

# Análisis de algoritmos (parte 3)

Clase 18

IIC 1253

Prof. Miguel Romero

# Outline

**Introducción**

Recurrencias y funciones armónicas

Teorema maestro

Epílogo

# Objetivos de la clase

- Deducir complejidad sin substituciones
- Conocer el teorema maestro de algoritmos

# Outline

Introducción

**Recurrencias y funciones armónicas**

Teorema maestro

Epílogo

# Ecuaciones de recurrencia: otra técnica

## Definición

Una función  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  es **asintóticamente no decreciente** si:

$$(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq f(n+1))$$

## Ejemplos

Las funciones  $\log_2(n)$ ,  $n$ ,  $n^k$  y  $2^n$  son asintóticamente no decrecientes.

# Ecuaciones de recurrencia: otra técnica

## Definición

Dado un natural  $b > 0$ , una función  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  es  **$b$ -armónica** si  $f(b \cdot n) \in O(f)$ .

## Ejemplos

Las funciones  $\log_2(n)$ ,  $n$  y  $n^k$  son  $b$ -armónicas para cualquier  $b$ .

La función  $2^n$  no es 2-armónica (porque  $2^{2n} \notin O(2^n)$ ).

Por contradicción, si  $2^{2n} \in O(2^n)$  esto significa que

$$\exists n_0, \exists c, \forall n \geq n_0 \quad 2^{2n} \leq c \cdot 2^n$$

$$2^{n+n} \leq c \cdot 2^n$$

$$2^n \cdot 2^n \leq c \cdot 2^n$$

$$2^n \leq c$$

Como  $c$  es una constante, no importa qué tan grande sea, siempre podemos tomar un  $n$  lo suficientemente grande.

# Ecuaciones de recurrencia: otra técnica

Sean  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ , un natural  $b > 1$  y

$$POTENCIA_b = \{b^i \mid i \in \mathbb{N}\}$$

## Teorema

Si  $f, g$  son asintóticamente no decrecientes,  $g$  es  $b$ -armónica y  $f \in O(g \mid POTENCIA_b)$ , entonces  $f \in O(g)$ .

Este teorema nos permite deducir lo que queríamos,  
sin tener que deducir  $n_0$  y  $c$

## Ejercicio

Demuestre el teorema.

# Ecuaciones de recurrencia: otra técnica

## Teorema

Si  $f, g$  son asintóticamente no decrecientes,  $g$  es  $b$ -armónica y  $f \in O(g \mid POTENCIA_b)$ , entonces  $f \in O(g)$ .

Como  $f$  es asintóticamente no decreciente:

$$(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq f(n+1)) \quad (1)$$

Como  $g$  es asintóticamente no decreciente:

$$(\exists n_1 \in \mathbb{N})(\forall n \geq n_1)(g(n) \leq g(n+1)) \quad (2)$$

Como  $f \in O(g \mid POTENCIA_b)$ :

$$(\exists c \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)(n \in POTENCIA_b \rightarrow f(n) \leq c \cdot g(n)) \quad (3)$$



# Ecuaciones de recurrencia: otra técnica

## Teorema

Si  $f, g$  son asintóticamente no decrecientes,  $g$  es  $b$ -armónica y  $f \in O(g \mid POTENCIA_b)$ , entonces  $f \in O(g)$ .

Como  $g$  es  $b$ -armónica:  $g(b \cdot n) \in O(g(n))$ :

$$(\exists d \in \mathbb{R}^+)(\exists n_3 \in \mathbb{N})(\forall n \geq n_3)(g(b \cdot n) \leq d \cdot g(n)) \quad (4)$$

Tomamos  $n_4 = \max\{1, n_0, n_1, n_2, n_3\}$ . Sea  $n \geq n_4$ . La idea es sustituir por potencias de  $b$ , para poder usar todas las ecuaciones. Para esto, vamos a acotar  $n$  entre potencias de  $b$ . Como  $n \geq 1$ , existe  $k \geq 0$  tal que

$$b^k \leq n < b^{k+1} \quad (5)$$

# Ecuaciones de recurrencia: otra técnica

## Teorema

Si  $f, g$  son asintóticamente no decrecientes,  $g$  es  $b$ -armónica y  $f \in O(g \mid POTENCIA_b)$ , entonces  $f \in O(g)$ .

Como  $n < b^{k+1}$ , de (1):  $f(n) \leq f(b^{k+1})$

De (3):  $f(b^{k+1}) \leq c \cdot g(b^{k+1})$

De (5) multiplicando por  $b$ :  $b^{k+1} \leq b \cdot n$

De (2):  $g(b^{k+1}) \leq g(b \cdot n)$

De (4):  $g(b \cdot n) \leq d \cdot g(n)$

# Ecuaciones de recurrencia: otra técnica

## Teorema

Si  $f, g$  son asintóticamente no decrecientes,  $g$  es  $b$ -armónica y  $f \in O(g \mid POTENCIA_b)$ , entonces  $f \in O(g)$ .

