ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

[DΣIM]

Metodologies de la Programació
Pràctica 1

Pràctica avaluable 1
Número Primer més Gran no Superior a un Altre
curs 2021-22

Estudiantes:	Marc Fonseca (GEI)
	Nil Monfort (GEI)
	Miguel Rodriguez (GEI)
Professor/a:	Simeó Reig Pelleja
Grupo:	3

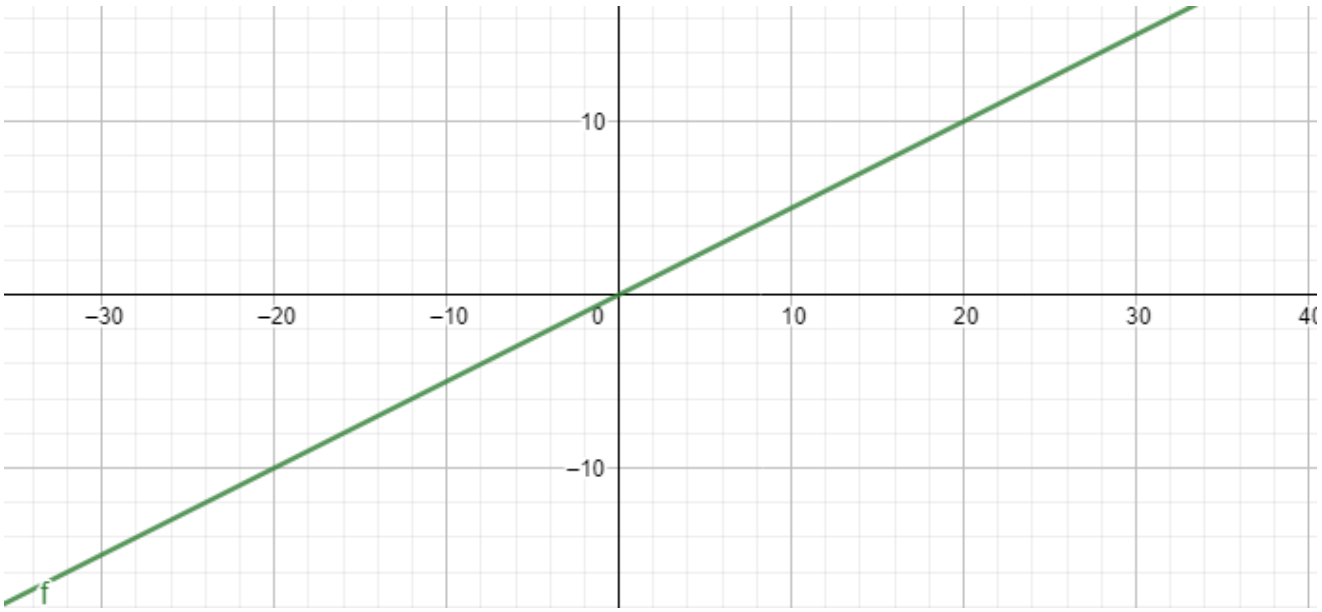
Diferentes estrategias que podemos tratar

"Nota: Las unidades de los gráficos són: en el eje X tamaño del parámetro que se recibe y en el eje Y tiempo de ejecución en términos del Big-O"

Version 1.

```java
public static boolean esNumeroPrimer_v1(long num) {

    int numeroDivisors = 0;

    // El nÃºmero u Ã©s NO primer per conveni, encara que nomÃ©s es divisible per
    // ell mateix i per u solament.
    if (num >= 2) {

        for (long i = 1; i <= num; i++) {
            if (num % i == 0) {
                numeroDivisors++;
            }
        }

        return (numeroDivisors == 2);

    } else {
        return false;
    }
}
```

O(n)



