

---

# ELEMENTOS DE CÁLCULO NUMÉRICO

Primer Cuatrimestre 2026

---

## Práctica N° 1: Fundamentos de la Computación Numérica

### Máquinas de Turing y computabilidad

**Ejercicio 1** Diseñe una máquina de Turing que multiplique por 2 un número natural representado en unario (como cadena  $1^n$ ).

**Ejercicio 2 (Codificación y cardinalidad)** Utilizando el concepto de **codificación** de una máquina de Turing como una cadena  $\langle M \rangle$  (y el de máquina de Turing Universal), demuestre que el conjunto de todas las máquinas de Turing es **enumerable** (existe una biyección con  $\mathbb{N}$ ).

**Ejercicio 3 (Números computables y cardinalidad)** Un número real  $r$  es **computable** si existe una máquina de Turing que, dado  $n \in \mathbb{N}$ , produce los primeros  $n$  dígitos de la expansión decimal de  $r$ . Asumiendo la tesis de Church-Turing, esto es equivalente a decir que existe un algoritmo para calcular  $r$  con precisión arbitraria:

(a) Demuestre que  $\pi$  y  $e$  son computables.

(b) Demuestre que el conjunto de números computables es enumerable.

**Ejercicio 4 (Problema del halting)** El problema del halting pregunta: ¿existe un algoritmo que determine si una máquina de Turing  $M$  se detiene con entrada  $w$ ?

Demuestre que **no existe tal algoritmo** usando un **argumento diagonal**:

(a) Suponga que existe una máquina  $H$  que resuelve el problema:  $H(\langle M \rangle, w)$  se detiene y responde "sí" si  $M$  se detiene con entrada  $w$ , y responde "no" en caso contrario.

(b) Construya una máquina  $D$  que recibe como entrada la codificación  $\langle M \rangle$  de una máquina y hace lo siguiente:

- Ejecuta  $H(\langle M \rangle, \langle M \rangle)$  para determinar si  $M$  se detiene con entrada  $\langle M \rangle$ .
- Si  $H$  responde "sí", entonces  $D$  entra en un bucle infinito.
- Si  $H$  responde "no", entonces  $D$  se detiene.

(c) Obtenga una contradicción analizando qué sucede cuando ejecutamos  $D$  con entrada  $\langle D \rangle$ : ¿se detiene  $D(\langle D \rangle)$  o no?

# Complejidad computacional

**Ejercicio 5 (Notación asintótica: definiciones)** Utilizando las definiciones formales de  $O$ ,  $\Omega$  y  $\Theta$ , determine si las siguientes afirmaciones son verdaderas o falsas.

(a) Demuestre o refute:  $3n^2 + 5n + 7 = \Theta(n^2)$ .

(b) Demuestre o refute:  $n^2 = O(n^2 \log n)$ .

(c) Demuestre o refute:  $2^n = \Omega(3^n)$ .

**Ejercicio 6 (Propiedades de la notación  $O$ )**

(a) Si  $f(n) = O(g(n))$  y  $g(n) = O(h(n))$ , ¿es cierto que  $f(n) = O(h(n))$ ?

(b) Si  $f_1(n) = O(g_1(n))$  y  $f_2(n) = O(g_2(n))$ , demuestre que:  $f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

(c) Si  $f_1(n) = O(g_1(n))$  y  $f_2(n) = O(g_2(n))$ , demuestre que:  $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

(d) Simplifique:  $O(n^2) + O(n^3)$

(e) Simplifique:  $O(n) \cdot O(\log n)$

**Ejercicio 7 (Análisis de complejidad: bucles simples)** Para cada fragmento de código, determine su complejidad temporal en el peor caso:

(a) 

```
for i = 1 to n:
    print(i)
```

(b) 

```
for i = 1 to n:
    for j = 1 to n:
        print(i, j)
```

(c) 

```
for i = 1 to n:
    for j = 1 to i:
        print(i, j)
```

(d) 

```
for i = 1 to n:
    for j = 1 to n:
        for k = 1 to n:
            print(i, j, k)
```

(e) 

```
i = n
while i >= 1:
    print(i)
    i = i / 2
```

**Ejercicio 8 (Análisis de complejidad: bucles anidados complejos)** Calcule la complejidad temporal:

```

1. for i = 1 to n:
   for j = i to n:
     for k = j to n:
       print(i, j, k)

2. for i = 1 to n:
   j = 1
   while j < n:
     print(i, j)
     j = j * 2

3. for i = n to 1 (decrementando):
   for j = 1 to i*i:
     print(i, j)

4. for i = 1 to log(n):
   for j = 1 to 2^i:
     print(i, j)

