

# 1 | Complejidad

## META

Que el alumno visualice el concepto de “función de complejidad computacional”.

## OBJETIVOS

Al finalizar la práctica el alumno será capaz de:

- Medir la complejidad en número de operaciones de un método de manera experimental.
- Comparar el desempeño entre las versiones iterativas y recursivas de un método.

## ANTECEDENTES

### Sucesión de Fibonacci.

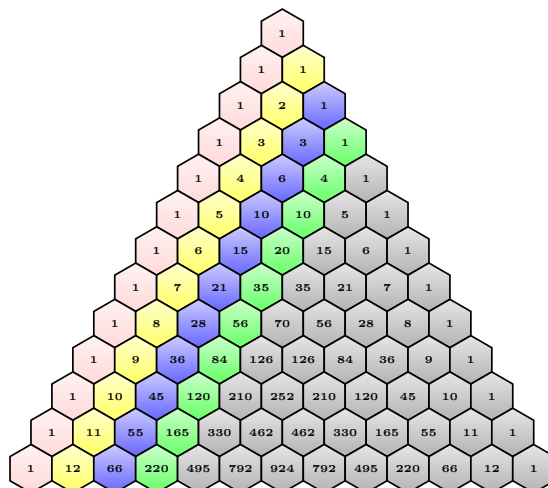
La sucesión de fibonacci fue descubierta por Fibonacci en relación a un problema de conejos. Supongamos que se tiene una pareja de conejos y cada mes esa pareja cría una nueva pareja. Después de dos meses, la nueva pareja se comporta de la misma manera. Entonces, el número de parejas nuevas nacidas  $a_n$  en el  $n$ -ésimo mes es  $a_{n-1} + a_{n-2}$ , ya que nace una pareja por cada pareja nacida en el mes anterior y cada pareja nacida hace dos meses cria una nueva pareja. Por convención consideremos  $a_0 = 0$  y  $a_1 = 1$ .

---

#### Actividad 1.1

Define, con estos datos, la función de fibonacci.

---



**Figura 1.1** Triángulo de Pascal. Autor:M.H. Ahmadi

## Triángulo de Pascal.

En la figura Figura 1.1 se muestran algunos términos del Triángulo de Pascal.

Matemáticamente, podemos definir el elemento  $\text{Pascal}_{ij}$  que corresponde al elemento en la fila  $i$ , columna  $j$  de la siguiente manera:

$$\text{Pascal}_{ij} = \begin{cases} 1 & \text{si } j = 0 \text{ ó } j = i \\ \text{Pascal}_{(i-1)(j-1)} + \text{Pascal}_{(i-1)(j)} & \text{En cualquier otro caso.} \end{cases} \quad (1.1)$$

## DESARROLLO

La práctica consiste en implementar métodos que calculen el triángulo de pascal y el  $n$ -ésimo número de fibonacci, al tiempo que estiman el número de operaciones realizadas. Esto se realizará en forma recursiva e iterativa. Se deberá implementar la interfaz `IComplejidad` en una clase llamada `Complejidad`. Se entregan pruebas unitarias para ayudar a verificar que estas funciones estén bien implementadas. Adicionalmente deberán llevar la cuenta del número de operaciones estimadas en un atributo de la clase para generar un reporte ilustrado sobre el número de operaciones que realiza cada método.

## GNU PLOT

Gnuplot es una herramienta interactiva que permite generar gráficas a partir de archivos de datos planos. Para esta práctica, los datos deben ser guardados en un archivo de este tipo y graficados con gnuplot. Supongamos que el archivo donde se guardan es llamado **datos.dat**.

Por ejemplo, para el método de Fibonacci el archivo **datos.dat** tendría algo semejante al siguiente contenido:

**Listing 1.1:** data/Fibonaccilt.dat

```
0      1
1      1
2      2
3      3
4      4
5      5
```

donde la primer columna es el valor del argumento y la segunda, el número de operaciones.

Para el método de Pascal el archivo **datos.dat** tendría algo semejante al siguiente contenido:

**Listing 1.2:** data/PascalRec.dat

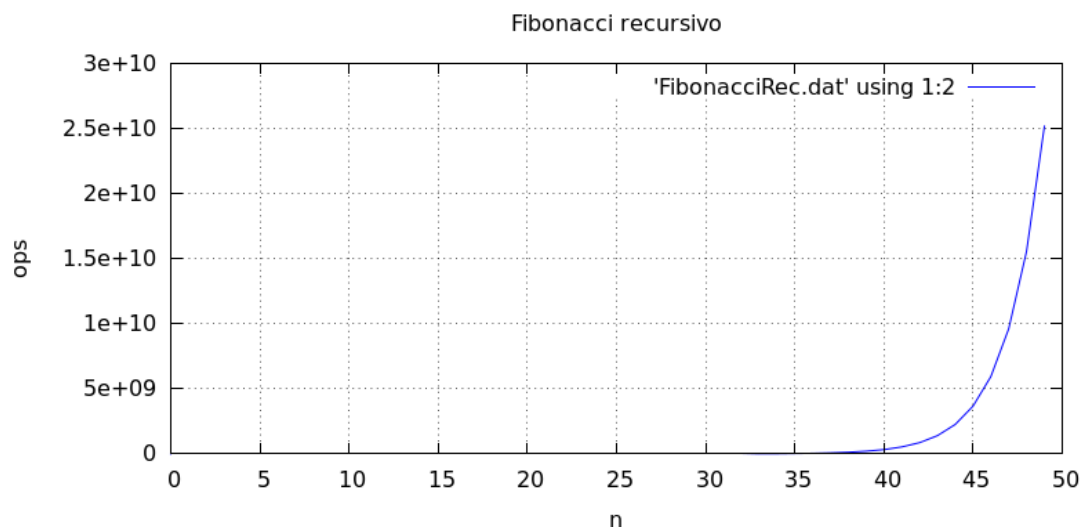
```
0      0      2
1      0      2
1      1      2

2      0      2
2      1      4
2      2      2

3      0      2
3      1      6
3      2      6
3      3      2
```