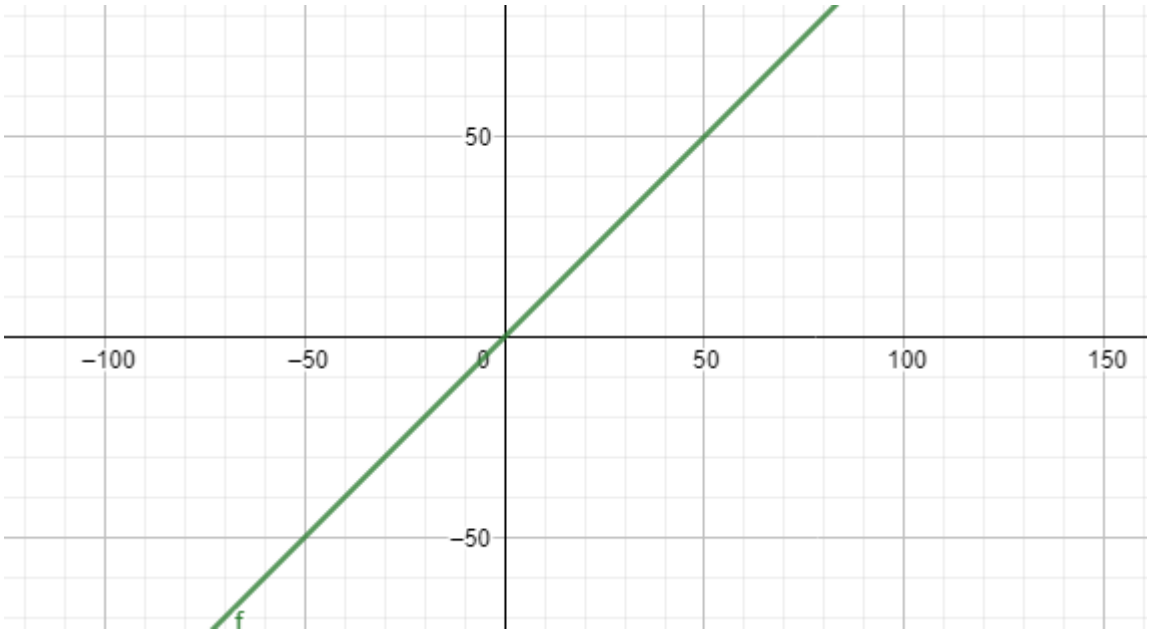
Prueba	Descripción	Resultado obtenido
1	1	0,478 ms
2	2	0,005 ms
3	3	0,008 ms
4	4	0,011 ms
5	2017	0,219 ms
6	2026	0,276 ms
7	2147483601L	194108,811 ms
8	2147485783L	mucho tiempo
9	9223372036854775807L	mucho tiempo
10	9223372036854775808	imposible calcular (OverFlow)

Podemos ver que en la versión 1 la complejidad es lineal ya que el coste va a depender del número que se pasa por parámetro. Este algoritmo tiene la desventaja de que con valores grandes el tiempo de ejecución aumenta en gran medida.

Version 2.

```java
public static boolean esNumeroPrimer_v2(long num) {

    boolean esPrimer = false;

    // El nÃºmero u Ã©s NO primer per conveni, encara que nomÃ©s es divisible per
    // ell mateix i per u solament.
    if (num >= 2) {

        esPrimer = true;

        for (long i = 2; i < num; i++) {
            if (num % i == 0) {
                esPrimer = false;
                break;
            }
        }

    }

    return esPrimer;
}
```

O(n/2)



Prueba	Descripción	Resultado obtenido
1	1	0,535 ms
2	2	0,005 ms
3	3	0,011 ms
4	4	0,015 ms
5	2017	0,092 ms
6	2026	0,095 ms
7	2147483601L	21984,873 ms
8	2147485783L	23131,578 ms
9	92233720368547758O7L	mucho tiempo
10	9223372036854775808	imposible calcular (OverFlow)

El coste de este algoritmo es similar al de la versión 1 ya que la única diferencia es que en esta versión trabaja con un booleano y no con un contador de divisores para declarar si el número es primo.

Version 3.

```
public static boolean esNumeroPrimer_v3(long num) {

    boolean esPrimer = false;

    if (num >= 2) {

        if ((num == 2) || (num == 3)) {
            esPrimer = true;
        } else {

            if (num % 2 != 0) {

                esPrimer = true;

                for (long i = 3; i < num / 2; i = i + 2) {
                    if (num % i == 0) {
                        esPrimer = false;
                        break;
                    }
                }
            }
        }
    }

    return esPrimer;
}
```

O(n/4)

