecuta. Al estudiarla vemos que su coste computacional es el de *cribaIntervalo* y, por la Observación 3.5, sabemos que en un intervalo cualquiera $[n0,\ nF]$ el tiempo de ejecución está en $\mathcal{O}((nF-n0)\ln\ln nF)$. Por otro lado, también sabemos que el número de veces que se ejecuta dicha sentencia es igual al número de primos que hay en $[p_{a+1},\ \sqrt{n}]$, es decir $b-a$. Por lo tanto el tiempo de ejecución de P_2 está acotado superiormente por:

$$\sum_{i=a+1}^{b}(n_{Fi}-n_{0i})\ln\ln n_{Fi} < \ln\ln\frac{n}{p_{a+1}}\sum_{i=a+1}^{b}(n_{Fi}-n_{0i}) < \ln\ln\frac{n}{p_{a+1}}\left(\frac{n}{p_{a+1}}-(\sqrt{n}+1)\right) < \ln\ln n\left(\frac{n}{p_{a+1}}\right).$$

Estudiemos ahora la memoria necesaria siguiendo el razonamiento anterior. Sabemos que en un intervalo $[n0,\ nF]$, *cribaIntervalo* tiene la memoria en $\mathcal{O}(min\{(nF-n0),\ \sqrt{nF}\})$. Por lo tanto, la memoria está acotada superiormente por $\sqrt{\frac{n}{p_{a+1}}}$.
Por otro lado, el algoritmo necesita acceso a los primos hasta $\sqrt{n}$. Así la memoria necesaria es la suma de ambas, $\mathcal{O}\left(\sqrt{\frac{n}{p_{a+1}}}+\pi(n^{\frac{1}{2}})\right)$. □

3.5. $P_3(n,a)$

3.5.1. Descripción del algoritmo

En este programa obtenemos $P_3(n,a)$ aplicando la siguiente fórmula:

$$P_3(n,a) = \sum_{i=a+1}^{c} P_2\left(\frac{n}{p_i},\ i-1\right),\ \text{con } c = \pi(n^{\frac{1}{3}}). \tag{3.7}$$

Como indica la Proposición 2.7, $P_3(n,a) \neq 0$ y $P_m(n,a) = 0\ \forall\ m \geq 4\ \Leftrightarrow\ a$ verifica, $n^{\frac{1}{4}} < p_{a+1} \leq n^{\frac{1}{3}}$. Luego es natural tomar $a < \pi(n^{\frac{1}{3}})$ para así no tener ningún término nulo.

Algoritmo 4 P_3

Entrada: Dos enteros positivos, n y a.
Una secuencia de N enteros $\{p_1,\ldots,p_N\}$, llamada *listaPrimos*.

Salida: Un entero.
En el caso particular de que *listaPrimos* contenga los N primeros primos, con $N \geq \pi(n^{\frac{1}{3}})$, devuelve el valor de $P_3(n,a)$, la cantidad de enteros positivos menores o iguales que n, producto de tres primos mayores o iguales que p_{a+1} y no necesariamente diferentes.

1: $c \leftarrow \pi(n^{\frac{1}{3}})$;
2: //Vamos a obtener el sumatorio $\sum_{p_i=p_{a+1}}^{n^{\frac{1}{3}}} P_2\left(\frac{n}{p_i},\ i-1\right)$
3: aux $\leftarrow$ a;
4: sumatorio $\leftarrow$ 0;
5: **para** $i=a+1;\ i\leq c;\ i=i+1$ **hacer**
6: sumatorio $\leftarrow$ sumatorio + $P_2\left(\frac{n}{p_i},\ \text{aux},\ \textit{listaPrimos}\right)$;
7: aux $\leftarrow$ aux +1;
8: **fin para**
9: **devolver** sumatorio;

3.5.2. Complejidad

Teorema 3.13. *Para el algoritmo de $P_3(n,a)$, ver Sección F.9,* $t(n) \in \mathcal{O}\left(\frac{n^{\frac{4}{3}}}{p_{a+1}^2}\ln\ln n\right)$ *y* $M(n) \in \mathcal{O}\left(\sqrt{\frac{n}{p_{a+1}^2}}+\pi\left(\sqrt{\frac{n}{p_{a+1}}}\right)\right)$.

Demostración. En este programa, la sentencia que requiere de mayor coste computacional se encuentra en la línea 6, donde calculamos cada término $P_2\left(\frac{n}{p_i},\ i-1\right)$ para $i=a+1,\ldots,c$.

El tiempo de ejecución del programa es la suma de todos los tiempos de ejecución de dicha sentencia las $c-a$ veces que se ejecuta. es decir, $\sum_{i=a+1}^{c}[$ complejidad de $P_2\left(\frac{n}{p_i},\ i-1\right)]$.
Por el Teorema 3.12 conocemos el tiempo de ejecución para cada término P_2, por lo tanto, el tiempo para P_3 está acotado superiormente por:

$$\sum_{i=a+1}^{c}\frac{n}{p_i^2}\ln\ln\frac{n}{p_i} < \left(\frac{n}{p_{a+1}^2}\right)(c-a)\ln\ln\frac{n}{p_{a+1}} < \left(\frac{n}{p_{a+1}^2}\right)(c)\ln\ln\frac{n}{p_{a+1}} <$$
$$< \left(\frac{nn^{\frac{1}{3}}}{p_{a+1}^2}\right)\ln\ln n = \left(\frac{n^{\frac{4}{3}}}{p_{a+1}^2}\right)\ln\ln n.$$

Veámos ahora el espacio de almacenamiento siguiendo el razonamiento anterior. La cota superior de la memoria requerida por las instancias de P_2 es $\sqrt{\frac{n}{p_{a+1}^2}}+\pi\left(\sqrt{\frac{n}{p_{a+1}}}\right)$ por el Teorema 3.12.
Por otro lado, el algoritmo necesita acceso a los $\pi(n^{\frac{1}{3}})$ primeros primos. Por lo tanto, la memoria de este algoritmo está en $\mathcal{O}\left(\sqrt{\frac{n}{p_{a+1}^2}}+\pi(n^{\frac{1}{3}})+\pi\left(\sqrt{\frac{n}{p_{a+1}}}\right)\right) = \mathcal{O}\left(\sqrt{\frac{n}{p_{a+1}^2}}+\pi\left(\sqrt{\frac{n}{p_{a+1}}}\right)\right)$. □

3.6. Meissel

3.6.1. Complejidad

El programa *piMeissel*, ver Sección F.10, obtiene $\pi(n)$ a partir de un parámetro de entrada, un natural n, siguiendo el método de Meissel, Teorema 2.10:

$$\pi(n) = a - 1 + \phi(n,a) - P_2(n,a), \text{ con } a = \pi(n^{\frac{1}{3}}). \tag{3.8}$$

Teorema 3.14. *Para el algoritmo piMeissel,* $t(n) \in \mathcal{O}(n) \cap \Omega\left(\frac{n}{(\ln n)^3}\right)$ *y* $M(n) \in \mathcal{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}\right)$.

Demostración. Para calcular $\pi(n)$, siguiendo la fórmula (3.8), empezamos obteniendo y contando los primos hasta $\sqrt{n}$ necesarios para obtener P_2. Esto conlleva un coste de tiempo $t(n) \in \mathcal{O}(\sqrt{n}\ln\ln\sqrt{n})$, y de memoria $M(n) \in \mathcal{O}(\pi(n^{\frac{1}{2}}))$, por Proposición 3.3.
El coste de calcular $\phi(n,\pi(n^{\frac{1}{3}}))$ es: $t(n) \in \mathcal{O}(n) \cap \Omega\left(\frac{n}{(\ln n)^3}\right)$ y $M(n) \in \mathcal{O}(\pi(n^{\frac{1}{3}}) + \ln n)$, por Teorema 3.9.
Obtenemos $P_2(n,\pi(n^{\frac{1}{3}}))$. Siendo $a = \pi(n^{\frac{1}{3}}) \Rightarrow p_{a+1} > n^{\frac{1}{3}} \Rightarrow \frac{n}{p_{a+1}} < \frac{n}{n^{\frac{1}{3}}} = n^{\frac{2}{3}}$ y $\sqrt{\frac{n}{p_{a+1}}} < n^{\frac{1}{3}}$, el tiempo está en $\mathcal{O}\left(\frac{n}{p_{a+1}}\ln\ln n\right) = \mathcal{O}\left(n^{\frac{2}{3}}\ln\ln n\right)$ y la memoria está en $\mathcal{O}\left(\sqrt{\frac{n}{p_{a+1}}} + \pi(n^{\frac{1}{2}})\right) = \mathcal{O}\left(n^{\frac{1}{3}} + \pi(n^{\frac{1}{2}})\right)$, por Teorema 3.12.

Para el tiempo, el término dominante es el de $\phi(n,\pi(n^{\frac{1}{3}}))$, luego en el programa *piMeissel*, $t(n) \in \mathcal{O}(n) \cap \Omega\left(\frac{n}{(\ln n)^3}\right)$.

El espacio de almacenamiento para este programa es la memoria necesaria para obtener los primos hasta $\sqrt{n}$, $\phi(n,a)$ y $P_2(n,a)$. Por lo tanto, $M(n) \in \mathcal{O}\left(\frac{n^{\frac{1}{2}}}{\ln n} + \ln n + n^{\frac{1}{3}}\right) = \mathcal{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}\right)$, ya que por el Teorema de Chebyshev, 4.10: $\pi(n^{\frac{1}{2}}) \sim \frac{n^{\frac{1}{2}}}{\ln n^{\frac{1}{2}}} = 2\frac{n^{\frac{1}{2}}}{\ln n}$. □

3.7. Lehmer

3.7.1. Complejidad

El programa *piLehmer*, ver Sección F.11, obtiene $\pi(n)$ a partir de un parámetro de entrada, un natural n, aplicando el método de Lehmer, Teorema 2.11:

$$\pi(n) = a - 1 + \phi(n,a) - P_2(n,a) - P_3(n,a), \text{ con } a = \pi(n^{\frac{1}{4}}). \tag{3.9}$$

Teorema 3.15. *Para el algoritmo piLehmer,* $t(n) \in \mathcal{O}(n) \cap \Omega\left(\frac{n}{(\ln n)^4}\right)$ *y* $M(n) \in \mathcal{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}\right)$.

Demostración. Para obtener $\pi(n)$, siguiendo la fórmula (3.9), en primer lugar obtenemos y contamos los primos hasta $\sqrt{n}$ necesarios para calcular P_2. Esto supone un coste en tiempo $t(n) \in \mathcal{O}(\sqrt{n}\ln\ln\sqrt{n})$, y en memoria $M(n) \in \mathcal{O}(\pi(n^{\frac{1}{2}}))$, por Proposición 3.3.
Calculamos $\phi(n,\pi(n^{\frac{1}{4}}))$ con un coste: $t(n) \in \mathcal{O}(n) \cap \Omega\left(\frac{n}{(\ln n)^4}\right)$ y $M(n) \in \mathcal{O}(\pi(n^{\frac{1}{4}}) + \ln n)$, por Teorema 3.9.

