# Técnicas Fundamentales

## Teorema maestro y Dividir para conquistar

Segundo semestre 2022

IIC2283

Prof. Nicolás Van Sint Jan

# Recordatorio: el Teorema Maestro

Muchas de las **ecuaciones de recurrencia** que vamos a usar en este curso tienen la siguiente forma:

$$T(n) = \begin{cases} c & n = 0 \\ a \cdot T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + f(n) & n \geq 1 \end{cases}$$

donde $a$, $b$ y $c$ son constantes, y $f(n)$ es una función arbitraria.

El **Teorema Maestro** nos dirá cuál es el **orden** de $T(n)$ dependiendo de ciertas condiciones sobre $a$, $b$ y $f(n)$.

¿ Qué pasa si cambiamos $\left\lfloor \frac{n}{b} \right\rfloor$ por $\left\lceil \frac{n}{b} \right\rceil$ ?

**R:** El Teorema Maestro también se puede utilizar cuando $\left\lfloor \frac{n}{b} \right\rfloor$ es reemplazado por $\left\lceil \frac{n}{b} \right\rceil$.

# Recordatorio: Una condición de regularidad sobre funciones

Antes de dar el enunciado del Teorema Maestro necesitamos definir una **condición de regularidad** sobre la función $f(n)$.

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$ una función y $a, b \in \mathbb{R}$ constantes tales que $a \geq 1$ y $b > 1$.

## Definición

La función $f$ es $(a, b)$-**regular** si existen constantes $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $c < 1$ y

$$\forall n \geq n_0.\ a \cdot f\left(\left\lfloor \frac{n}{b} \right\rfloor\right) \leq c \cdot f(n)$$

## Ejercicio

1. Demuestre que las funciones $n$, $n^2$ y $2^n$ son $(a, b)$-regulares si $a < b$.
2. Demuestre que la función $\log_2(n)$ no es (1,2)-regular.

# Outline

Teorema maestro

Dividir para conquistar

# Outline

Teorema maestro

Dividir para conquistar

# El enunciado del Teorema Maestro

## Teorema Maestro

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$ una función, $a, b, c \in \mathbb{R}_0^+$ constantes tales que $a \geq 1$ y $b > 1$, y $T(n)$ una función definida por la siguiente ecuación de recurrencia:

$$T(n) \quad = \quad \begin{cases} c & n = 0 \\ a \cdot T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + f(n) & n \geq 1 \end{cases}$$

Se tiene que:

1. Si $f(n) \in \mathcal{O}\left(n^{\log_b(a)-\varepsilon}\right)$ para $\varepsilon > 0$, entonces $T(n) \in \Theta\left(n^{\log_b(a)}\right)$
2. Si $f(n) \in \Theta\left(n^{\log_b(a)}\right)$, entonces $T(n) \in \Theta\left(n^{\log_b(a)} \cdot \log_2(n)\right)$
3. Si $f(n) \in \Omega\left(n^{\log_b(a)+\varepsilon}\right)$ para $\varepsilon > 0$ y $f$ es $(a, b)$-regular, entonces $T(n) \in \Theta\left(f(n)\right)$

# Usando el Teorema Maestro

## Ejemplo

Considere la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} 1 & n = 0 \\ 3 \cdot T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c \cdot n & n \geq 1 \end{cases}$$

Dado que $\log_2(3) > 1.5$, tenemos que $\log_2(3) - 0.5 > 1$

Deducimos que $c \cdot n \in \mathcal{O}\left(n^{\log_2(3)-0.5}\right)$, por lo que usando el Teorema Maestro concluimos que $T(n) \in \Theta\left(n^{\log_2(3)}\right)$

# El Teorema Maestro y la función $\lceil x \rceil$

Suponga que cambiamos $\lfloor \frac{n}{b} \rfloor$ por $\lceil \frac{n}{b} \rceil$ en la definición de $(a, b)$-regularidad.

El Teorema Maestro sigue siendo válido pero con $T\left(\lfloor \frac{n}{b} \rfloor\right) + f(n)$ reemplazado por $T\left(\lceil \frac{n}{b} \rceil\right) + f(n)$.

Ahora considere la siguiente ecuación:

$$T(n) = \begin{cases} 1 & n = 0 \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + c \cdot n & n \geq 1 \end{cases}$$

¿Se puede utilizar el Teorema Maestro en la ecuación anterior?

# Analizando la complejidad de un algoritmo

Sea $\mathcal{A} : \Sigma^* \to \Sigma^*$ un algoritmo.