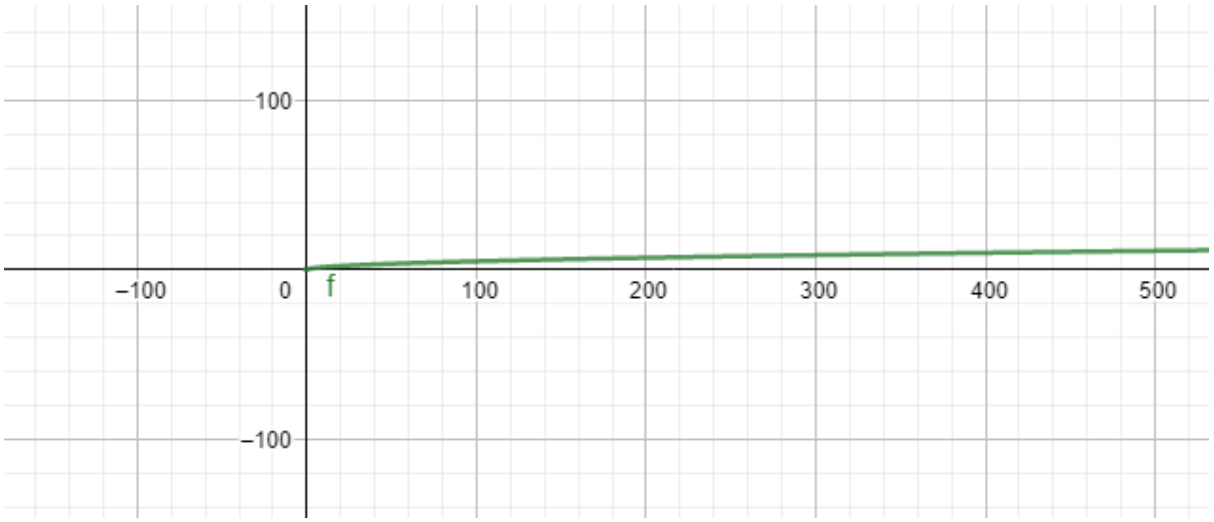
Prueba	Descripción	Resultado obtenido
1	1	**0,807 ms**
2	2	**0,007 ms**
3	3	0,012 ms
4	4	0,015 ms
5	2017	0,099 ms
6	2026	0,087 ms
7	2147483601L	5415,215 ms
8	2147485783L	5401,370 ms
9	92233720368547758O7L	mucho tiempo
10	9223372036854775808	imposible calcular (OverFlow)

El coste de esta versión es menor que las anteriores ya que solo recorremos hasta la mitad del número y solo comprobamos los números impares en este rango.

Version 4.

```java
public static boolean esNumeroPrimer_v4(long num) {

    boolean esPrimer = false;
    long arrelDeN = (long) Math.sqrt((double) num) + 1; // Sumem 1 per evitar problemes d'arrodoniment

    if (num >= 2) {

        if ((num == 2) || (num == 3)) {
            esPrimer = true;
        } else {

            if (num % 2 != 0) {

                esPrimer = true;

                for (long i = 3; i <= arrelDeN; i = i + 2) {
                    if (num % i == 0) {
                        esPrimer = false;
                        break;
                    }
                }
            }
        }
    }
```

$O(\sqrt{n}/2)$



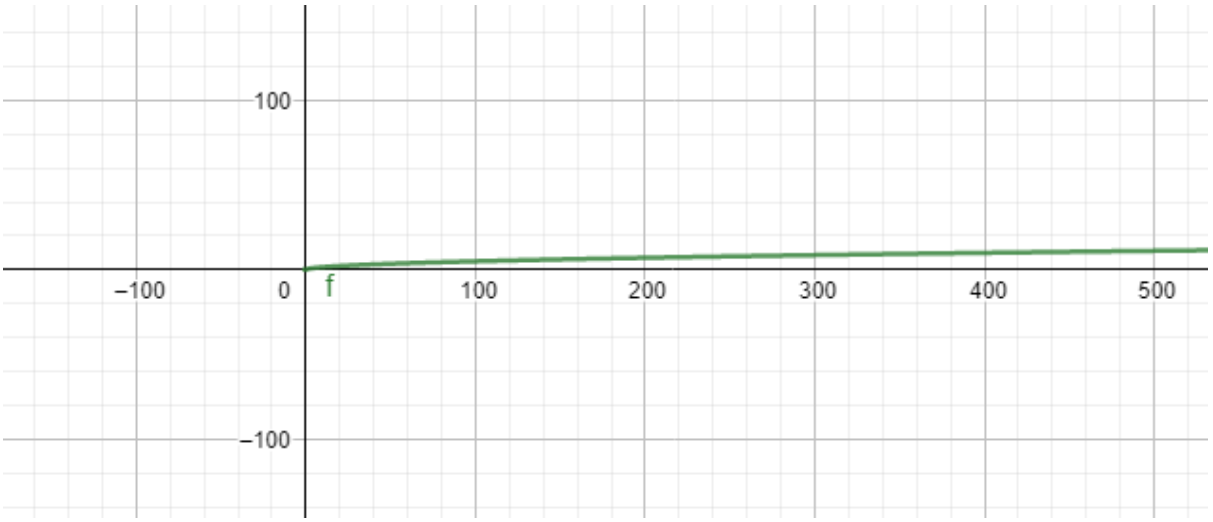
Prueba	Descripción	Resultado obtenido
1	1	0,537 ms
2	2	0,005 ms
3	3	0,011 ms
4	4	0,089 ms
5	2017	0,023 ms
6	2026	0,017 ms
7	2147483601L	0,977 ms
8	2147485783L	2,460 ms
9	9223372036854775807L	15323,436 ms
10	9223372036854775808	imposible calcular (OverFlow)

El coste de este algoritmo es la raíz/2 comparado con las versiones anteriores ya que con la raíz podemos descartar una gran cantidad de números y a parte solo tratamos los impares.