Calculamos $P_2(n,\pi(n^{\frac{1}{4}}))$. Siendo $a=\pi(n^{\frac{1}{4}}) \Rightarrow p_{a+1}>n^{\frac{1}{4}} \Rightarrow \frac{n}{p_{a+1}}<\frac{n}{n^{\frac{1}{4}}}=n^{\frac{3}{4}}$ y $\sqrt{\frac{n}{p_{a+1}}}<n^{\frac{3}{8}}$, tenemos $t(n)\in\mathscr{O}\left(n^{\frac{3}{4}}\ln\ln n\right)$ y $M(n)=\mathscr{O}\left(\sqrt{\frac{n}{p_{a+1}}}+\pi(n^{\frac{1}{2}})\right)=\mathscr{O}\left(n^{\frac{3}{8}}+\pi(n^{\frac{1}{2}})\right)$, por Teorema 3.12.

Calculamos $P_3(n,\pi(n^{\frac{1}{4}}))$. Teniendo en cuenta que $\frac{n^{\frac{4}{3}}}{p_{a+1}^2}<n^{\frac{1}{2}}$ y $\sqrt{\frac{n}{p_{a+1}^2}}<n^{\frac{1}{4}}$, obtenemos que calcular P_3 tiene un coste: $t(n)\in\mathscr{O}\left(\frac{n^{\frac{4}{3}}}{p_{a+1}^2}\ln\ln n\right)=\mathscr{O}\left(n^{\frac{1}{2}}\ln\ln n\right)$ y $M(n)\in\mathscr{O}\left(\sqrt{\frac{n}{p_{a+1}^2}}+\pi\left(\sqrt{\frac{n}{p_{a+1}}}\right)\right)=\mathscr{O}\left(n^{\frac{1}{4}}+\pi(n^{\frac{3}{8}})\right)$, por Teorema 3.13.

Para el tiempo de ejecución, vemos que el término dominante es el de $\phi(n,a)$, por lo tanto, en el programa *piLehmer*, $t(n)\in\mathscr{O}(n)\cap\Omega\left(\frac{n}{(\ln n)^4}\right)$.

Por otro lado, la memoria del programa será el espacio requerido para obtener los primos hasta $\sqrt{n}$, $\phi(n,a)$, $P_2(n,a)$ y $P_3(n,a)$. Así, $M(n)\in\mathscr{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}+n^{\frac{1}{4}}+\ln n+n^{\frac{3}{8}}+n^{\frac{1}{4}}\right)=\mathscr{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}\right)$, por el Teorema de Chebyshev, 4.10. □

3.8. Conclusiones y resultados

En este trabajo hemos computado el valor de $\pi(n)$, la cantidad exacta de primos menores que n, por cuatro métodos distintos. En primer lugar, la criba de Eratóstenes que, pese a su antigüedad, sigue siendo competitivo para encontrar y contar los primos mencionados. Los siguientes métodos, Legendre, Meissel y Lehmer, son sucesivamente más eficientes y pueden expresarse de forma genérica como:

$$\pi(n)=a-1+\phi(n,a)-P_2(n,a)-P_3(n,a)-\cdots=a-1+\phi(n,a)+\sum_{m=2}^{k}P_m(n,a).$$

En la siguiente tabla resumen se recoge, para cada método estudiado, el programa que lo ha implementado con su respectiva complejidad, tiempo de ejecución y memoria de almacenamiento necesarios, así como las propiedades, es decir, el valor de a en ese método y hasta qué término de P_k aparece en la fórmula.

Método	**Propiedades**		**Complejidad**			**Programa**
	a	k	Tiempo	Memoria	Referencia	
Criba Eratóstenes	-	-	$\mathscr{O}(n\ln\ln n)$	$\mathscr{O}(\sqrt{n})$	3.4	*piEratótenes*
Legendre	$\pi(\sqrt{n})$	-	$\mathscr{O}(n)\cap\Omega\left(\frac{n}{(\ln n)^2}\right)$	$\mathscr{O}(\sqrt{n})$	3.11	*piLegendre*
Meissel	$\pi(\sqrt[3]{n})$	2	$\mathscr{O}(n)\cap\Omega\left(\frac{n}{(\ln n)^3}\right)$	$\mathscr{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}\right)$	3.14	*piMeissel*
Lehmer	$\pi(\sqrt[4]{n})$	3	$\mathscr{O}(n)\cap\Omega\left(\frac{n}{(\ln n)^4}\right)$	$\mathscr{O}\left(\frac{n^{\frac{1}{2}}}{\ln n}\right)$	3.15	*piLehmer*

Cuadro 3.1: Tabla comparativa de métodos.

En la gráfica siguiente se presenta el tiempo de ejecución que ha sido necesario para calcular $\pi(n)$ para diferentes valores de n con un procesador Intel Core i5 con 2.20 GHz de frecuencia. La dependencia del tiempo de ejecución en función de n se ajusta muy bien mediante una función lineal, que

es compatible con la predicha teóricamente, (ver Cuadro 3.1). Como se aprecia, el ahorro de tiempo es notable: el algoritmo de Lehmer es diez veces más rápido que el de Meissel, que a su vez es casi diez veces más rápido que el de Legendre, que es un 40 % más rápido que el de Eratóstenes.

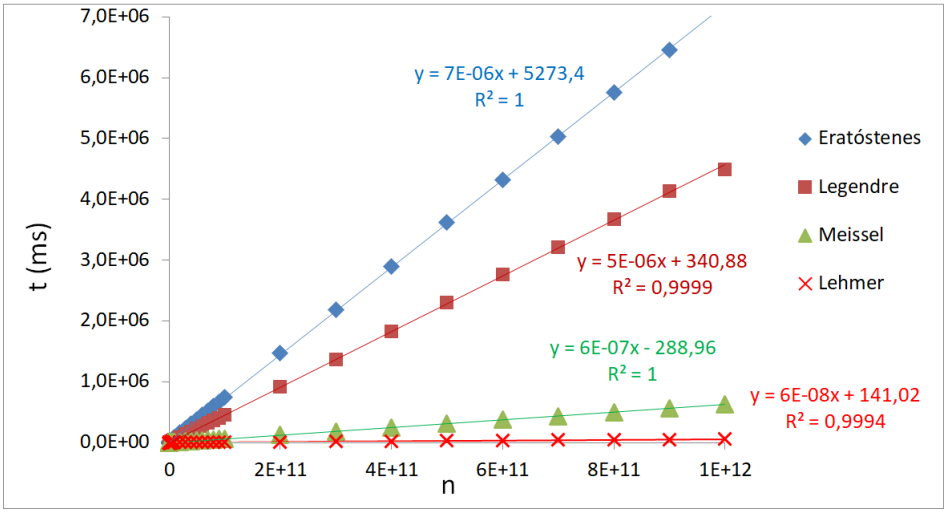

Figura 3.1: Gráfica t/n, para los cuatro métodos.

En el Apéndice D encontramos una tabla con resultados de $\pi(n)$ para diferentes valores de n, desde 1000 hasta $1 \cdot 10^{12}$. La tabla es producto de los programas diseñados en este trabajo. Los resultados son originales (aunque conocidos).

Capítulo 4

Distribución de los números primos. Métodos de Chebyshev

El objetivo de esta sección es obtener una cota superior e inferior para $\pi(n)$, para ello nos vamos a apoyar en las funciones de Chebyshev. En primer lugar, obtendremos las cotas de dichas funciones y veremos las cotas históricas que el propio Chebyshev desarrolló. A partir de ellas, deduciremos resultados como el postulado de Bertrand y, finalmente, obtendremos una cota superior e inferior para $\pi(n)$. La teoría y el procedimiento seguido en las demostraciones de este capítulo se han seguido del propio artículo de Chebysev, *Mémoire sur les Nombres premiers*, [9], y del libro [1].

A lo largo de esta sección vamos a considerar $x \in \mathbb{R}^+$.

4.0.1. Definición funciones de Chebyshev

Definición 4.1 (Función $\theta(x)$)**.** Para cada $x \geq 1$. definimos la función θ de Chebyshev:

$$\theta(x) = \sum_{p \leq x} \ln p. \text{ Con } \theta(x) = 0, \text{ para } x \in [1,\ 2),$$

donde p recorre todos los primos $\leq x$.

Definición 4.2 (Función $\psi(x)$)**.** Para cada $x \geq 1$, definimos la función ψ de Chebyshev:

$$\psi(x) = \sum_{p \leq x} \sum_{p^m \leq x} \ln p = \sum_{p \leq x} \ln p + \sum_{p^2 \leq x} \ln p + \cdots = \sum_{m \in \mathbb{N}} \theta(x^{\frac{1}{m}}). \tag{4.1}$$

La suma anterior es finita y además $p^m \leq x \Leftrightarrow m \leq \dfrac{\ln x}{\ln p} \Rightarrow m = \left[\dfrac{\ln x}{\ln p}\right]$, por tanto:

$$\psi(x) = \sum_{p \leq x} \left[\frac{\ln x}{\ln p}\right] \ln p.$$

Definición 4.3 (Función $T(x)$)**.** Para cada $x \geq 1$, definimos la función $T(x)$:

$$T(x) = \ln[x]! = \sum_{i \leq x} \ln i. \tag{4.2}$$

Proposición 4.1. *Sea $x \geq 1$,*

$$T(x) = \sum_{n \leq x} \psi\left(\frac{x}{n}\right) \tag{4.3}$$

Demostración. Descomponiendo en factores primos $[x]!$ y tomando logaritmos, resulta, $\ln[x]! = k_2 \ln 2 + k_3 \ln 3 + \cdots$, siendo k_i la mayor potencia con la que el primo p_i aparece en la descomposición de $[x]!$:

- La cantidad de términos en $\{1, 2, \ldots, [x]\}$ divisibles por p_i, es igual a $\left[\frac{x}{p_i}\right]$.
- La cantidad de términos en $\{1, 2, \ldots, [x]\}$ divisibles por p_i^{α}, es igual a $\left[\frac{x}{p_i^{\alpha}}\right]$.

Por lo tanto, en la descomposición de $\ln[x]!$ se tiene: $k_i = \left[\frac{x}{p_i}\right] + \left[\frac{x}{p_i^2}\right] + \cdots + \left[\frac{x}{p_i^{m_i}}\right]$, donde m_i es tal que, $p_i^{m_i} \leq [x]$, es decir, $m_i = \left[\frac{\ln[x]}{\ln p_i}\right]$ $\forall\ p_i \leq x$.