Combinando todo lo anterior:

$$f(n) \leq f(b^{k+1}) \leq c \cdot g(b^{k+1}) \leq c \cdot g(b \cdot n) \leq c \cdot d \cdot g(n)$$

Por lo tanto:

$$\forall n \geq n_4, f(n) \leq (c \cdot d) \cdot g(n)$$

y entonces  $f \in O(g)$ .

# Ecuaciones de recurrencia: otra técnica

Volviendo al ejemplo de `BinarySearch`...

## Ejercicio

Demuestre que  $T \in O(\log n)$  usando el teorema anterior.

Algunas observaciones:

- Ya sabemos que  $T \in O(\log_2(n) \mid POTENCIA_2)$ .
- Ya sabemos que  $\log_2(n)$  es asintóticamente no decreciente y 2-armónica.

Nos falta demostrar que  $T$  es asintóticamente no decreciente...  
usaremos inducción

# Ecuaciones de recurrencia: otra técnica

## Ejercicio

Demuestre que

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

es asintóticamente no decreciente.

Necesitamos lo siguiente:

PD:  $T(1) \leq T(n)$ , para todo  $n \geq 1$ .

Inducción fuerte en  $n \geq 1$ :

**BI:** Para  $n = 1$  es directo.

**HI:** Para  $n \geq 2$ , supongamos que  $T(1) \leq T(k)$ , para todo  $k \in \{1, \dots, n-1\}$ .

**TI:** Para  $n \geq 2$ , tenemos que  $1 \leq \lfloor \frac{n}{2} \rfloor < n$ , luego:

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + 4 \geq T(1) + 4 \geq T(1).$$

# Ecuaciones de recurrencia: otra técnica

## Ejercicio

Demuestre que

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

es asintóticamente no decreciente.

Ahora sí:

PD:  $(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq T(n+1))$

Tomamos  $n_0 = 1$ . PD:  $(\forall n \geq 1)(T(n) \leq T(n+1))$

Para facilitar la demostración vamos a demostrar un resultado más fuerte:

PD:  $(\forall n \geq 1)(T(m) \leq T(n))$ , con  $1 \leq m \leq n$

De este resultado se deduce que  $T(n)$  es asintóticamente no decreciente (en lugar de demostrarlo para el antecesor, lo hacemos para todos los anteriores).

## Ecuaciones de recurrencia: otra técnica

### Ejercicio

Demuestre que

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

es asintóticamente no decreciente.

PD:  $(\forall n \geq 1)(T(m) \leq T(n))$ , con  $1 \leq m \leq n$

Demostraremos esto por inducción fuerte sobre  $n$ :

**BI:** Para  $n = 1$  es directo.

**HI:** Para  $n \geq 2$ , supongamos que  $\forall k \in \{1, \dots, n-1\}$  se cumple que

$$T(m) \leq T(k) \text{ , con } 1 \leq m \leq k$$

## Ecuaciones de recurrencia: otra técnica

**TI:** PD:  $T(m) \leq T(n)$ , donde  $n \geq 2$  y  $1 \leq m \leq n$ .

Supongamos primero que  $m = 1$ . Por lo que probamos al principio, tenemos que  $T(1) \leq T(n)$ .

Supongamos ahora que  $m \geq 2$ . Tenemos que  $1 \leq \lfloor \frac{n}{2} \rfloor < n$ , y  $1 \leq \lfloor \frac{m}{2} \rfloor \leq \lfloor \frac{n}{2} \rfloor$  y entonces podemos aplicar la HI:

$$\begin{array}{rcl} T\left(\left\lfloor \frac{m}{2} \right\rfloor\right) & \leq & T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ T\left(\left\lfloor \frac{m}{2} \right\rfloor\right) + 4 & \leq & T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 \\ T(m) & \leq & T(n) \end{array}$$





# Outline

Introducción

Recurrencias y funciones armónicas

**Teorema maestro**

Epílogo

# Dividir para conquistar

- Muchos algoritmos conocidos y usados en la práctica se basan en dividir el input en instancias más pequeñas para resolverlas recursivamente.
- Típicamente, existe un umbral  $n_0$  desde el cual se resuelve recursivamente el problema (es decir, para inputs de tamaño  $n \geq n_0$ ).
- Se divide el input por una constante  $b$  y se aproxima a un entero (usando  $\lfloor \cdot \rfloor$  o  $\lceil \cdot \rceil$ ), haciendo  $a_1$  y  $a_2$  llamadas recursivas para cada caso.
- Además, en general se hace un procesamiento adicional antes o después de las llamadas recursivas, que llamaremos  $f(n)$ .

# Dividir para conquistar: un ejemplo

## Ejercicio

¿Cómo ordenamos dos listas ya ordenadas en una?

$$L_1 = \{4, 7, 17, 23\}$$

$$L_2 = \{1, 9, 10, 15\}$$

¿Cómo podemos ocupar esta técnica para ordenar una lista?

¿Cuál es la complejidad de este algoritmo?

# Dividir para conquistar: un ejemplo

## Ejercicio

¿Cómo ordenamos dos listas ya ordenadas en una?