Version 5.

```java
public static boolean esNumeroPrimer_vBigInt(BigInteger num) {
    boolean esPrimer = true;

    // Calculamos la raiz cuadrarda del numero pasado y sumamos 1 para evitar aproximaciones
    BigInteger raiz = num.sqrt().add(BigInteger.ONE);
    // Comprobamos si los numeros de 3 hasta la raiz del numero, es alguno de ellos divisor del número
    for (BigInteger i = new BigInteger("3"); i.compareTo(raiz) == -1 || i.compareTo(raiz)==0; i = i.add(BigInteger.TWO)) {
        if (num.mod(i).equals(BigInteger.ZERO)) {   // num % i
            esPrimer = false;
            break; // No hace falta recorrer todo el bucle
        }
    }

    return esPrimer;
}
```

O($\sqrt{n}/2$)



Prueba	Descripción	Resultado obtenido
1	1	2,024 ms
2	2	0,027 ms
3	3	0,031 ms
4	4	0,125 ms
5	2017	0,046 ms
6	2026	0,048 ms
7	2147483601L	0,954 ms
8	2147485783L	1,768 ms
9	9223372036854775807L	15507,184 ms
10	9223372036854775808	106202,277 ms

El coste de este algoritmo es también raíz/2 ya que la base es la misma que la version 4, solo que adaptada a los BigInteger. Como los BigInteger son una clase compleja, tarda más de lo normal en hacer x función.

Otros algoritmos alternativos que hemos encontrado:

Primer algoritmo.

```
public class Main {
  public static void main(String[] args) {
    int num = 33, i = 2;
    boolean flag = false;
    while (i <= num / 2) {
      // condition for nonprime number
      if (num % i == 0) {
        flag = true;
        break;
      }      ++i;
    }
    if (!flag)
      System.out.println(num + " is a prime number.");
    else
      System.out.println(num + " is not a prime number.");
  }
}
```

En este algoritmo se usa un bucle while en vez de uno for. El while corre hasta que i <=num/2. En cada iteración se comprueba si el número es divisible entre i , que incrementa en cada vuelta.

Por otra parte nos encontramos que el programa debe pasar por todos los números uno por uno (coste lineal) hasta llegar al “num”, con lo que se está mucho tiempo en calcular y más si miramos de poner un número long o de la clase BigInteger. En definitiva es poco eficiente.

Segundo algoritmo.

```
public class Tester {
   public static void main(String args[]) {
      int i, m = 0, flag = 0;
      int n = 41;// it is the number to be checked
      m = n / 2;
      if (n == 0 || n == 1) {
         System.out.println(n + " not a prime number");
      } else {
         for (i = 2; i <= m; i++) {
            if (n % i == 0) {
               System.out.println(n + " not a prime number");
               flag = 1;
               break;
            }
         }
         if (flag == 0) {
            System.out.println(n + " is a prime number");
         }
      }
   }
}
```

En este ejemplo se elige un número de partida y a partir de un bucle for determina si es un número primo o no. Este algoritmo tiene similitudes con el anterior, debido a la poca eficacia en números grandes aunque cuenta con una sutil mejora. Al hacer ” m = n / 2” se ahorra la mitad de números a diferencia del algoritmo anterior.