## Definición

Decimos que $\mathcal{A}$ **en el peor caso es** $\mathcal{O}(f(n))$ si

$$t_{\mathcal{A}}(n) \in \mathcal{O}(f(n))$$

Recuerde que $t_{\mathcal{A}}(n)$ es el mayor número de pasos realizados por $\mathcal{A}$ sobre las entradas $w \in \Sigma^*$ de largo $n$

# Analizando la complejidad de un algoritmo

## Notación

Las definición de peor caso puede ser modificada para considerar las notaciones $\Theta$ y $\Omega$

- Simplemente reemplazando $\mathcal{O}(f(n))$ por $\Theta(f(n))$ u $\Omega(f(n))$, respectivamente

Por ejemplo, decimos que $\mathcal{A}$ **en peor caso es** $\Omega(f(n))$ si

$$t_{\mathcal{A}}(n) \in \Omega(f(n)).$$

# Outline

Teorema maestro

Dividir para conquistar

# Dividir para conquistar

Esta es la forma genérica de una algoritmo que utiliza la técnica de **dividir para conquistar**:

## Algoritmo

**DividirParaConquistar**($w$)
    **if** $|w| \leq k$ **then return InstanciasPequeñas**($w$)
    **else**
        Dividir $w$ en $w_1, \ldots, w_\ell$
        **for** $i := 1$ **to** $\ell$ **do**
            $S_i :=$ **DividirParaConquistar**($w_i$)
        **return Combinar**($S_1, \ldots, S_\ell$)

¿Cuál es la complejidad de un algoritmo de **dividir para conquistar**?

# Dividir para conquistar

Considerar lo siguiente respecto al pseudo-código anterior:

- En **DividirParaConquistar** $k$ es un constante que indica cuando el tamaño de una entrada $w$ es considerado pequeño: $|w| \leq k$
- Las entradas pequeñas son solucionadas utilizado un algoritmo diseñado para ellas: **InstanciasPequeñas**
  - En general este algoritmo es sencillo y ejecuta un número pequeño de operaciones.
- Si el tamaño de una entrada $w$ no es pequeño ($|w| > k$), entonces $w$ es dividido es una secuencia $w_1, \ldots, w_\ell$ de entradas de menor tamaño para **DividirParaConquistar**

# Dividir para conquistar

- Para cada $i \in \{1, \ldots, \ell\}$ la llamada **DividirParaConquistar**($w_i$) resuelve el problema para la entrada $w_i$
  - El resultado de esta llamada es almacenado en $S_i$
- Finalmente la llamada **Combinar**($S_1$, ..., $S_\ell$) combina los resultados $S_1$, ..., $S_\ell$ para obtener el resultados para $w$.

Vimos un ejemplo de esta técnica en el algoritmo de **búsqueda binaria**.

Vamos a ver como utilizar esta técnica para obtener un algoritmo eficiente para la **multiplicación de dos números enteros**.

# Antes de multiplicar: suma de números enteros

Sean $a, b \in \mathbb{Z}$ con $n \geq 1$ dígitos cada uno. Sea

$$c = a + b.$$

Considere **el algoritmo usual de la suma** para calcular $c$.

Consideramos la **suma de dos dígitos**, **comparación de dos dígitos** y **resta de un número con a lo más dos dígitos con uno de un dígito** como las operaciones a contar, cada una con costo 1.

## Preguntas

1. ¿Tiene sentido suponer que todas tienen el mismo costo?
2. ¿Cuál es el peor caso para el algoritmo usual?
3. ¿Cuántas operaciones realiza el algoritmo en el peor caso?
4. ¿Cuántos dígitos puede tener $c$?

¿Se puede **sumar** más rápido que $\mathcal{O}(n)$?

# Multiplicación de números enteros

Sean $a, b \in \mathbb{Z}$ con $n \geq 1$ dígitos cada uno. Sea

$$d = a \cdot b$$

Considere **el algoritmo usual de la multiplicación** para calcular $d$.

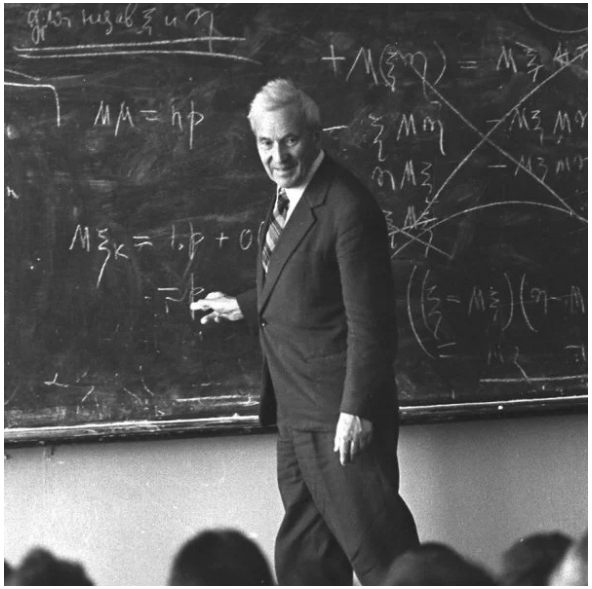
Esta vez tome **la suma** y **la multiplicación de dígitos** como las operaciones a contar, ambas con costo 1.

## Preguntas

1. ¿Tiene sentido suponer que ambas operaciones tienen el mismo costo?
2. ¿Cuál es el peor caso para el algoritmo usual?
3. ¿Cuántas operaciones realiza el algoritmo en este caso?
4. ¿Cuántos dígitos puede tener $d$?

¿Se puede **multiplicar** más rápido que $\mathcal{O}(n^2)$?

