Análisis de Algoritmos

IIC1253

Complejidad de un algoritmo

Un algoritmo A puede ser pensado como una función
$A : \{0,1\}^* \to \{0,1\}^*$

- ¿Qué tan general es esta representación? ¿Qué restricciones hay que imponer sobre A?

Complejidad de un algoritmo

Un algoritmo A puede ser pensado como una función $A : \{0,1\}^* \to \{0,1\}^*$

- ¿Qué tan general es esta representación? ¿Qué restricciones hay que imponer sobre A?

Ejercicio

Represente un algoritmo para decidir si un número natural es primo con la notación anterior.

Complejidad de un algoritmo

A cada algoritmo A asociamos una función $\text{tiempo}_A : \{0,1\}^* \to \mathbb{N}$ tal que:

$$\text{tiempo}_A(w)\text{: número de pasos realizados por } A \text{ con entrada } w$$

¿Qué es un "paso" de un algoritmo? ¿Qué operaciones debemos considerar?

Complejidad de un algoritmo

A cada algoritmo A asociamos una función $\text{tiempo}_A : \{0,1\}^* \to \mathbb{N}$ tal que:

$\text{tiempo}_A(w)$: número de pasos realizados por A con entrada w

¿Qué es un "paso" de un algoritmo? ¿Qué operaciones debemos considerar?

Ejercicio

En general, un algoritmo puede ser pensado como una función $A : \Gamma^* \to \Gamma^*$, donde Γ es un alfabeto finito.

De un algoritmo *Ord* para ordenar una lista de números naturales. Calcule la función tiempo_{Ord}.

- Suponga que el alfabeto utilizado es $\{0, 1, \#\}$
- Indique cuáles son las operaciones que deben ser consideradas

Análisis de un algoritmo en el peor caso

Para analizar un algoritmo vamos a considerar el peor caso.

Para cada algoritmo A asociamos una función $t_A : \mathbb{N} \to \mathbb{N}$ tal que

$$t_A(n) \quad = \quad \text{máx}\{\text{tiempo}_A(w) \mid w \in \{0,1\}^* \text{ y } |w| = n\},$$

donde $|w|$ es el largo de w.

Análisis de un algoritmo en el peor caso

Para analizar un algoritmo vamos a considerar el peor caso.

Para cada algoritmo A asociamos una función $t_A : \mathbb{N} \to \mathbb{N}$ tal que

$$t_A(n) \quad = \quad \text{máx}\{\text{tiempo}_A(w) \mid w \in \{0,1\}^* \text{ y } |w| = n\},$$

donde $|w|$ es el largo de w.

Ejercicio

Calcule la función t_{Ord} para el algoritmo *Ord* construido en la transparencia anterior.

Notación asintótica

En muchos casos, nos interesa conocer el *orden* de un algoritmo en lugar de su complejidad exacta.

- Queremos decir que un algoritmo es lineal o cuadrático, en lugar de decir que su complejidad es $3n^2 + 17n + 22$

Vamos a desarrollar notación para hablar del orden de un algoritmo.

Vamos a considerar funciones de la forma $f : \mathbb{N} \to \mathbb{R}_0^+$, donde $\mathbb{R}^+ = \{r \in \mathbb{R} \mid r > 0\}$ y $\mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$

- Incluyen a las funciones definidas en las transparencias anteriores, y también sirven para modelar el tiempo de ejecución de un algoritmo

La notación $O(f)$

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$

Definición

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(g(n) \leq c \cdot f(n))\}$$

La notación $O(f)$

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$

Definición

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(g(n) \leq c \cdot f(n))\}$$

Ejercicio

Demuestre que $3n^2 + 17n + 22 \in O(n^2)$

Las notaciones $\Omega(f)$ y $\Theta(f)$

Definición

$$\begin{aligned}
\Omega(f) = \ & \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\
& \qquad (\forall n \geq n_0)\,(c \cdot f(n) \leq g(n))\} \\
\Theta(f) = \ & O(f) \cap \Omega(f)
\end{aligned}$$