Por otro lado,

$$\begin{aligned}\sum_{n\leq x} \psi\left(\frac{x}{n}\right) = \sum_{m\in\mathbb{N}} \theta\left(\left(\frac{x}{n}\right)^{\frac{1}{m}}\right) =& \theta(x) + \theta\left(\frac{x}{2}\right) + \theta\left(\frac{x}{3}\right) + \theta\left(\frac{x}{4}\right) + \cdots \\ &+ \theta(\sqrt{x}) + \theta\left(\sqrt{\frac{x}{2}}\right) + \theta\left(\sqrt{\frac{x}{3}}\right) + \theta\left(\sqrt{\frac{x}{4}}\right) + \cdots \\ &+ \theta(\sqrt[3]{x}) + \theta\left(\sqrt[3]{\frac{x}{2}}\right) + \theta\left(\sqrt[3]{\frac{x}{3}}\right) + \theta\left(\sqrt[3]{\frac{x}{4}}\right) + \cdots \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\end{aligned} \tag{4.4}$$

va a ser de la forma: $k_2 \ln 2 + k_3 \ln 3 + \cdots$, siendo k_i el número de veces que aparece $\log p_i$ en la suma:

- El número k de veces que aparece $\ln p_i$ en los sumandos $\left\{\theta(x), \theta\left(\frac{x}{2}\right), \theta\left(\frac{x}{3}\right), \ldots, \theta\left(\frac{x}{k}\right), \ldots\right\}$, es tal que $\frac{x}{k} \geq p_i$, es decir, $k = \left[\frac{x}{p_i}\right]$.
- El número k de veces que aparece $\ln p_i$ en los sumandos $\left\{\theta(\sqrt[\alpha]{x}), \theta\left(\sqrt[\alpha]{\frac{x}{2}}\right), \ldots\right\}$ es $\left[\frac{x}{p_i^{\alpha}}\right]$.

Por lo tanto, en $\sum_{n\leq x} \psi\left(\frac{n}{x}\right)$, cada $p_i \leq x$ aparece $k_i = \left[\frac{x}{p_i}\right] + \left[\frac{x}{p_i^2}\right] + \cdots + \left[\frac{x}{p_i^{m_i}}\right]$ veces, donde m_i es tal que, $\sqrt[m_i]{x} \geq p_i \Leftrightarrow p_i^{m_i} \leq [x] \Leftrightarrow m_i = \left[\frac{\ln[x]}{\ln p_i}\right]$. □

4.0.2. Acotaciones asintóticas para las funciones de Chebyshev

Teorema 4.2 (Acotaciones para la función $T(x)$)**.**

$$x\ln x - x - \frac{1}{2}\ln x + \frac{1}{2}\ln(2\pi) < T(x) < x\ln x - x + \frac{1}{2}\ln x + \frac{1}{2}\ln(2\pi) + \frac{1}{12},\ \forall\ x \geq 2. \tag{4.5}$$

Demostración. Denotamos $n = [x]$.

Aplicando la fórmula de Stirling (1730): $n! \approx n^n e^{-n}\sqrt{2\pi n}\left(1 + \frac{\theta_n}{12n}\right)$, $0 < \theta_n < 1$ y tomando logaritmos:

$$\ln n! = n\ln n - n + \frac{1}{2}\ln(2\pi n) + \frac{\theta_n}{12n},\ \text{con } 0 < \theta_n < 1.$$

Podemos expresar $T(x) = \ln n! = \ln(n+1)! - \ln(n+1)$. De esta manera, aplicando la igualdad anterior con $0 < \theta_n < 1$ y considerando la función $y^{\pm}(x) = x\ln x - x \pm \frac{1}{2}\ln x$, resulta:

$$\begin{aligned}T(x) = \ln n! &< n\ln n - n + \frac{1}{2}\ln n + \frac{1}{2}\ln(2\pi) + \frac{1}{12n} = y^{+}(n) + \frac{1}{2}\ln(2\pi) + \frac{1}{12n} \\ T(x) = \ln(n+1)! - \ln(n+1) &> (n+1)\ln(n+1) - (n+1) - \frac{1}{2}\ln n + \frac{1}{2}\ln(2\pi) = y^{-}(n+1) + \frac{1}{2}\ln(2\pi)\end{aligned} \tag{4.6}$$

Tenemos $n=[x] \Rightarrow n \le x < n+1$ y la función $y^{\pm}(x)$ se comprueba trivialmente que es creciente $\forall\, x \ge 2$, de aquí, aplicándolo a (4.6) obtenemos el resultado buscado. □

Teorema 4.3 (Acotaciones para la función $\psi(x)$)**.**

$$(\ln 2)\,x - \frac{3}{2}\ln x + \left(-\frac{1}{2}\ln 2\,\pi + \ln 2 - \frac{1}{6}\right) < \psi(x) < (2\ln 2)\,x - \frac{3}{4\ln 2}(\ln x)^2 + \left(\frac{1-6\ln 2\,\pi}{12\ln 2} - 1\right)\ln x. \tag{4.7}$$

Demostración. Consideramos la función

$$V(x) = T(x) - 2T\left(\frac{x}{2}\right), \tag{4.8}$$

aplicando la fórmula (4.3) de $T(x)$, resulta:

$$V(x) = \psi(x) - \psi\left(\frac{x}{2}\right) + \psi\left(\frac{x}{3}\right) - \psi\left(\frac{x}{4}\right) + \cdots$$

La función $\psi(x)$ es monótona no decreciente, de aquí y de la fórmula anterior de $V(x)$, es inmediato deducir las desigualdades: $\psi(x) - \psi\left(\frac{x}{2}\right) \le V(x) \le \psi(x)$, y de ellas:

$$\begin{cases} \psi\left(\frac{x}{2}\right) - \psi\left(\frac{x}{4}\right) \le V\left(\frac{x}{2}\right) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \psi\left(\frac{x}{2^m}\right) - \psi\left(\frac{x}{2^{m+1}}\right) \le V\left(\frac{x}{2^m}\right) \end{cases} \tag{4.9}$$

El valor de m se toma hasta que $\frac{x}{2^{m+1}} < 2 \le \frac{x}{2^m}$, es decir, tomando logaritmos, hasta $m = \left[\frac{\ln x}{\ln 2}\right] - 1$. Por tanto, $\psi\left(\frac{x}{2^m}\right) \ne 0,\ \psi\left(\frac{x}{2^{m+1}}\right) = 0$.

Sumando las desigualdades de (4.9), obtenemos:

$$V(x) \le \psi(x) \le V(x) + V\left(\frac{x}{2}\right) + \cdots + V\left(\frac{x}{2^m}\right). \tag{4.10}$$

A continuación hallamos una acotación asintótica para $V(x)$ aplicando las cotas de $T(x)$ y $T\left(\frac{x}{2}\right)$, (4.5), a la definición de $V(x)$, (4.8):

$$x\ln 2 - \frac{3}{2}\ln x + \left(\ln 2 - \frac{1}{2}\ln 2\,\pi - \frac{1}{6}\right) < V(x) < x\ln 2 + \frac{3}{2}\ln x + \left(-\ln 2 - \frac{1}{2}\ln 2\,\pi + \frac{1}{12}\right).$$

De aquí, aplicando (4.10), sale directamente la cota inferior de $\psi(x)$ y finalmente, obtenemos la cota superior de $\psi(x)$ con esta cota de $V(x)$ y (4.10), siendo $m = \left[\frac{\ln x}{\ln 2}\right] - 1$. □

Observación 4.4. *En particular, podemos afirmar que* $\psi(x) \in \mathcal{O}(x)$.

Teorema 4.5 (Acotaciones para la función $\theta(x)$)**.**

$$(\ln 2)\,x - 4\ln 2\sqrt{x} - \frac{3}{8\ln 2}(\ln x)^2 + \left(-\frac{1}{2} + \frac{6\ln 2\pi - 1}{12\ln 2}\right)(\ln x) + \left(-\frac{1}{2}\ln 2\,\pi - \frac{1}{6} + \ln 2\right) < \theta(x)$$
$$\theta(x) < (2\ln 2)\,x - 4\ln 2\sqrt{x} - \frac{3}{8\ln 2}(\ln x)^2 + \left(-\frac{1}{2} + \frac{6\ln 2\,\pi - 1}{12\ln 2}\right)(\ln x) + \left(-\frac{1}{2}\ln 2\,\pi - \frac{1}{6} + \ln 2\right). \tag{4.11}$$

Demostración. Aplicando la fórmula (4.1) de $\psi(x)$, obtenemos las siguientes igualdades:

$$\begin{aligned}\psi(x)-\psi(\sqrt{n}) &= \theta(x)+\theta(\sqrt[3]{x})+\theta(\sqrt[5]{x})+\cdots\\ \psi(x)-2\,\psi(\sqrt{n}) &= \theta(x)-\theta(\sqrt{x})+\theta(\sqrt[3]{x})+\cdots\end{aligned} \tag{4.12}$$

La función $\theta(x)$ es monótona no decreciente y positiva, de aquí y de las igualdades anteriores, obtenemos:

$$\psi(x)-\psi(\sqrt{n}) \leq \theta(x) \leq \psi(x)-2\,\psi(\sqrt{n}).$$

Aplicando a esta cota la acotación de $\psi(x)$ y $\psi(\sqrt{x})$, (4.7), tenemos el resultado deseado. □

Observación 4.6. *En particular, se puede decir que* $\theta(x) \in \mathcal{O}(x)$.

Teorema 4.7 (Postulado de Bertrand)**.** *Entre x y $2x$ siempre hay al menos un número primo,* $\forall\ x \geq 6.5\cdot 10^{11}$.

Demostración. Para demostrar que en el intervalo $(x,\ 2x]$ hay al menos un primo es suficiente probar que $\theta(2x)-\theta(x)=\sum_{x<p\leq 2x}\log p>0$. Aplicando las cotas de $\theta(x)$ y $\theta(2x)$, (4.11), obtenemos:

$$\theta(2x)-\theta(x) > (1-4\sqrt{2})\ln 2\sqrt{x}-\frac{9}{8\ln 2}(\ln x)^2-\ln x+\left(\frac{9}{8}\ln 2-\frac{1}{2}\ln 2\pi-\frac{5}{12}\right).$$

Estudiando gráficamente la cota obtenida, ver Sección E.1, obtenemos que es positiva $\forall\ x \geq 6.5\ \cdot 10^{11}$. □

Este resultado es cierto $\forall\ x \geq 6.5\cdot 10^{11}$, el objetivo es que sea $\forall\ x$, por ello hemos implementado un algoritmo para verificar el postulado de Bertrand $\forall\ x < 6.5\cdot 10^{11}$. Para ello, presentamos la base del razonamiento del algoritmo:

Observación 4.8 (Idea del algoritmo)**.** *Este algortimo, a partir de un natural x, confirma si existe un primo entre n y* $2n$, $\forall\ n \leq x$ *y* $\forall\ x \geq 8$. *Ver Sección F.12, Algoritmo postuladoBertrand.*

En primer lugar comprobamos explícitamente la existencia de un primo en $(n,\ 2n]\ \forall\ n \leq 2\sqrt{x}$. *A continuación, basándonos en la observación de que si tenemos dos naturales,* x_0 *y* L *tal que* $L \leq x_0$, *si existe* $p \in [x_0,\ x_0+L]$, *entonces,* $\forall\ y \in \left[\frac{x_0+L}{2},\ x_0\right],\ p \in [y,\ 2y]$. *Estudiamos intervalos de longitud* $L=\sqrt{n}$; *de modo que si* $\exists\ p \in [aL,\ (a+1)L]$, *se asegure que* $\forall\ y \in \left[\frac{(a+1)L}{2},\ aL\right],\ \exists\ p \in [y,\ 2y]$.