donde la primer columna es el valor del renglón, la segunda es la columna, y la tercera el número de operaciones. Observa que cada vez que cambies de renglón, debes dejar una linea en blanco para indicarle a `gnuplot` cuándo cambia el valor en el eje x.

## 1. Complejidad



**Figura 1.2** Complejidad en tiempo al calcular el n-ésimo coeficiente de la serie de Fibonacci en forma recursiva.

## Gráficas en 2D

Al iniciar el programa `gnuplot` aparecerá un prompt y se puede iniciar la sesión de trabajo. A continuación se muestra cómo crear una gráfica 2D. Deberás obtener algo como la Figura 1.2.

```

1  gnuplot> set title "Mi_gráfica"      //Título para la gráfica
2  gnuplot> set xlabel "Eje_X:n"       //Título para el eje X
3  gnuplot> set ylabel "Eje_Y:ops"     //Título para el eje Y
4  gnuplot> set grid "front";          // Decoración
5  gnuplot> plot "datos.dat" using 1:2 with lines lc rgb 'blue' //
    ↪ graficamos los datos
6  gnuplot> set terminal pngcairo size 800,400 //algunas caracterí
    ↪ sticas de la imagen que se guardará
7  gnuplot> set output 'fib.png'       //nombre de la imagen que se
    ↪ guardará
8  gnuplot> replot                    //lo graficamos para que se
    ↪ guarde en la imagen

```

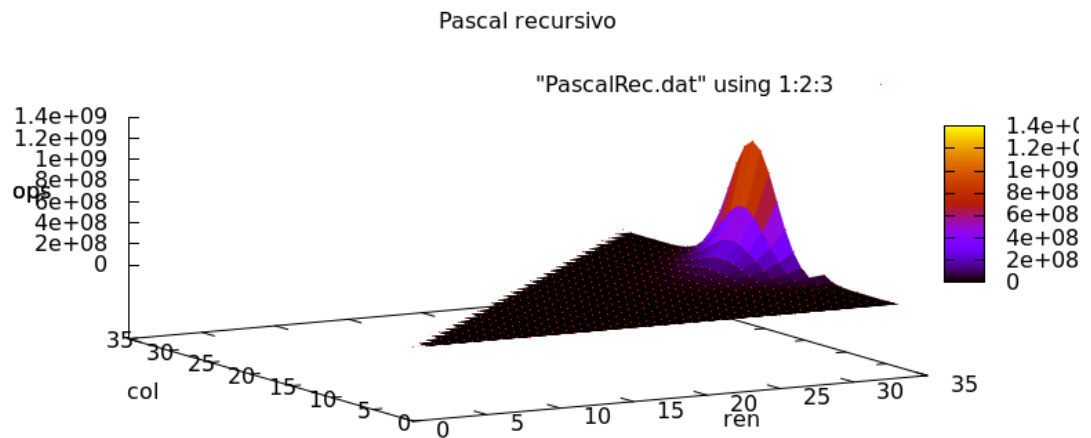
## Gráficas en 3D

A continuación se muestra cómo crear una gráfica 3D. Observa que, en este caso, el archivo de datos requiere tres columnas. Deberás obtener algo como la Figura 1.3.

```

1  gnuplot> set title "Mi_gráfica"
2  gnuplot> set xlabel "Eje_X"

```



**Figura 1.3** Complejidad en tiempo al calcular el coeficiente del triángulo de Pascal para el renglón y columna dados.

```

3  gnuplot> set ylabel "Eje_Y"
4  gnuplot> set zlabel "Eje_Z"
5  gnuplot> set pm3d
6  gnuplot> splot "datos.dat" using 1:2:3 with dots
7  gnuplot> set terminal pngcairo size 800,400
8  gnuplot> set output 'pascal.png'
9  gnuplot> replot

```

## EJERCICIOS

1. Crea la clase `Complejidad`, que implemente `IComplejidad`. Agrega las firmas de los métodos requeridos y asegúrate de que compile, aunque aún no realice los cálculos.
2. Programa los métodos indicados en la interfaz. Las pruebas unitarias te ayudarán a verificar tus implementaciones de Fibonacci y Pascal. Compila tu código utilizando el comando `ant` en el directorio donde se encuentra el archivo `build.xml`, si compila correctamente las pruebas se ejecutarán automáticamente.

