

# Elementos de complejidad algorítmica

## Relaciones de recurrencia

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

11 de marzo de 2021



# Definición

- 1 Definición
- 2 Recursión doble
- 3 Bibliografía

# Definición matemática

## Definición (Relación de recurrencia)

Una *relación de recurrencia* para una **sucesión**  $\{a_n\}_{n=0}^{\infty} = \{a_0, a_1, a_2, a_3, \dots\}$  es una fórmula que expresa cada término  $a_n$ , a partir de cierto  $n_0 \in \mathbb{N}$ , en función de uno o más de los términos que le preceden.

- Los valores de los términos necesarios para empezar a calcular se llaman condiciones iniciales.
- Una **solución** de la relación de recurrencia es una sucesión cuyos términos (a partir de  $a_2$ ) verifiquen la relación.
- **Resolver** una recurrencia (que liga los términos de una sucesión  $\{a_n\}$ ) significa hallar una fórmula explícita *algebraica* en la que al sustituir  $n$  obtengamos al término  $a_n$ . Fernández Gallardo y Fernández Pérez 2018

# Ejemplos básicos

## Ejemplos:

- *Progresión aritmética*:  $a_n = a_{n-1} + d$ .

Ejemplar:

$$a_0 = 3$$

$$d = 7$$

$$\{a_n\} = \{3, 10, 17, 24, 31, 38, 45, 52, 59, 66, 73\}$$

Solución:  $a_n = a_0 + dn$ .

- *Progresión geométrica*:  $a_n = ra_{n-1}$

Ejemplar:

$$a_0 = 3$$

$$r = 5$$

$$\{a_n\} = \{3, 15, 75, 375, 1875, 9375, 46875, 234375\}$$

Solucion:  $a_n = a_0 r^n$

- *Sucesión de Fibonacci:*

$$a_0 = a_1 = 1$$

$$a_n = a_{n-1} + a_{n-2} \quad \forall n \geq 2$$

$$\{a_n\} = \{1, 1, 2, 3, 5, 8, 13, 21, 34\}$$

Solución:

$$a_n = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^n$$

- *Lineales homogéneas* de orden  $m$

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_m a_{n-m}$$

# Relaciones de recurrencia y complejidad

- Las relaciones de recurrencia se utilizan en forma natural para estimar la complejidad en tiempo de algoritmos recursivos.
- Para el cálculo, cada llamada recursiva se sustituye por el número de pasos que toma evaluarla.

```
1 public static long factorial(long n) {  
2     if (n < 0) throw new IllegalArgumentException(); // O(1)  
3     if (n == 0 || n == 1) return 1; // O(1)  
4     else return n * factorial(n-1); // c + Tf(n-1)  
5 }
```

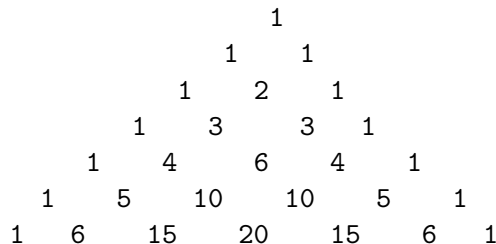
$$\begin{aligned}T_{\text{factorial}}(n) &= \Theta(1) + T_f(n-1) \\ &= \mathcal{O}(n)\end{aligned}$$

¡Es la progresión aritmética!

# Recursión doble

- 1 Definición
- 2 Recursión doble
- 3 Bibliografía

# Ejemplo: Triángulo de Pascal




---

```

1  public static long pascal(int n, int m) {
2      if (n < 0 || m < 0 || m > n)
3          throw new IllegalArgumentException();
4      if (n == 0 || n == m) return 1;
5      else return pascal(n-1, m-1) + pascal(n-1, m);
6  }
```

---

$$T_p(n) = \Theta(1) + 2T_p(n-1) \quad (1)$$



# Orden de complejidad para el Triángulo de Pascal

$$T_p(n) = \Theta(1) + 2T_p(n-1) \quad (2)$$

$$= \Theta(1) + 2(\Theta(1) + 2T_p(n-2)) \quad (3)$$

$$= (1+2)\Theta(1) + 2^2T_p(n-2) \quad (4)$$

$$= (1+2+2^2)\Theta(1) + 2^3T_p(n-3) \quad (5)$$

$$\dots \quad (6)$$



$$= \left( \frac{2^{n-1} - 1}{2 - 1} \right) \Theta(1) + 2^n \Theta(1) \quad (7)$$

$$= \mathcal{O}(2^n) \quad (8)$$

# Bibliografía

- 1 Definición
- 2 Recursión doble
- 3 Bibliografía**

# Bibliografía I

-  Fernández Gallardo, Pablo y José Luis Fernández Pérez (2018). «El discreto encanto de la matemática». En: cap. Recurrencias. URL:  
<http://verso.mat.uam.es/~pablo.fernandez/cap8-dic18.pdf>.
-  Preiss, Bruno (1999). *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. Wiley.

# Licencia

Creative Commons  
Atribución-No Comercial-Compartir Igual