4.0.3. Acotaciones asintóticas históricas

La acotación obtenida para $\psi(x)$, y por tanto la cota de $\theta(x)$, el postulado de Bertrand y el Teorema de Chebyshev, que se verá a continuación, están directamente condicionados por la elección de la función $V(x)$. El procedimiento de la demostración es el mismo independientemente de la elección de $V(x)$. En nuestro caso hemos trabajado con $V(x)=T(x)-2\,T\left(\frac{x}{2}\right)$ para tener demostraciones más cortas, a cambio de obtener peores cotas numéricas. Esto hace que la versión presentada del postulado de Bertrand sea cierta $\forall\ x > 6.5\cdot 10^{11}$, un valor muy grande. Este problema lo hemos solventado creando un programa que nos verificase el resultado $\forall\ x < 6.5\cdot 10^{11}$.

En la demostración original, [9], Chebyshev toma

$$V(x)=T(x)-T\left(\frac{x}{2}\right)-T\left(\frac{x}{3}\right)-T\left(\frac{x}{5}\right)+T\left(\frac{x}{30}\right),$$

esto hace que se obtengan las siguientes acotaciones válidas para todo $x \geq 160$,

$$Ax-\frac{5}{2}\ln x-1<\psi(x)<\frac{6}{5}Ax+\frac{5}{4\ln 6}(\ln x)^2+\frac{5}{4}\ln x+1,\ \text{con}\ A=\ln\frac{2^{1/2}3^{1/3}5^{1/5}}{30^{1/3}}=0.92129202\ldots \tag{4.13}$$

$$Ax-\frac{12}{5}A\sqrt{x}-\frac{5}{8\ln 6}(\ln x)^2-\frac{15}{4}\ln x-3<\theta(x)<\frac{6}{5}Ax-A\sqrt{x}+\frac{5}{4\ln 6}(\ln x)^2+\frac{5}{2}\ln x+2. \tag{4.14}$$

Los cambios en $V(x)$ dan mejores cotas de $\psi(x)$ y $\theta(x)$, se consigue obtener $A=0.9212920\ldots$, en vez de $A=\ln 2\approx 0.6931\ldots$, que es el valor obtenido en el Teorema 4.3. Lo que se ve reflejado en el postulado de Bertrand y el Teorema de Chebyshev, se consigue demostrar: "Entre x y $2x$ siempre hay al menos un número primo, $\forall\, x\geq 160$". Como vemos, se obtiene una mejor cifra.

Podemos mencionar que los mejores resultados obtenidos, según cita [1], se han conseguido en [2]. Se obtienen constantes y resultados más precisos que Chebyshev, como $A=0.998405\ldots$, con la elección de $V(x)$:

$$V(x)=T(x)-T\left(\frac{x}{2}\right)-T\left(\frac{x}{3}\right)-T\left(\frac{x}{5}\right)+T\left(\frac{x}{6}\right)-T\left(\frac{x}{7}\right)+T\left(\frac{x}{10}\right)-T\left(\frac{x}{11}\right)+T\left(\frac{x}{2310}\right).$$

4.0.4. Teorema de Chebyshev

En este apartado vamos a trabajar con las cotas de Chebyshev para $\psi(x)$ y $\theta(x)$, (4.13), (4.14).

Proposición 4.9. *Sea $x\geq 2$ y $A=0.9212920\ldots$,*

$$A\frac{x}{\ln x}-\frac{1}{\ln x}-\frac{5}{2}<\pi(x)<\frac{12}{5}A\left(\frac{x}{\ln x}\right)+\sqrt{x}-2A\left(\frac{\sqrt{x}}{\ln x}\right)+\frac{5}{2\ln 6}\ln x+\frac{4}{\ln x}+5. \tag{4.15}$$

Demostración. Por la definición de $\psi(x)$ tenemos:

$$\psi(x)=\sum_{p\leq x}\left[\frac{\ln x}{\ln p}\right]\ln p<\sum_{p\leq x}\ln x=\ln x\,\pi(x)\ \Rightarrow\ \pi(x)>\frac{\psi(x)}{\ln x}.$$

Aplicando la acotación inferior de $\psi(x)$ de Chebyshev, obtenemos la acotación inferior de $\pi(x)$.

Por la definición de $\theta(x)$, tenemos:

$$\theta(x)>\sum_{\sqrt{x}<p\leq x}\ln p>\frac{1}{2}\ln x\left(\pi(x)-\pi(\sqrt{x})\right)\ \Rightarrow\ \pi(x)<\frac{2\,\theta(x)}{\ln x}+\sqrt{x},\ \text{ya que}\ \pi(\sqrt{x})<\sqrt{x}.$$

Tomando la acotación superior para $\theta(x)$ de Chebyshev, obtenemos la acotación superior de $\pi(x)$. □

Teorema 4.10 (Teorema de Chebyshev)**.** *Existen unas constantes reales $0<\overline{A}<\overline{B}$ tal que*

$$\overline{A}\frac{x}{\ln x}<\pi(x)<\overline{B}\frac{x}{\ln x},\ \forall\, x\geq 2.$$

Observación 4.11. *Con las cotas anteriores,* (4.15)*, resulta que podemos tomar $\overline{A}=0.835$, $\overline{B}=2.211$, $\forall\, x\geq 160$. En particular, $\pi(x)\in\mathscr{O}\left(\frac{x}{\ln x}\right)$.*

Demostración. Dividiendo (4.15) por $\frac{x}{\ln x}$, resulta la acotación superior e inferior de $\overline{A}<\dfrac{\pi(x)}{\frac{x}{\ln x}}<\overline{B}$.

$$\pi(x)>A\frac{x}{\ln x}-\frac{1}{\ln x}-\frac{5}{2}>\overline{A}\frac{x}{\ln x}$$

$$\frac{\pi(x)}{\frac{x}{\ln x}}<\frac{12}{5}A-\frac{2A}{\sqrt{x}}+\frac{5}{2\ln 6}\frac{(\ln x)^2}{x}+\frac{5\ln x}{x}+\frac{4}{x}+\frac{\ln x}{\sqrt{x}}<\overline{B}.$$

Por métodos elementales obtenemos las constantes $\overline{A}$ y $\overline{B}$ mencionadas $\forall\, x\geq 160$. Ver Sección E.2. □

Bibliografía

[1] E. APARICIO, *Teoría de los números*, Servicio Editorial UPV, 1993.

[2] E. APARICIO, *"P.L. Chebishev y los números primos" Actas de las IV Jornadas Matemáticas Hispano-Lusitanas, Jaca, mayo 1977*, Universidad de Zaragoza, 1980. 157-182.

[3] T.M. APOSTOL, *Introducción a la teoría analítica de números*, Reverté, 1984.

[4] G.BRASSARD Y P. BRATLEY, *Fundamentos de Algoritmia*, Prentice Hall, 2002.

[5] H. DEITEL Y P. DEITEL, *C++ how to program*, Prentice Hall, 2004.

[6] M. DELEGLISE Y J. RIVAT, *"Computing $\pi(x)$: the Meissel, Lehmer, Lagarias, Miller, Odlyzko metho"*, Mathematics of Computation of the American Mathematicial Society **65**, 213 (1996), 235-245.

[7] W. NARKIEWICZ, *The Development of Prime Number Theory: From Euclid to Hardy and Littlewood*, Springer Science & Business Media, 2000.

[8] H. RIESEL, *Prime numbers and computer methods for factorization*, Springer Science & Business Media, 2012.

[9] M. TCHEBYSHEF, *"Mémoire sur les nombres premiers"*, Journal de mathématiques pures et appliquées 1^{re} série, **17** (1852), 366-390.

Apéndice A

Nociones de complejidad

El contenido de esta sección está tomado de [4].

En este capítulo se introducen conceptos de complejidad necesarios a lo largo de este trabajo. Comenzamos introduciendo la idea de algoritmo como un conjunto finito de operaciones, organizadas de manera lógica y ordenada, diseñadas para resolver un problema, efectuar un cálculo.

En general, un problema puede ser resuelto por numerosos algoritmos; nos interesa seleccionar aquel que haga un uso más eficiente de los recursos, esencialmente, tiempo de ejecución y memoria de almacenamiento. Luego, entre algoritmos correctos, preferimos el que requiera de menor tiempo de ejecución y memoria.

Para caracterizar formalmente la eficiencia introducimos la noción de complejidad como la cantidad de recursos necesarios para cada algoritmo. Para presentarlo necesitamos definir: ejemplar de un problema, tamaño y operación elemental.

- Un problema tiene en general infinitos ejemplares o casos, por ejemplo, 289 + 89 ó 4952 + 1052 son ejemplares del problema de la suma.
- Caracterizamos el tamaño de un ejemplar como la cantidad de bits que son necesarios para representarlos.
- Definimos operación elemental como aquella cuyo tiempo de ejecución se puede acotar superiormente por una constante que depende solamente de la implementación usada, como puede ser el lenguaje de programación o la máquina utilizada.

A continuación vamos a introducir la notación asintótica. Para proporcionar cotas superiores sobre la cantidad de recursos requeridos haremos uso de la notación $\mathcal{O}(f(n))$, "$\mathcal{O}$ de $f(n)$". Sea $f : \mathbb{N} \to \mathbb{R}^+$ una función arbitraria, se indicará $\mathcal{O}(f(n))$ como el conjunto de todas las funciones $t : \mathbb{N} \to \mathbb{R}^+$ tal que $t(n) \leq c\, f(n)\ \forall\ n \geq n_0$, para una constante real positiva c y para un umbral entero n_0. Se dirá que $t(n)$ es del orden de $f(n)$, $t(n) \in \mathcal{O}(f(n))$.

$$\mathcal{O}(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ |\ \exists\, c \in \mathbb{R}^+,\ \forall\, n \geq n_0,\ n_0 \in \mathbb{N} \text{ tq } t(n) \leq c\, f(n)\}$$

Necesitamos una notación dual para cotas inferiores, por ello, presentamos la notación $\Omega(f(n))$, Ω de $f(n)$. Sean $f,\ t : \mathbb{N} \to \mathbb{R}^+$, diremos que $t(n)$ está en "Ω de $f(n)$", $t(n) \in \Omega(f(n))$, si $t(n)$ está acotada superiormente por un múltiplo real positivo de $f(n)$, para todo n suficientemente grande.

$$\Omega(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ |\ \exists\, d \in \mathbb{R}^+,\ \forall\, n \geq n_0,\ n_0 \in \mathbb{N} \text{ tq } t(n) \geq d f(n)\}$$

Es claro que cuando analizamos un algoritmo deseamos que los recursos estén acotados superior e interiormente, por ello presentamos finamente la notación $\Theta(f(n))$, "Θ de $f(n)$". Sean $f,\ t : \mathbb{N} \to \mathbb{R}^+$,

diremos que $t(n)$ está en el orden exacto de $f(n)$, $t(n) \in \Theta(f(n))$, si $t(n) \in \mathcal{O}(f(n))$ y $t(n) \in \Omega(f(n))$, es decir, $\Theta(f(n)) = \mathcal{O}(f(n)) \cap \Omega(f(n))$.

$$\Theta(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ | \; \exists \, c, \, d \in \mathbb{R}^+, \; \forall \, n \geq n_0, \; n_0 \in \mathbb{N} \text{ tq } d\, f(n) \leq t(n) \leq c\, f(n)\}$$

Caracterizar la complejidad de un algoritmo consiste en buscar una función $f : \mathbb{N} \to \mathbb{R}^+$, tal que $f(n)$ acote los recursos requeridos por el algoritmo para resolver un ejemplar de tamaño n. Por recursos entendemos a la cantidad de operaciones elementales o de bits, según atendamos al tiempo de ejecución o a la memoria.