¿Cuál es la complejidad de este algoritmo?

Recorremos ambas, comparando el primer elemento. En cada paso ponemos el menor de ellos en una nueva lista y avanzamos. Si alguna de las listas se acaba, ponemos lo que quede de la otra al final.

En el peor caso, recorremos ambas listas comparando uno por uno sus elementos, con lo que hacemos  $n - 1$  comparaciones.

# Dividir para conquistar: un ejemplo

## Ejercicio

¿Cómo podemos ocupar esta técnica para ordenar una lista?

Suponiendo que tenemos un método **Combinar** que implementa el procedimiento visto anteriormente:

**input** : Arreglo  $A[0, \dots, n-1]$ , largo  $n$

**output**: Arreglo ordenado

MergeSort( $A, n$ ):

```
1  if  $n \leq 1$  then
2      return  $A$ 
3  else
4       $m \leftarrow \lfloor n/2 \rfloor$ 
5       $A_1 \leftarrow \text{MergeSort}(A[0 \dots m-1], m)$ 
6       $A_2 \leftarrow \text{MergeSort}(A[m \dots n-1], n-m)$ 
7      return  $\text{Combinar}(A_1, A_2)$ 
```

¿Cómo obtenemos la complejidad? ¿Habrá algún método adicional?

# Teorema Maestro

## Teorema

Si  $a_1, a_2, b, c, c_0, d \in \mathbb{R}^+$  y  $b > 1$ , entonces para una recurrencia de la forma

$$T(n) = \begin{cases} c_0 & 0 \leq n < n_0 \\ a_1 \cdot T\left(\lceil \frac{n}{b} \rceil\right) + a_2 \cdot T\left(\lfloor \frac{n}{b} \rfloor\right) + c \cdot n^d & n \geq n_0 \end{cases}$$

se cumple que

$$T(n) \in \begin{cases} \Theta(n^d) & a_1 + a_2 < b^d \\ \Theta(n^d \cdot \log(n)) & a_1 + a_2 = b^d \\ \Theta(n^{\log_b(a_1 + a_2)}) & a_1 + a_2 > b^d \end{cases}.$$

¿Qué valor debe tomar  $n_0$ ?

# Teorema Maestro

La idea es que las llamadas recursivas deben ser más pequeñas que la original, para que el algoritmo termine. Esto significa que

$$\left\lfloor \frac{n}{b} \right\rfloor < n \quad \text{y} \quad \left\lceil \frac{n}{b} \right\rceil < n$$

Como  $\left\lfloor \frac{n}{b} \right\rfloor \leq \left\lceil \frac{n}{b} \right\rceil$ , basta con que  $\left\lceil \frac{n}{b} \right\rceil < n$ .

Como  $\left\lceil \frac{n}{b} \right\rceil < \frac{n}{b} + 1$ , basta con que

$$\frac{n}{b} + 1 \leq n$$

$$n + b \leq nb$$

$$b \leq nb - n$$

$$\frac{b}{b-1} \leq n$$

Es decir,  $n_0$  solo depende de  $b$

# Teorema Maestro

## Teorema

Si  $a_1, a_2, b, c, c_0, d \in \mathbb{R}^+$  y  $b > 1$ , entonces para una recurrencia de la forma

$$T(n) = \begin{cases} c_0 & 0 \leq n < \frac{b}{b-1} \\ a_1 \cdot T\left(\lceil \frac{n}{b} \rceil\right) + a_2 \cdot T\left(\lfloor \frac{n}{b} \rfloor\right) + c \cdot n^d & n \geq \frac{b}{b-1} \end{cases}$$

se cumple que

$$T(n) \in \begin{cases} \Theta(n^d) & a_1 + a_2 < b^d \\ \Theta(n^d \cdot \log(n)) & a_1 + a_2 = b^d \\ \Theta(n^{\log_b(a_1 + a_2)}) & a_1 + a_2 > b^d \end{cases}.$$

## Ejercicio

Demuestre el teorema.



# Teorema Maestro

## Ejercicio

¿Cuál es la complejidad de MergeSort?

Como vimos antes, el peor caso es que Combinar tenga que comparar todos los elementos. En tal caso, se hacen  $n - 1$  comparaciones, a la que sumamos la comparación que se hace para verificar el tamaño de la lista. Entonces, la ecuación de recurrencia para MergeSort es:

$$T(n) = \begin{cases} 1 & n < 2 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n & n \geq 2 \end{cases}$$

Aplicamos el teorema maestro:

$$a_1 = 1, a_2 = 1, b = 2, c = 1, d = 1, c_0 = 1$$

$$a_1 + a_2 = 2, b^d = 2^1 = 2 \rightarrow \text{Entramos en el segundo caso: } a_1 + a_2 = b^d$$

Por lo tanto,  $T(n) \in \Theta(n \cdot \log(n))$ .

# Outline

Introducción

Recurrencias y funciones armónicas

Teorema maestro

**Epílogo**

# Objetivos de la clase

- Deducir complejidad sin substituciones
- Conocer el teorema maestro de algoritmos