```

**Ejercicio 9 (Teorema maestro)** *El teorema maestro resuelve recurrencias de la forma:*

$$T(n) = aT(n/b) + f(n).$$

(a) *Enuncie los tres casos del teorema maestro.*

(b) *Aplique el teorema maestro para resolver:*

- $T(n) = 2T(n/2) + O(n)$
- $T(n) = 2T(n/2) + O(1)$
- $T(n) = 3T(n/4) + O(n \log n)$
- $T(n) = T(n/2) + O(n)$

(c) *Para  $T(n) = 2T(n/2) + O(n \log n)$ , explique por qué no se puede aplicar directamente el teorema maestro y resuelva usando el método de sustitución.*

**Ejercicio 10 (Análisis de algoritmos recursivos)** *Para cada algoritmo recursivo, calcule su complejidad usando relaciones de recurrencia:*

(a) *Factorial:*

```

def factorial(n):
    if n <= 1: return 1
    return n * factorial(n-1)

```

*Plantee  $T(n) = T(n - 1) + O(1)$  y resuelva.*

(b) *Fibonacci ingenuo:*

```

def fib(n):
    if n <= 1: return n
    return fib(n-1) + fib(n-2)

```

Plantee  $T(n) = T(n - 1) + T(n - 2) + O(1)$  y demuestre que  $T(n) = \Omega(2^{n/2})$ .

(c) *Búsqueda binaria:*

```

def binary_search(arr, x, low, high):
    if low > high: return -1
    mid = (low + high) / 2
    if arr[mid] == x: return mid
    if arr[mid] > x: return binary_search(arr, x, low, mid-1)
    return binary_search(arr, x, mid+1, high)

```

Plantee  $T(n) = T(n/2) + O(1)$  y resuelva.

(d) *Mergesort:*

```

def mergesort(arr):
    if len(arr) <= 1: return arr
    mid = len(arr) / 2
    left = mergesort(arr[0:mid])
    right = mergesort(arr[mid:len(arr)])
    return merge(left, right) # O(n) para merge

```

Plantee  $T(n) = 2T(n/2) + O(n)$  y resuelva usando el teorema maestro.

**Ejercicio 11 (Algoritmo de Karatsuba)** Las bibliotecas de enteros de precisión arbitraria (como Python's `int`, Java's `BigInteger`, o GMP) deben multiplicar eficientemente números de miles de dígitos. La multiplicación estándar requiere  $O(n^2)$  operaciones para números de  $n$  dígitos. El **algoritmo de Karatsuba** (1960) usa divide-y-vencerás: divide cada número  $x, y$  de  $n$  dígitos en mitades de  $n/2$  dígitos:

$$x = x_1 \cdot B^{n/2} + x_0, \quad y = y_1 \cdot B^{n/2} + y_0$$

donde  $B$  es la base (típicamente  $B = 2^{32}$  o  $2^{64}$ ). En lugar de las 4 multiplicaciones recursivas del método ingenuo, calcula solo 3:

$$p_1 = x_1 y_1, \quad p_2 = x_0 y_0, \quad p_3 = (x_1 + x_0)(y_1 + y_0)$$

y obtiene el producto como:

$$xy = p_1 \cdot B^n + (p_3 - p_1 - p_2) \cdot B^{n/2} + p_2.$$

Demuestre que la complejidad del algoritmo de Karatsuba es  $T(n) = \Theta(n^{\log_2 3})$ .

Sugerencia: Plantee la recurrencia  $T(n) = 3T(n/2) + O(n)$  (justificando cada término) y aplique el teorema maestro.

# Aritmética de punto flotante

En los siguientes ejercicios trabajamos con un sistema de punto flotante **decimal** con  $t = 3$  dígitos de mantisa y redondeo al más cercano. Los números se representan como  $\pm d_1.d_2d_3 \times 10^e$  con  $d_1 \neq 0$ . Denotamos  $\text{fl}(x)$  al representante de punto flotante de  $x$ .

**Propiedad fundamental:** Si  $x$  está en el rango representable, entonces

$$|\text{fl}(x) - x| \leq \frac{1}{2} \times 10^{e-t+1}$$

donde  $e$  es el exponente de  $x$ . Esto implica que el **error relativo** satisface:

$$\left| \frac{x - \text{fl}(x)}{x} \right| \leq \varepsilon_{\text{mach}} = \frac{1}{2} \times 10^{1-t}.$$

**Ejercicio 12 (Operaciones básicas)** Con  $t = 3$  dígitos:

1. Calcule  $\text{fl}(1.23 + 4.56)$  y su error relativo.
2. Calcule  $\text{fl}(1.23 \times 4.56)$  y su error relativo.
3. Calcule  $\text{fl}(9.87/3.21)$  y su error relativo.