Finalmente debemos señalar que una técnica para calcular la complejidad de un algoritmo es identificar una sentencia barómetro, definida como una instrucción elemental que se ejecuta por lo menos con tanta frecuencia como cualquier otra del algoritmo. Es inmediato demostrar que el tiempo de ejecución del algoritmo es del orden exacto del número de veces que se ejecute dicha sentencia barómetro. Siempre que el tiempo requerido por cada instrucción esté acotado por una constante, el tiempo empleado por el algoritmo es del orden exacto del número de veces que se ejecuta la instrucción barómetro.

Apéndice B

Cantidad de primos menores que n. Anexo del Capítulo 2

Ejemplo B.1. Presentamos un ejemplo del método de Legendre, Teorema 2.5, para $n = 100$:

$$\pi(100) = \pi(10) - 1 + \phi(100, \pi(\sqrt{100})) =$$

$$= 4 - 1 + 100 - \sum_{1\le i\le 4} \left[\frac{n}{p_i}\right] + \sum_{1\le i<j\le 4} \left[\frac{n}{p_i p_j}\right] - \sum_{1\le i<j<k\le 4} \left[\frac{n}{p_i p_j p_k}\right] + \left[\frac{n}{2\cdot 3\cdot 5\cdot 7}\right] =$$

$$= 4 - 1 + 100 - \left[\frac{n}{2}\right] - \left[\frac{n}{3}\right] - \left[\frac{n}{5}\right] - \left[\frac{n}{7}\right] + \left[\frac{n}{2\cdot 3}\right] + \left[\frac{n}{2\cdot 5}\right] + \left[\frac{n}{2\cdot 7}\right] + \left[\frac{n}{3\cdot 5}\right] + \left[\frac{n}{3\cdot 7}\right] + \left[\frac{n}{5\cdot 7}\right] -$$

$$- \left[\frac{n}{2\cdot 3\cdot 5}\right] - \left[\frac{n}{2\cdot 3\cdot 7}\right] - \left[\frac{n}{3\cdot 5\cdot 7}\right] + \left[\frac{n}{2\cdot 3\cdot 5\cdot 7}\right] = 4 - 1 + 22 - 0 + 0 = 25.$$

Apéndice C

Algoritmos. Anexo del Capítulo 3

Demostraciones de las Poposiciones 3.1 y 3.2.

Proposición C.1.

$$\sum_{i=2}^{a} \frac{1}{i \ln i} \in \Theta(\ln\ln a).$$

Demostración. Se puede expresar el sumatorio como una integral:

$$\sum_{i=2}^{a} \frac{1}{i \ln i} = \int_1^a \frac{dx}{\lceil x \rceil \ln\lceil x \rceil}.$$

Dado que $\frac{1}{(x+1)\ln(x+1)} \leq \frac{1}{\lceil x \rceil \ln\lceil x \rceil} \leq \frac{1}{x \ln x}\ \forall\, x \geq 1$, tenemos:

$$\int_1^a \frac{dx}{\lceil x \rceil \ln\lceil x \rceil} > \int_1^a \frac{dx}{(x+1)\ln(x+1)} = \int_2^{a+1} \frac{dy}{y \ln y} = \ln\ln(a+1) - \ln\ln 2 > \ln\ln a. \ (\ln\ln 2 < 0)$$

Por otro lado,

$$\int_1^a \frac{dx}{\lceil x \rceil \ln\lceil x \rceil} < \frac{1}{2\ln 2} + \int_2^a \frac{dx}{x \ln x} = \frac{1}{2\ln 2} + \ln\ln a - \ln\ln 2 < 2\ln\ln a \ \ \forall a \geq 20$$

ya que $\ln\ln 20 = 1.0972 > \frac{1}{2\ln 2} - \ln\ln 2 = 1.0878$, y la función $\ln\ln a$ es estrictamente creciente en a. □

Proposición C.2.

$$\sum_{i=1}^{a} \frac{1}{p_i} \in \Theta(\ln\ln a)$$

Demostración. Del Teorema de Chebyshev, 4.10, se sigue que existen reales A y B con $0 < A < B$ tales que $A\frac{x}{\ln x} < \pi(x) < B\frac{x}{\ln x}$, luego

$$A\frac{p_i}{\ln p_i} < i < B\frac{p_i}{\ln p_i} \Rightarrow \frac{A}{i \ln p_i} < \frac{1}{p_i} < \frac{B}{i \ln p_i} < \frac{B}{i \ln i}, \tag{C.1}$$

donde la última desigualdad se debe a que $\forall\, i\ p_i > i$. Así pues,

$$\sum_{i=1}^{a} \frac{1}{p_i} = \frac{1}{2} + \sum_{i=2}^{a} \frac{1}{p_i} < \frac{1}{2} + B\sum_{i=2}^{a} \frac{1}{i \ln i} < C\sum_{i=2}^{a} \frac{1}{i \ln i} \tag{C.2}$$

donde para $a \geq 2$, basta con tomar $C = B + 1$ ya que $\frac{1}{2} < \frac{1}{2\ln 2} \leq \sum_{i=2}^{a} \frac{1}{i \ln i}$.

Por otro lado, tomando logaritmos en (C.1):

$$\frac{\ln i}{\ln p_i} > 1 + \frac{\ln A - \ln\ln p_i}{\ln p_i}$$

Dado que $\lim_{i\to\infty} \frac{\ln A - \ln\ln p_i}{\ln p_i} = 0$, existe un valor i_m tal que $\frac{\ln A - \ln\ln p_i}{\ln p_i} > -\frac{1}{2}$ para $i > i_m$. Así, para valores de i lo suficientemente grandes:

$$\sum_{i=1}^{a} \frac{1}{p_i} = \frac{1}{2} + \sum_{i=2}^{a} \frac{1}{p_i} > A \sum_{i=2}^{a} \frac{1}{i \ln p_i} = A \sum_{i=2}^{a} \frac{1}{i \ln p_i} \frac{\ln i}{\ln i} > \frac{A}{2} \sum_{i=2}^{a} \frac{1}{i \ln i} \tag{C.3}$$

El Resultado C.2 se sigue de aplicar el Resultado C.1 a las ecuaciones (C.2) y (C.3). □

Demostración del Resultado 3.8.

Resultado C.3.

$$Si\ a > \frac{m^2}{m-1} \Rightarrow \binom{a}{m} > \frac{a^m}{m!\, m^m}.$$

Demostración. Observamos que $a > \frac{m^2}{m-1} \Leftrightarrow a - m > \frac{a}{m}$, entonces,

$$\binom{a}{m} = \frac{a(a-1)(a-2)\cdots(a-m+1)}{m!} > \frac{(a-m)^m}{m!} > \left(\frac{a}{m}\right)^m \frac{1}{m!} = \frac{a^m}{m^m\, m!}.$$ □

Apéndice D

Tabla con valores de $\pi(n)$

Tabla con diversos resultados de $\pi(n)$ para valores de n entre 1000 y $1 \cdot 10^{12}$, obtenidos con los programas desarrollados en este trabajo.

n	$\pi(n)$	n	$\pi(n)$	n	$\pi(n)$
1000	168	$1 \cdot 10^6$	78498	$1 \cdot 10^9$	50847534
2000	303	$2 \cdot 10^6$	148933	$2 \cdot 10^9$	98222287
3000	430	$3 \cdot 10^6$	216816	$3 \cdot 10^9$	144449537
4000	550	$4 \cdot 10^6$	283146	$4 \cdot 10^9$	189961812
5000	669	$5 \cdot 10^6$	348513	$5 \cdot 10^9$	234954223
6000	783	$6 \cdot 10^6$	412849	$6 \cdot 10^9$	279545368
7000	900	$7 \cdot 10^6$	476648	$7 \cdot 10^9$	323804352
8000	1007	$8 \cdot 10^6$	539777	$8 \cdot 10^9$	367783654
9000	1117	$9 \cdot 10^6$	602489	$9 \cdot 10^9$	411523195
10000	1229	$1 \cdot 10^7$	664579	$1 \cdot 10^{10}$	455052511
20000	2262	$2 \cdot 10^7$	1270607	$2 \cdot 10^{10}$	882206716
30000	3245	$3 \cdot 10^7$	1857859	$3 \cdot 10^{10}$	1300005926
40000	4203	$4 \cdot 10^7$	2433654	$4 \cdot 10^{10}$	1711955433
50000	5133	$5 \cdot 10^7$	3001134	$5 \cdot 10^{10}$	2119654578
60000	6057	$6 \cdot 10^7$	3562115	$6 \cdot 10^{10}$	2524038155
70000	6935	$7 \cdot 10^7$	4118064	$7 \cdot 10^{10}$	2925699539
80000	7837	$8 \cdot 10^7$	4669382	$8 \cdot 10^{10}$	3325059246
90000	8713	$9 \cdot 10^7$	5216954	$9 \cdot 10^{10}$	3722428991
100000	9592	$1 \cdot 10^8$	5761455	$1 \cdot 10^{11}$	4118054813
200000	17984	$2 \cdot 10^8$	11078937	$2 \cdot 10^{11}$	8007105059
300000	25997	$3 \cdot 10^8$	16252325	$3 \cdot 10^{11}$	11818439135
400000	33860	$4 \cdot 10^8$	21336326	$4 \cdot 10^{11}$	15581005657
500000	41538	$5 \cdot 10^8$	26355867	$5 \cdot 10^{11}$	19308136142
600000	49098	$6 \cdot 10^8$	31324703	$6 \cdot 10^{11}$	23007501786
700000	56543	$7 \cdot 10^8$	36252931	$7 \cdot 10^{11}$	26684074310
800000	63951	$8 \cdot 10^8$	41146179	$8 \cdot 10^{11}$	30341383527
900000	71274	$9 \cdot 10^8$	46009215	$9 \cdot 10^{11}$	33981987586
–	–	–	–	$1 \cdot 10^{12}$	37607912018

Cuadro D.1: Tabla de resultados de $\pi(n)$.