# Algoritmo de multiplicación de Karatsuba



Andréi Kolmogorov

Anatoli Karatsuba

# Algoritmo de multiplicación de Karatsuba

Sean $a, b \in \mathbb{Z}$ con $n$ dígitos cada uno, donde $n = 2^k$ para algún $k \in \mathbb{N}$.

Se puede representar $a$ y $b$ de la siguiente forma:

$$\begin{aligned} a &= a_1 \cdot 10^{\frac{n}{2}} + a_2 \\ b &= b_1 \cdot 10^{\frac{n}{2}} + b_2 \end{aligned}$$

Tenemos entonces que:

$$a \cdot b = a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2$$

# Algoritmo de multiplicación de Karatsuba

$$a \cdot b = a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2$$

Para calcular $a \cdot b$ entonces debemos calcular las siguientes multiplicaciones:

1. $a_1 \cdot b_1$
2. $a_1 \cdot b_2$
3. $a_2 \cdot b_1$
4. $a_2 \cdot b_2$

Obtenemos entonces un algoritmo recursivo

- Para resolver el caso de largo $n$ realizamos 4 llamadas para los casos de largo $\frac{n}{2}$

¿Cuál el la complejidad de este algoritmo?

**R:** Se puede usar el teorema Maestro para deducir que este algoritmo es de orden $\Theta(n^2)$.

# La idea clave en el algoritmo de Karatsuba

Podemos calcular $a \cdot b$ realizando las siguientes multiplicaciones:

1. $c_1 = a_1 \cdot b_1$
2. $c_2 = a_2 \cdot b_2$
3. $c_3 = (a_1 + a_2) \cdot (b_1 + b_2)$

Tenemos entonces que:

$$a \cdot b = c_1 \cdot 10^n + (c_3 - (c_1 + c_2)) \cdot 10^{\frac{n}{2}} + c_2$$

Esta expresión se conoce como **el algoritmo de Karatsuba**.

¿Cuántas **operaciones** realiza este algoritmo?

# Tiempo de ejecución del algoritmo de Karatsuba

Sea $T(n)$ el número de operaciones realizadas en el **peor caso** por el algoritmo de Karatsuba para dos números de entrada con $n$ dígitos cada uno.

Para determinar el orden de $T(n)$ utilizamos la siguiente **ecuación de recurrencia** (con $e \in \mathbb{N}$ una constante):

$$T(n) = \begin{cases} 1 & n = 1 \\ 3 \cdot T\left(\frac{n}{2}\right) + e \cdot n & n > 1 \end{cases}$$

# Tiempo de ejecución del algoritmo de Karatsuba

$$a \cdot b = c_1 \cdot 10^n + (c_3 - (c_1 + c_2)) \cdot 10^{\frac{n}{2}} + c_2$$

$$T(n) = \begin{cases} 1 & n = 1 \\ 3 \cdot T\left(\frac{n}{2}\right) + e \cdot n & n > 1 \end{cases}$$

## Preguntas

1. ¿Qué supuestos realizamos al formular esta ecuación?
   - $n$ es una potencia de 2 y que $(a_1 + a_2)$ y $(b_1 + b_2)$ tienen $\frac{n}{2}$ dígitos cada uno.
2. ¿Qué representa la constante $e$?
   - Calcular $(a_1 + a_2)$, $(b_1 + b_2)$, $(c_1 + c_2)$ y $(c_3 - (c_1 + c_2))$.
   - Construir $a \cdot b$ a partir de $c_1$, $c_2$ y $(c_3 - (c_1 + c_2))$, lo cual puede tomar tiempo lineal en el peor caso. ¿Por qué?

# Resolviendo la ecuación de recurrencia

Utilizando el **Teorema Maestro** obtenemos que $T(n)$ es $\Theta(n^{\log_2(3)})$.

- Pero este resultado es válido bajo los supuestos realizados anteriormente.

¿Cómo debe formularse el algoritmo de Karatsuba en el **caso general**?

# Caso general del algoritmo de Karatsuba

En el caso general, representamos las entradas $a$ y $b$ de la siguiente forma:

$$\begin{aligned} a &= a_1 \cdot 10^{\lfloor \frac{n}{2} \rfloor} + a_2 \\ b &= b_1 \cdot 10^{\lfloor \frac{n}{2} \rfloor} + b_2 \end{aligned}$$

La siguiente ecuación de recurrencia para $T(n)$ captura la cantidad de operaciones realizadas por el algoritmo (para constantes $e_1, e_2$):

$$T(n) = \begin{cases} e_1 & n \leq 3 \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lceil \frac{n}{2} \rceil + 1\right) + e_2 \cdot n & n > 3 \end{cases}$$

## Ejercicio

Demuestre usando inducción constructiva que $T(n)$ es $\mathcal{O}(n^{\log_2(3)})$

- En particular, demuestre lo siguiente:
  $\exists c \in \mathbb{R}^+.\ \exists d \in \mathbb{R}^+.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ T(n) \leq c \cdot n^{\log_2(3)} - d \cdot n$