**Ejercicio 13 (Propagación del error)** Sea  $\text{fl}(x) = x(1 + \delta_x)$  y  $\text{fl}(y) = y(1 + \delta_y)$  con  $|\delta_x|, |\delta_y| \leq \varepsilon_{\text{mach}}$ .

1. Demuestre que  $\text{fl}(\text{fl}(x) + \text{fl}(y)) \approx (x + y)(1 + \delta)$  con  $|\delta| \leq \varepsilon_{\text{mach}}$  si no hay cancelación.
2. Demuestre que  $\text{fl}(\text{fl}(x) \times \text{fl}(y)) \approx xy(1 + \delta)$  con  $|\delta| \leq 2\varepsilon_{\text{mach}}$ .
3. ¿Por qué el error relativo puede ser arbitrariamente grande en la suma si  $x + y \approx 0$ ?

**Ejercicio 14 (Cancelación catastrófica)** Con  $t = 3$  dígitos:

1. Calcule  $\text{fl}(1.23 - 1.22)$  y observe la pérdida de dígitos significativos.
2. Suponga  $x = 1.234$  e  $y = 1.224$  (no representables exactamente). Calcule  $\text{fl}(x)$ ,  $\text{fl}(y)$  y luego  $\text{fl}(x) - \text{fl}(y)$ . Compare con  $x - y = 0.01$  exacto.
3. Explique por qué la cancelación amplifica los errores de redondeo previos.

**Ejercicio 15 (Raíces de la ecuación cuadrática)** Considere  $x^2 - 200x + 1 = 0$  con  $t = 3$  dígitos. Las raíces son  $x = 100 \pm \sqrt{9999}$ .

1. Calcule ambas raíces usando  $x = \frac{200 \pm \text{fl}(\sqrt{9999})}{2}$ . Use  $\text{fl}(\sqrt{9999}) = 100$ . Observe la cancelación en  $x_2 = \frac{200 - 100}{2}$ .
2. Calcule  $x_2$  usando la fórmula alternativa  $x_2 = \frac{1}{x_1}$  (aprovechando que  $x_1 x_2 = 1$ ). Compare.
3. Alternativamente, use  $x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$  para calcular la raíz pequeña. Compare.

**Ejercicio 16 (Reformulación algebraica I: conjugado)** Con  $t = 3$  dígitos, calcule  $f(x) = \sqrt{x+1} - \sqrt{x}$  para  $x = 100$ .

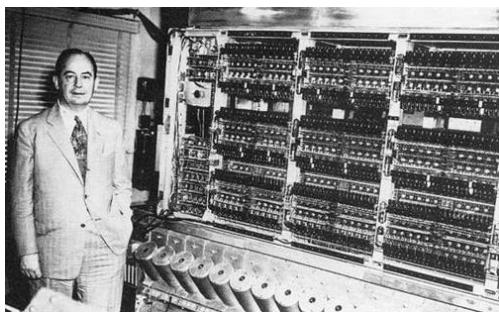
1. Método directo: calcule  $fl(\sqrt{101}) - fl(\sqrt{100})$ . Use  $fl(\sqrt{101}) = 10.0$ .
2. Reformule usando el conjugado:  $\sqrt{x+1} - \sqrt{x} = \frac{1}{\sqrt{x+1} + \sqrt{x}}$ . Calcule con esta fórmula.
3. Compare los errores relativos (valor exacto:  $\approx 0.04988$ ).

**Ejercicio 17 (Reformulación algebraica II: Taylor)** Con  $t = 3$  dígitos, calcule  $f(x) = \frac{1-\cos x}{x^2}$  para  $x = 0.1$ .

1. Método directo: use  $fl(\cos(0.1)) = 0.995$  y calcule  $fl\left(\frac{1-0.995}{0.01}\right)$ .
2. Reformule usando Taylor:  $\frac{1-\cos x}{x^2} = \frac{1}{2} - \frac{x^2}{24} + O(x^4)$ . Calcule para  $x = 0.1$ .
3. Compare con el valor exacto  $\approx 0.4996$ .

**Ejercicio 18 (Orden de operaciones)** Con  $t = 3$  dígitos, considere sumar 0.001 al número 1.00 repetidamente 1000 veces.

1. Calcule  $fl(1.00 + 0.001)$ . ¿Qué observa?
2. ¿Qué resultado se obtiene después de 1000 iteraciones?
3. Proponga una forma más precisa de calcular  $1.00 + 1000 \times 0.001$ .



John von Neumann  
Budapest 1903 - Washington 1957



Alan Turing  
Londres 1912 - Cheshire 1954