Apéndice E

Distribución de los números primos. Métodos de Chebyshev. Anexo del Capítulo 4

E.1. Postulado de Bertrand

Demostración gráfica del postulado de Bertrand, Teorema 4.7.

Teorema E.1 (Postulado de Bertrand)**.** *Entre x y $2x$ siempre hay al menos un número primo, $\forall\, x \geq 6.5 \cdot 10^{11}$.*

Demostración gráfica.

$$\theta(2x)-\theta(x) > (1-4\sqrt{2})\ln 2\sqrt{x} - \frac{9}{8\ln 2}(\ln x)^2 - \ln x + \left(\frac{9}{8}\ln 2 - \frac{1}{2}\ln 2\,\pi - \frac{5}{12}\right) > 0,\ \forall\, x \geq 6.5\cdot 10^{11}:$$

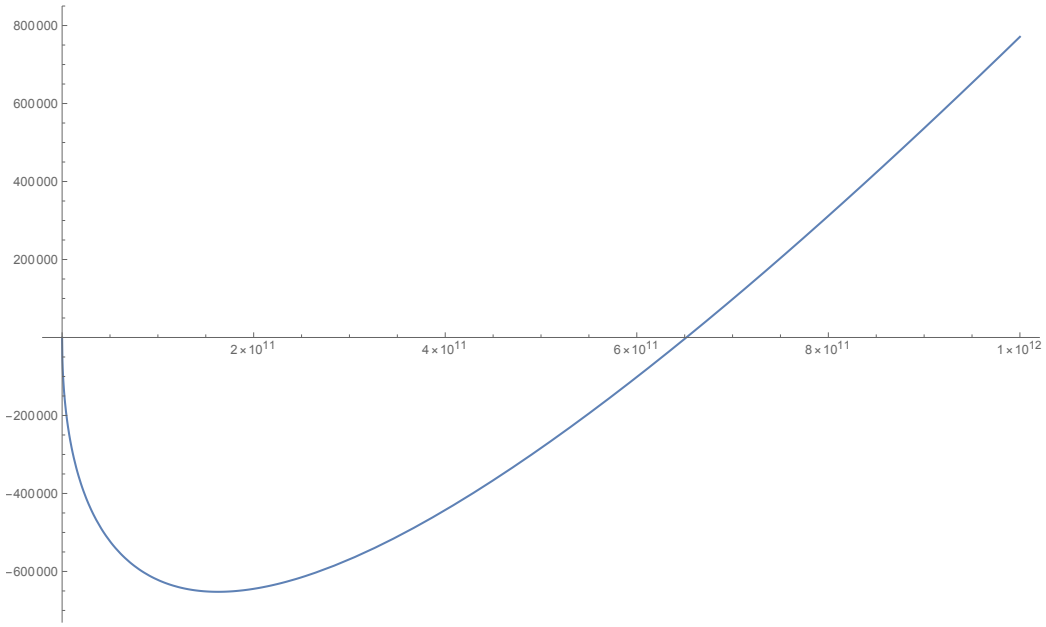

Figura E.1: Gráfica de la cota inferior de $\theta(2x) - \theta(x)$.

□

E.2. Teorema de Chebyshev

Demostración del Teorema de Chebyshev, 4.10, y la Observación 4.11.

Teorema E.2 (Teorema de Chebyshev)**.** *Existen unas constantes reales* $0 < \overline{A} < \overline{B}$ *tal que*

$$\overline{A}\,\frac{x}{\ln x} < \pi(x) < \overline{B}\,\frac{x}{\ln x},\ \forall\, x \geq 2.$$

Observación E.3. *Con las cotas anteriores,* (4.15)*, resulta que podemos tomar* $\overline{A} = 0.835$, $\overline{B} = 2.211$, $\forall\, x \geq 160$. *En particular,* $\pi(x) \in \mathcal{O}\left(\frac{x}{\ln x}\right)$.

Demostración. • Acotación inferior y estimación de $\overline{A}$.

Aplicando la acotación (4.15) de $\pi(x)$ y tomando $f(x) = (\overline{A} - A)\,\frac{x}{\ln x} - \frac{1}{\ln x} - \frac{5}{2}$, resulta:

$$\pi(x) > A\,\frac{x}{\ln x} - \frac{1}{\ln x} - \frac{5}{2} > \overline{A}\,\frac{x}{\ln x} \;\Leftrightarrow\; f(x) > 0.$$

El objetivo es obtener $f(x) > 0\ \forall\ x > x_0$, con un buen valor de $\overline{A}$ y de x_0, es decir, A próximo e inferior a 1 y x_0 no demasiado grande. Para ello, estudiamos gráficamente $f(x)$ para diferentes valores de $\overline{A}$. Obtenemos que para $\overline{A} \approx 0.835,\ f(x) > 0\ \forall\, x \geq 160$.

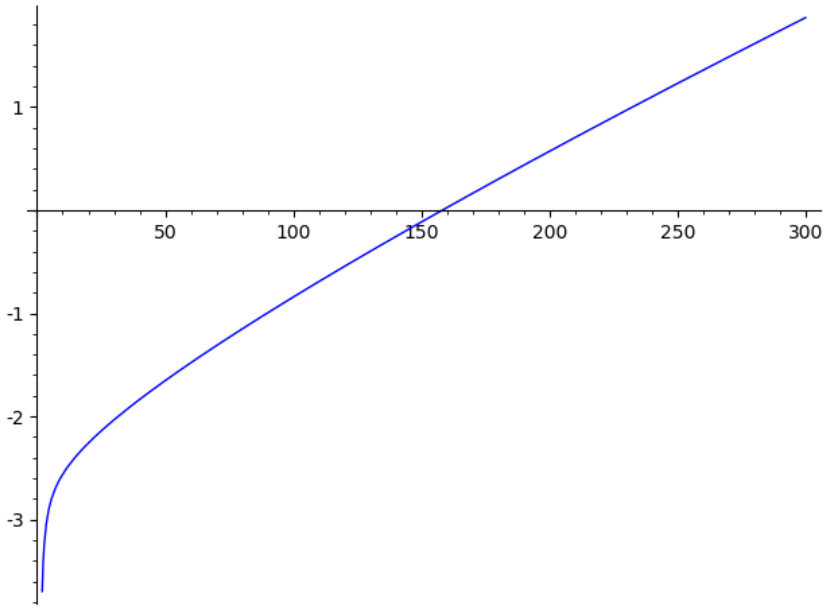

Figura E.2: Gráfica $f(x)$ con $\overline{A} \approx 0.835$.

• Acotación superior y estimación de $\overline{B}$.

Dividiendo (4.15) por $\frac{x}{\ln x}$ y denotando $g(x) = \frac{12}{5}A - \frac{2A}{\sqrt{x}} + \frac{5}{2\ln 6}\,\frac{(\ln x)^2}{x} + \frac{5\ln x}{x} + \frac{4}{x} + \frac{\ln x}{\sqrt{x}}$, resulta:

$$\frac{\pi(x)}{\frac{x}{\ln x}} < \frac{12}{5}A - \frac{2A}{\sqrt{x}} + \frac{5}{2\ln 6}\,\frac{(\ln x)^2}{x} + \frac{5\ln x}{x} + \frac{4}{x} + \frac{\ln x}{\sqrt{x}} = g(x) < \overline{B}.$$

Se busca un valor para $\overline{B}$ próximo y superior a 1 tal que $g(x) < \overline{B}\ \forall\ x > x_0$, para un x_0 no demasiado grande. Estudiando gráficamente la función $g(x)$ vemos que en $(2.5,\ 5.52)$ tiene un máximo absoluto, por lo tanto, $g(x) < 5.52\ \forall\ x$. No podemos aceptar este resultado ya que el valor de $\overline{B}$ es demasiado grande. Por otro lado, $\lim_{x\to\infty} g(x) = \frac{12A}{5} \approx 2.211$, y en particular, podemos observar que $g(x) < 2.875\ \forall\, x \geq 160$.

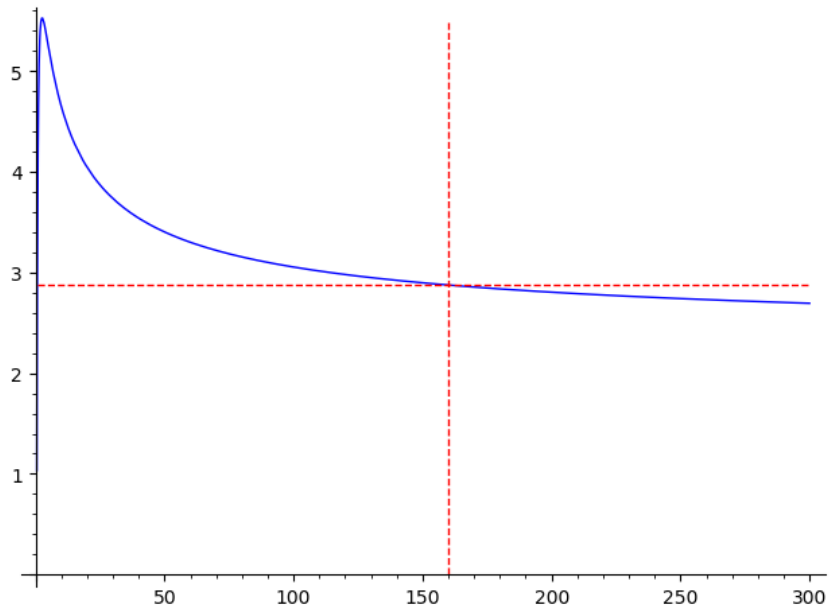

Figura E.3: Gráfica $g(x)$.

□

Apéndice F

Programas

En este Apéndice recogemos los programas implementados en Java a lo largo del trabajo. Los desarrollados en el Capítulo 3 y el programa desarrollado en el Capítulo 4 para el postulado de Bertrand.