Especificación pre/post numeroPrimerMesGran

{Pre ≡ 1 ≤ x ≤ 92233720368547758O7}

{Post ≡ max p : p ≤ x : ¬(#ɑ,β : 1<ɑ ∧ 1<β ; ɑ*β = p) = 0}

Función elegida para la práctica

```
public static String numeroPrimerMesGran(String num) {
      String primer = "1";
      BigInteger numero = new BigInteger(num);

  if (numero.compareTo(new BigInteger("92233720368547758O8")) == -1{
      // Si num es mas pequeno que dos, es un numero invalido
      if (Long.parseLong(num) >= 2) {
            // Si num es 2 o 3, ya es primo
            if ((Long.parseLong(num) == 2) ||
            (Long.parseLong(num) == 3)) {
                  primer = num;
            }
            else {
                  // Si num es par, lo pasamos a impar
                  if (Long.parseLong(num) % 2 == 0) {
                        num = ""+(Long.parseLong(num)-1);
                  }
                  // Bucle for para encontrar el primo mas grande
                  for (long i = Long.parseLong(num); i > 2; i = i - 2)
                  {
                        if (esNumeroPrimer_vLong(i)) {
                              primer = ""+i;
                              break;
                        }
                  }
            }
      }
      return primer;
}
```

```
        else {
              // Si el numero es par, lo pasamos a impar
          if (numero.mod((BigInteger.TWO)).equals((BigInteger.ZERO))){
              ,     numero = numero.subtract(BigInteger.ONE);
                    }

              // Bucle for para encontrar el primo mas grande
              for (BigInteger i = numero; i.compareTo(BigInteger.TWO)
              == 1; i = i.subtract(BigInteger.TWO)) {
                    if (esNumeroPrimer_vBigInt(i)) {
                          primer = i.toString();
                          break;
                    }
              }
                          return primer;
        }
}
```

.

Leer y Escribir con ficheros

En primer lugar, para implementar el método de leer un fichero de texto estuvimos consultando diferentes fuentes y formas distintas. Nos encontramos con distintas formas de hacer prácticamente lo mismo, usando métodos como *BufferedReader*, *Scanner*, *StreamTokenizer*, *DataInputStream*. Al no estar familiarizados con la mayoría decidimos al final optar por usar el método Scanner con la finalidad de leer el documento. Una vez escrito el código, el resultado sería el siguiente:

```
// Usamos el método static ya que la función está en el main
public static String[] leerFichero(int numLinies) {
      String[] result = new String[numLinies];
            try {
                  Scanner f = new Scanner(new File("libro.txt"));
                  // Bucle for para leer cada línea del documento
                  for (int i = 0; i < numLinies; i++) {
                        result[i] = f.nextLine();
                  }
                  f.close();
            }
            // Si n se encuentra el fichero deseado se muestra el
            // mensaje por consola
            catch (Exception e) {
                  System.out.println("Error occurs!!");
            }
            return result;
      }
```

Como se aprecia en el código primeramente creamos una variable String[] llamada result, de la dimensión de la variable numLinies (cantidad de números que deseamos del .txt).
Si se encuentra el fichero deseado creamos un nuevo Scanner f con el que leemos con un for y f.nextLine(), almacenamos el contenido en nuestro String[] result. La función finalmente nos devuelve la lista result.

Si hablamos de imprimir el fichero, en este caso también usamos el Scanner para plasmar la información en un fichero CSV. En el archivo de salida se muestra, en cada línea, N, P, y también el tiempo de computación en ms.

```
public static void imprimirFichero(String[] fichero, long[]
primeros, double[] tiempo) {
            try {
                  // Pedimos al usuario el nombre del fichero
                  System.out.println("Com vols que es digui
                                    el fitxer?");
                  String fit = teclat.nextLine();
                  // Creamos un fichero nuevo
                  File myObj = new File(fit+".csv");

                  PrintWriter f = new PrintWriter(myObj);

                  // Bucle for para escribir sobre el fichero
                  for (int i = 0; i < primeros.length; i++) {
                        f.printf("%d.;%d; %d; %.3f ms\n", i + 1,
                                    Long.parseLong(fichero[i]),
                                    primeros[i], tiempo[i] * 1.0e-6);
                  }
                  // Cerramos la impresion
                  f.close();

            }

            catch (IOException e) {
                  System.out.println("An error occurred.");
                  e.printStackTrace();
            }
      }
```

Como podemos ver, en primer lugar preguntamos al usuario el nombre del documento. Si ya existe dicho documento, se sobreescribirá automáticamente sino, se creará uno nuevo. Seguidamente leemos e imprimimos nuestros datos de las tablas fichero[], primeros[] y tiempo[] con un for y f.printf(“ “).

Conclusiones::

En este trabajo hemos mejorado en diferentes aspectos como por ejemplo: detectar el coste de un algoritmo, hemos mejorado con la POO y también hemos aprendido a usar la clase BigInteger, que ninguno de nosotros conocía anteriormente. También hemos repasado cómo crear el contenido de pre post y como leer e imprimir un fichero.

En cuanto a problemas que nos han surgido, en primer lugar a la hora de usar la classe BigInteger no sabíamos cómo comparar dos valores ya que al ser una classe a parte no lo podíamos poner tan fácil como (x > y), y que nos marcaba error. Finalmente hemos usado el método compareTo() y lo hemos podido solucionar.