En particular, nota que los métodos estáticos se implementan en la interfaz, sólo son auxiliares para escribir los datos en el archivo. El archivo se debe abrir para agregar (modo *append*), de modo que los datos se acumulen entre llamadas sucesivas al método, revisa la documentación de `PrintStream` y `FileOutputStream`, te ayudarán mucho en esta parte.

3. Abre el archivo `ComplejidadTest.java`. Lee el código. Observa que cada mé-

## 1. Complejidad

todo marcado con la anotación `@Test` se ejecuta como una prueba unitaria. La expresión `assertEquals` se utiliza para verificar que el código devuelva el valor esperado. Por lo demás, el archivo contiene la definición de una clase común y corriente. Agrega cuatro métodos que prueben el funcionamiento de los cuatro métodos que programaste para calcular Pascal y Fibonacci. Elige un número y calcula a mano la respuesta correcta, tus pruebas deberán mandar ejecutar el código y comparar su resultado con la respuesta que calculaste.

Para saber más sobre la programación de pruebas unitarias revisa la documentación oficial del paquete `org.junit`.

4. Agrégale un atributo a la clase para que cuente lo siguiente:

**Iterativos** El número de veces que se ejecuta el ciclo más anidado. Observa que puedes inicializar el valor del atributo auxiliar al inicio del método y después incrementarlo en el interior del ciclo más anidado.

**Recursivos** El número de veces que se manda llamar la función. Aquí utilizarás una técnica un poco más avanzada que sirve para optimizar varias cosas. Necesitarás crear una función auxiliar (*privada*) que reciba los mismos parámetros. En la función original revisarás que se cumplan las precondiciones de los datos e inicializarás la variable que cuenta el número de llamadas recursivas. La función auxiliar es la que realmente realizará la recursión. Ya no revises aquí las precondiciones, pues ya sólo depende de ti garantizar que no la vas a llamar con parámetros inválidos. Incrementa aquí el valor del atributo contador, deberá incrementarse una vez por cada vez en que mandes llamar esta función.

A continuación se ilustra la idea utilizando la función factorial: (OJO: tu código no es igual, sólo se ilustra el principio).

```

1  /** Ejemplo de cómo contar el número de llamadas a la
2   * implementación recursiva de la función factorial. */
3  public class ComplejidadFactorial {
4
5      /* Número de operaciones realizadas en la última
6       * llamada a la función. */
7      private long contador;
8
9      /** Valor del contador de operaciones después de la ú
10       ↪ ltima
11       * llamada a un método. */
12      public long leeContador() {
13          return contador;
14      }
15
16      /** n! */
17      public int factorial(int n) {
18          contador = 1;
19          if (n < 0) throw new IndexOutOfBoundsException();
20          if (n == 0) return 1;

```

```

20     return factorialAux(n);
21 }
22
23 private int factorialAux(int n) {
24     operaciones++;
25     if (n == 1) return 1;
26     else return factorialAux(n - 1);
27 }
28
29 /** Imprime en pantalla el número de llamadas a la funci
    ↪ ón para
30     * varios parámetros. */
31 public static void main(String[] args) {
32     ComplejidadFactorial c = new ComplejidadFactorial();
33     for(int n = 0; n < 50; n++) {
34         int f = c.factorial(n);
35         System.out.format("Para n=%d se realizaron %d
    ↪ operaciones",
36                             n, c.leeContador());
37     }
38 }
39 }

```

5. Crea un método `main` en una clase `UsComplejidad` que mande llamar los métodos programados para diferentes valores de sus parámetros y que guarde los resultados en archivos de texto. Podrás ejecutarlo con el comando `ant run`.
6. Para el método de fibonacci, genera las gráficas  $n$ (entrada) vs número de operaciones y haz un análisis de lo que sucede. ¿Cuál es el orden de complejidad? Justifica.
7. Para el método recursivo del cálculo del triángulo de Pascal, genera una gráfica en 3-D en donde el parámetro renglón se encontrará en el eje X, el parámetro columna se encontrará en el eje Y, el número de operaciones en el eje Z y haz un análisis de lo que sucede. ¿Cuál es el orden de complejidad? Justifica.
8. Entrega tus resultados en un reporte en un archivo `.pdf`, junto con tu código limpio y empaquetado.

## PREGUNTAS

1. ¿Cuál es el máximo valor de  $n$  que pudiste calcular para el factorial sin que se alentara tu computadora? (Puede variar un poco de computadora a computadora ( $\pm 3$ ), así que no te esfuerces en encontrar un valor específico).
2. ¿Cuál es el máximo valor de  $ren$  que pudiste calcular para el triángulo de Pascal sin que se alentara tu computadora?

## 1. Complejidad

---

3. Justifica a partir del código ¿cuál es el orden de complejidad para cada uno de los métodos que programaste?
4. Escribe un reporte con tus gráficas generadas y las respuestas a las preguntas anteriores.