F.1. *cribaEratóstenes*

```
/**
 *Dado un entero no negativo n devuelve un BitSet indicando los primos en [3,n]
 *implementando la criba de Eratóstenes.
 *@param n un entero no negativo
 *@return  Un BitSet de 1's y 0's que codifica los primos en el intervalo [3,n].
 *                El i-ésimo bit representa el número 3+2*i.
 *                El índice i vale 1 sii mi es primo, y 0 en caso contrario.
 */

 static BitSet cribaEratóstenes (int n){

 int iMax =(n-3)/2;                //Último índice del BitSet, índice máximo
 int L = iMax+1;                   //Longitud cadena=índice máximo+1

 BitSet listado = new BitSet (L);  //Creamos un BitSet de 1's de longitud L
 int tamaño = listado.size();
 listado.set(0, tamaño, false);    //Por seguridad, inicializamos a 0
 listado.set(0, L);                //Inicializamos nuevamente con 1's

 int ip = 0;                       //ip índice que recorre el listado
 int ifinal= (int)((sqrt(n)-3)/2); //p<=sqrt(n) -> índices: ifinal
 int p, j;

 //Estudiamos los primos p<=sqrt(n) -> corresponde en índices: ip<=ifinal.
 //Tomamos dichos primos y quitamos sus múltiplos<=n.
 while (ip<=ifinal)
 {
 //Si es 0, no es primo y no lo estudiamos más. Pasamos al siguiente índice
 while (listado.get(ip)==false)
 {
 ip=ip+1;
 }
 p=3+2*ip;         //Número correspondiente al índice ip: 3+2*ip
```

```
 j=(p*p-3)/2;   //Primer múltiplo aún no cribado de p: p^2 -> índice: j=(p*p-3)/2

 //Recorremos BitSet mientras el múltiplo del primo sea <=n -> índice: j<=iMax
 while(j<=iMax)
 {
 listado.set(j,false); //En el índice de los correspondientes múltiplos ponemos 0
 j=j+p;                //Múltiplos de p: p^2+p, p^2+2p ... -> índices: j+p, j+2p ...
 }
 ip=ip+1;
 }
 return listado;
 }
```

F.2. *cadenaPrimos*

```
/**
*Construye un array de enteros a partir de un BitSet.
*@param listado un BitSet de 1's y 0's
*@return Un array de enteros.
*       En particular, si se cumple:
*       El BitSet de entrada codifica con 1's y 0's los primos en [3,n],
*       donde el i-ésimo bit representa el número 3+2*i y
*       el índice i vale 1 sii mi es primo, y 0 en caso contrario,
*       devuelve un array con los primos en [2,n]
*/

static int[] cadenaPrimos (BitSet listado){

//PROGRAMA EXPLICADO PARA EL CASO PARTICULAR. CASO QUE NOS INTERESA.
int [] listaPrimos = new int[listado.cardinality()+1]; //Longitud=nº 1's en listado+1
listaPrimos[0]=2;             //Añadir directamente el 2 en la 1ª componente
int t=0;                      //t índice recorre listado
int k=1;                      //k recorre listaPrimos desde 1, listaPrimos[0]=2
while (t<=listado.length()){
if (listado.get(t)==true){
listaPrimos[k]=3+2*t;         //nº corrrespondiente al índice t -> 3+2*t
k++;
}
t++;
}
return listaPrimos;
}
```

F.3. *subCribaIntervalo*

```
/**
*@param listaPrimos secuencia de N enteros {p1, p2,...,pN}
*@param n0 el extremo inferior del intervalo que se desea cribar
*@param nF el extremo superior del intervalo que se desea cribar
*@return La cantidad de números en el intervalo [n0, nF] que
*          NO son múltiplos de ninguno de los nº del listado <= sqrt(nF)
*          En particular, si se cumple:
*          listaPrimos contiene todos los primos hasta sqrt(nF) y
*          n0 > sqrt(nF),
*          el valor devuelto es la cantidad de primos en el intervalo [n0, nF]
*/

static int subCribaIntervalo (int[] listaPrimos, long n0, long nF){

//PROGRAMA EXPLICADO PARA EL CASO PARTICULAR. CASO QUE NOS INTERESA.
/* nMin es el primer número impar a partir de n0:
* si n0 es par tomamos el siguiente impar: nMin=n0+1 y si es impar: nMin=n0.
* Interesan los impares de ese intervalo: el bit i-ésimo representa
*el nº mi=nMin+3*i.
*/
long nMin = n0 + 1-(n0%2);        //Si n0 es par tomamos el siguiente impar
if (nF<nMin)  return 0;           //Si nF<nMin devolver: nº de primos es 0
int L = (int) ((nF - nMin)/2 +1); //Longitud cadena final de impares en [n0, nF]

BitSet listado = new BitSet (L);  //Creamos cadena inicializada con 1's, longitud L
listado.set (0, listado.size(), false);
listado.set(0, L);

int k;                            //k índice que recorre la cadena listaPrimos
int primo, resto;
long primerMultiplo;
int r;                            //r índice que recorre listado
int j0;
int nPrimos;                      //Contabiliza la cantidad de primos en [n0,nF]
int sqrtNF = (int) sqrt(nF);    //Interesa listaPrimos hasta sqrt(nF) o primo anterior
int kFinal = listaPrimos.length; //El número de primos en listaPrimos

//Recorremos listaPrimos mientras:
//el primo pertenezca a listaPrimos -> índice: 1<=k<kFinal y el primo sea <=sqrt(nF).
//Quitamos los múltiplos<=nF de los primos en listaPrimos <=sqrt(nF)
for(k=1; k<kFinal && listaPrimos[k]<=sqrtNF  ; k++)
{
primo=listaPrimos[k];             //primo(k+1)-ésimo

//¿Cuál es el primer múltiplo impar de primo?
resto = (int) (nMin%primo);
primerMultiplo =  nMin + (primo - resto)%primo;

//Si el 1º múltiplo es par, tomar el siguiente múltiplo ->
//primerMultiplo = primerMultiplo + primo
```

```
if (primerMultiplo %2==0) primerMultiplo = primerMultiplo + primo;
//j0 índice correspondiente al primer múltiplo impar
j0 =(int)((primerMultiplo - nMin)/2);

//En cada cadena recorremos los L números quitando los múltiplos ->
//índices: j0+p, j0+2p,...
for(r=j0; r<L; r+= primo)
{
listado.set(r,false);             //Ponemos 0 en los múltiplos del primo
}
}
nPrimos = listado.cardinality();  //nPrimos=cantidad de 1's en listado
return nPrimos;
}
```

F.4. *cribaIntervalo*

```
/**
*Criba los enteros del intervalo [n0, nF] utilizando un listado de primos.
*@param listaPrimos secuencia de N enteros {p1, p2,...,pN}
*@param n0 el extremo inferior del intervalo que se desea cribar
*@param nF el extremo superior del intervalo que se desea cribar
*@return la cantidad de números en el intervalo [n0, nF] que
*          NO son múltiplos de ninguno de los n° del listado <= sqrt(nF)
*          En particular, si se cumple:
*          listaPrimos contiene todos los primos hasta sqrt(nF) y
*          n0 > sqrt (nF),
*          el valor devuelto es la cantidad de primos en el intervalo [n0, nF]
*/

static int cribaIntervalo (int[] listaPrimos, long n0, long nF){

//PROGRAMA EXPLICADO PARA EL CASO PARTICULAR. CASO QUE NOS INTERESA.
//Este programa criba el intervalo [n0, nF] aplicando subCribaIntervalo
//en tramos de longitud sqrt(nF)

int sqrtNF = (int) sqrt(nF);       //Se estudian tramos de sqrt(nF) números
int ntramos= (int)(nF-n0)/sqrtNF;  //Número de tramos de longitud sqrt(nF)
int  sumaPrimos=0;                 //contabiliza los primos en cada tramo
int i;
for(i=0; i<ntramos; i++)
{
sumaPrimos = sumaPrimos + subCribaIntervalo(listaPrimos, n0+i*sqrtNF,
 n0+(i+1)*sqrtNF-1);
}

//Cuando i>=nTramos salimos del for, con ese i
//calculamos el último tramo: [n0+i*sqrtNF, nF]
sumaPrimos = sumaPrimos + subCribaIntervalo(listaPrimos, n0+i*sqrtNF, nF);
return sumaPrimos;
}
```

F.5. *piEratóstenes*

```
/**
*Dado un entero positivo n, calcula la cantidad de primos <= n
*aplicando la criba de Eratóstenes.
*@param n un entero no negativo
*@return pi(n) = cantidad de primos en [2,n]
*/

/* Comentario: dado que el tipo de n es long,
 valor máximo permitido para n es 2^63-1 */

static long piEratóstenes (long n){

//Obtenemos la lista de primos hasta n^1/2
int L = (int)(sqrt(n));
BitSet listado= cribaEratóstenes(L);
int[] listaPrimos=cadenaPrimos(listado);

//Obtenemos pi(n^1/2)=nPrimos
long nPrimos = listaPrimos.length;

//Cribamos los primos en (n^1/2, n], para ello:
//generamos intervalos de longitud L=n^1/2 y aplicamos cribaIntervalo.
//La i_ésima cadena representa los L números:
//[(i+1)*L+1, (i+2)*L] = [n0, n0+L-1], para cada nuevo n0
long n0;
for (n0= L+1; n0+L<=n ; n0=n0+L)
{
nPrimos = nPrimos + cribaIntervalo(listaPrimos, n0, n0+L-1);
}

//Cuando el extremo superior del intervalo verifica: n0+L>n, salimos del for,
//con ese n0 estudiamos el último intervalo [n0,n]
nPrimos = nPrimos + cribaIntervalo(listaPrimos, n0, n);
return nPrimos;
}
```

F.6. *sumaPhi*

```
/**
*@param n un entero no negativo
*@param w un entero no negativo
*@param listaPrimos secuencia de N enteros {p1, p2,...,pN}
*@return  Un entero.
*            En el caso particular en que listaPrimos contenga al menos
*            los primos hasta w, devuelve:
*            la cantidad de enteros positivos <= n,
*            que no son divisibles por ninguno de los primos <= w
*/
```

```
static long sumaPhi(long n, int w, int[] listaPrimos){

//PROGRAMA EXPLICADO PARA EL CASO PARTICULAR. CASO QUE NOS INTERESA.
//Obtenemos pi(w)=a
int a, z=0;
while(z<listaPrimos.length && listaPrimos[z]<=w){
z++;
}
a=z;

/*¿Cuántos términos de listaPrimos vamos a necesitar en sumaPhi?
En listaPrimos tenemos almacenados los primos hasta w.
Nos iteresan los sumandos no nulos, es decir, los sumandos |n/pi|>0:
los pi en [p1,...,pa] tq n>pi.
*/

/*¿Cuántos términos de listaPrimos vamos a necesitar simultánemanete en sumaPhi?
Tomamos s, índice donde se acumula el producto de primos.
Nos iteresan los sumandos no nulos, es decir, los sumandos tq
|n/(pi1*...*pimax)|>0 -> n>pi1*...*pimax.
Vamos a obtener el último índice imax tq |n/(pi1*...*pimax)|>0:
recorremos los primos con listaPrimos[imax], desde imax=0 hasta imax<a=pi(w)
y vemos cuál es el último pimax verificando: s= pi1*pi2*...*pimax< n -->
imax es la máxima cantidad de primos diferentes tq su producto es menor que n.
*/
long s;
int imax;
for (imax=0, s=1; s<n && imax < a; imax++) s*=listaPrimos[imax];

//iTermino, array con los sumandos necesarios para calcular el siguiente
long[] itermino= new long[imax+1];
itermino[0]=n;                    //El primer término lo fijamos -> n

//iPrimo, arraw con los índices de los primos necesarios para calcular
/cada sumando,
//listaPrimos[iPrimo[i]]= primo i-ésimo = listaPrimos[i-1]
int[] iPrimo = new int[imax+1];  //imax+1 pq dejamos primera compoennte vacía
iPrimo[1]=0;                      //Fijamos el primer primo p1=2

//suma almacena los sumandos. Fijamos primer sumando=n
long suma= n;
int i=1;                          //Se comienza en i=1 -> p1=2
while(i>0){                       //i índice que recorre itermino e iPrimo
//factor=sumando= -(sumando anterior)*(1/nuevo primo)
long factor = -itermino[i-1] / listaPrimos[iPrimo[i]];
suma = suma + factor;             //suma va almacenando
itermino[i]=factor;               //itermino[i]=factor con el que trabajamos

//Si |factor|<n y el primo<=w, es decir, iPrimo[i]<a-1,avanzamos,
//añadimos un primo:
//Análogo con iTérmino, añadimos el nuevo sumando
```

```
if  (abs(factor) > 0 && iPrimo[i]<a-1){
i++;                          //Avanzamos, añadimos un nuevo primo
iPrimo[i]=iPrimo[i-1]+1;      //Añadimos el siguiente primo

}
//Sino, retrocedemos, quitamos un primo y en el último que se queda ponemos
//su siguiente.
//Análogo con iTérmino, quito el último y en el actual último pongo el nuevo sumando
else{
i--;                          //Retrocedemos, quitamos el último primo
iPrimo[i]=iPrimo[i]+1;        //En el alctual último primo, ponemos su siguiente
}
}
return suma;
}
```

F.7. *piLegendre*

```
/**
*Dado un entero positivo n, calcula la cantidad de primos <= n
*aplicando la fórmula de Legendre.
*@param n un entero no negativo
*@return pi(n) = sumaPhi(n,n^1/2,listaPrimos) + pi(n^1/2) - 1
*/

/* Comentario: dado que el tipo de n es long,
el valor máximo permitido para n es 2^63-1*/

static long piLegendre (long n){

//Obtenemos la lista de primos hasta n^1/2
int sqrtn = (int) sqrt(n);
BitSet listado = cribaEratóstenes (sqrtn);
int[] listaPrimos= cadenaPrimos(listado);

//Obtenemos a=pi(n^1/2)
int a = listaPrimos.length;

//SOLUCION
long sL= sumaPhi (n,sqrtn,listaPrimos);
return sL - 1 + a; //pi(n)= sumaPhi(n,sqrtn,listaPrimos) + pi(sqrtn) - 1
}
```

F.8. $P_2(n,a)$

```
/**
*@param n un entero no negativo
*@param a un entero no negativo
*@param listaPrimos secuencia de N enteros {p1, p2,...,pN}
*@return  Un entero.
*           En el caso particular en que listaPrimos contenga al menos
*           los primos hasta n^1/2, devuelve:
*           la cantidad de enteros positivos <= n, que son producto
*           de dos primos >= pa+1, no necesariamente distintos, es decir,
*           la cantidad de enteros x=pi*pj <= n con pa+1 <= pi <= pj.
*/

/* Comentario:  P2(n,a)!=0 sii a es elegido tq  pa < n^1/2.
Si a es elegido tq pa+1> n^1/2, es claro que pipj> n^1/2*n^1/2= n si pa+1<=pi<=pj
Luego, es natural tomar a < pi(n^1/2) para no tener nigún término nulo en P2*/

static long P2(long n, long a, int[] listaPrimos){

//PROGRAMA EXPLICADO PARA EL CASO PARTICULAR. CASO QUE NOS INTERESA.
//Obtenemos b=pi(n^1/2), siempre es asi para cualquier n
int sqrtn = (int)Math.sqrt(n);
int b, zb=0;
while(zb<listaPrimos.length && listaPrimos[zb]<=sqrtn){
zb++;
}
b=zb;

//Obtener el sumatorio
long p;
long itermino=b;                //Inicializamos: itermino=pi(sqrtn)=b
long sumatorio=0;
long n0= (long) Math.pow(n, 1./2)+1;
long nF;

//Vamos en orden decreciente de primos: pb, pb-1,..., pa+2, pa+1, es decir,
//en orden creciente de los términos: n/pb, n/pb-1,...,n/pa+1.
//Inicializamos itermino=pi(sqrtn)=b, ya que el primer término:
//pi(n/pb)=pi(sqrtn)+(primos en [sqrtn+1, n/pb])
for (int i=b; i>=a+1 ;i--){
p=listaPrimos[i-1];         //p=i-ésimo primo
nF=n/p;
//itermino = pi(n/pj) = pi(n/pj+1) + cantidad de primos en [n/pj+1, n/pj]
itermino= itermino + cribaIntervalo(listaPrimos, n0, nF);
sumatorio = sumatorio + itermino;
n0=nF+1;
}
return sumatorio - (b-a)*(b+a-1)/2;
}
```

F.9. $P_3(n,a)$

```
**
*@param n un entero no negativo
*@param a un entero no negativo
*@param listaPrimos secuencia de N enteros {p1, p2,...,pN}
*@return  Un entero.
*           En el caso particular en que listaPrimos contenga al menos
*           los N primeros primos hasta pi(n^1/3), devuelve:
*           la cantidad de enteros positivos <= n, que son producto
*           de tres primos >= pa+1, no necesariamente distintos, es decir,
*           la cantidad de enteros x=pi*pj*pk <= n con pa+1 <= pi <= pj <= pk.
*/

/* Comentario: P3(n,a)!=0 sii a es elegito tq pa < n^1/3.
Si a es elegido tq pa+1> n^1/3, es claro que pipjpk> n^1/3*n^1/3*n^1/3= n si
pa+1<pi<=pj<=pk.
Luego, es natural tomar a<pi(n^1/3), par ano tener términos nulos en P3*/

static long P3(long n,long a, int[] listaPrimos){

//PROGRAMA EXPLICADO PARA EL CASO PARTICULAR. CASO QUE NOS INTERESA.
//Obtenemos c=pi(n^1/3), siempre es asi para cualquier n
int sqrt3= (int) Math.pow(n, 1./3);
int c, zc=0;
while(zc<listaPrimos.length &&listaPrimos[zc]<=sqrt3 ){
zc++;
}
c=zc;

//Obtenemos el sumatorio
long p, sumatorio=0;
long aux = a;             //El primer sumando del sumatorio comienza con aux=a

//Empezamos por el menor primo: pa+1 hasta sqrt3.
for(int i=(int)(a+1); i<=(c);i++){
p=listaPrimos[i-1];       //p=i_ésimo primo
sumatorio=sumatorio + P2((long)(n/p) , aux, listaPrimos);
aux++;
}
return sumatorio;
}
```

F.10. *piMeissel*

```
/**
*Dado un entero positivo n, calcula la cantidad de primos <= n
*aplicando la fórmula de Meissel.
*@param n un entero no negativo
*@return pi(n)= sumaPhi(n,a,listaPrimos) + a + 1 - P2(n,a), a=pi(n^1/3)
*/

/* Comentario: dado que el tipo de n es long,
el valor máximo permitido para n es 2^63-1 */

static long piMeissel(long n){
//Obtenemos la lista de primos hasta n^1/2
BitSet listado=cribaEratóstenes((int) sqrt(n));
int[] listaPrimos=cadenaPrimos(listado);

//Obtenemos a=pi(n^1/3)
int a, za=0;
while(za<listaPrimos.length && listaPrimos[za]<=Math.pow(n, 1./3)){
za++;
}
a=za;

//SOLUCION
long sumaFi;
sumaFi = sumaPhi((long) n, (int) Math.pow(n, 1./3), listaPrimos);
long P2;
P2=P2((long) n, (long) a, listaPrimos);
//pi(n)= sumaPhi(n,a,listaPrimos) + a + 1 - P2(n,a)
return  sumaFi+a-1-P2;
}
```

F.11. *piLehmer*

```
/**
*Dado un entero positivo n, calcula la cantidad de primos <= n
*aplicando la fórmula de Lehmer.
*@param n un entero no negativo
*@return pi(n)= sumaPhi(n,a,listaPrimos) + a + 1 - P2(n,a) - P3(n,a), a=pi(n^1/4)
*/

/* Comentario: dado que el tipo de n es long,
el valor máximo permitido para n es 2^63-1 */

static long piLehmer(long n){

//Obtenemos la lista de primos hasta n^1/2
BitSet listado=cribaEratóstenes((int) Math.pow(n, 1./2));
int[] listaPrimos=cadenaPrimos(listado);
```

```
//Obtenemos a=pi(n^1/4)
int a, za=0;
while(za<listaPrimos.length && listaPrimos[za]<=Math.pow(n, 1./4)){
za++;
}
a=za;

//SOLUCION
long sumaFi;
sumaFi = sumaPhi((long) n, (int) Math.pow(n, 1./4), listaPrimos);
long P2;
P2=P2((long) n, a, listaPrimos);
long P3;
P3=P3((long) n,a, listaPrimos);
//pi(n)= sumaPhi(n,a,listaPrimos) + a + 1 - P2(n,a) - P3(n,a)
return  sumaFi+a-1-P2-P3;
}
```

F.12. *postuladoBertrand*

```
/**
*Dado un entero positivo n, afirma si existe un primo p en [x,2x],
*para todo x en [2,n]
*@param n un entero no negativo
*@return verdad si existe un primo p en [x,2x] para todo x en [2,n]
*/

static boolean bertrand(long n){

//cada i_ésima cadena tiene L núm: [(i+1)*L+1, (i+2)*L]
int L;
L = (int)(sqrt(n));
//Generamos el BitSet de 1's y 0's hasta sqrt(n)
BitSet miCriba= cribaEratostenes(L);
//Generamos los primos en [2,L] como un array de enteros
int[] miCribaPrimos=cadenaPrimos(miCriba);

//Compruebo explicitamente si existe un primo p en [x,2x] para todo x<=sqrt(n):
//Intervalo [2,4] existe un primo p=3, no lo detecta pq sqrt(4)=2 ->
//luego lo añadimos directamnte
//Si existe p en el intervalo [i,2i], suma 1 a nexiste.
//Si nexiste=L/2-1 --> hay un primo en cada [x,2x] hasta 2*sqrt(n)
for (int i=3; i<=2*L;i++){
if (cuentaPrimos(miCribaPrimos, i,2*i)==0)  return false;
}

//Compruebo si existe p en [x0, x0+L] --> para todo y en [x0+L/2, x0],
//existe un p perteneciente a [y,2y]
long a;
for(a=3; a<=L; a=2*a-1){
```

```
if (cuentaPrimos(miCribaPrimos, a*L,(a+1)*L)==0){
return false;
}
}
System.out.println("Estudio el último intervalo ["+ a*L+" , "+ (a+1)*L+"]");
if (cuentaPrimos(miCribaPrimos, a*L ,(a+1)*L )==0){
return false;
}
return true;
}
```