Las notaciones $\Omega(f)$ y $\Theta(f)$

Definición

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(c \cdot f(n) \leq g(n))\}$$

$$\Theta(f) = O(f) \cap \Omega(f)$$

Ejercicios

1. Demuestre que $3n^2 + 17n + 22 \in \Theta(n^2)$
2. Demuestre que $g \in \Theta(f)$ si y sólo si existen $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que para todo $n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$

Ejercicios

1. Sea $p(n)$ un polinomio de grado $k \geq 0$ con coeficientes en los números enteros. Demuestre que $p(n) \in O(n^k)$.

2. ¿Cuáles de las siguientes afirmaciones son ciertas?
   - $n^2 \in O(n)$
   - Si $f(n) \in O(n)$, entonces $f(n)^2 \in O(n^2)$
   - Si $f(n) \in O(n)$, entonces $2^{f(n)} \in O(2^n)$

3. Decimos que $f \preceq g$ si $f \in O(g)$. ¿Es $\preceq$ un orden parcial? ¿Es un orden total?

4. Decimos que $f \sim g$ si $f \in \Theta(g)$. Demuestre que $\sim$ es una relación de equivalencia.
   - Indique qué es $[f]_\sim$

Suponga que tiene una lista ordenada (de menor a mayor) L de números naturales

- L tiene n elementos, para referirnos al elemento i-ésimo $(1 \leq i \leq n)$ usamos la notación $L[i]$

¿Cómo podemos verificar si un número a está en L?

Ecuaciones de recurrencia: Búsqueda binaria

Para verificar si un número a está en L usamos el siguiente algoritmo:

encontrar(a, L, i, j)
  if $i > j$ **then return** no
  else if $i = j$ **then**
    if $L[i] = a$ **then return** i
    else return no
  else
    $p = \lfloor \frac{i+j}{2} \rfloor$
    if $L[p] < a$ **then return encontrar**(a, L, $p + 1$, j)
    else if $L[p] > a$ **then return encontrar**(a, L, i, $p - 1$)
    else return p

Llamada inicial al algoritmo: **encontrar**(a, L, 1, n)

Ecuaciones de recurrencia: Búsqueda binaria

¿Cuál es la complejidad del algoritmo?

- ¿Qué operaciones vamos a considerar?
- ¿Cuál es el peor caso?

Ecuaciones de recurrencia: Búsqueda binaria

¿Cuál es la complejidad del algoritmo?

- ¿Qué operaciones vamos a considerar?
- ¿Cuál es el peor caso?

Si contamos sólo las comparaciones, entonces la siguiente expresión define la complejidad del algoritmo:

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

donde $c \in \mathbb{N}$ y $d \in \mathbb{N}$ son constantes tales que $c \geq 1$ y $d \geq 1$.

¿Cuál es la complejidad del algoritmo?

- ¿Qué operaciones vamos a considerar?
- ¿Cuál es el peor caso?

Si contamos sólo las comparaciones, entonces la siguiente expresión define la complejidad del algoritmo:

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

donde $c \in \mathbb{N}$ y $d \in \mathbb{N}$ son constantes tales que $c \geq 1$ y $d \geq 1$.

Esta es una ecuación de recurrencia.

Ecuaciones de recurrencia: Búsqueda binaria

¿Cómo podemos solucionar una ecuación de recurrencia?

- Técnica básica: sustitución de variables

Ecuaciones de recurrencia: Búsqueda binaria

¿Cómo podemos solucionar una ecuación de recurrencia?

- Técnica básica: sustitución de variables

Para la ecuación anterior usamos la sustitución $n = 2^k$.

- Vamos a resolver la ecuación suponiendo que n es una potencia de 2
- Vamos a estudiar condiciones bajo las cuales el resultado para una potencia de 2 puede ser extendido a todo n
  - Estas condiciones van a servir para cualquier potencia

Si realizamos la sustitución $n = 2^k$ en la ecuación:

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

obtenemos:

$$T(2^k) = \begin{cases} c & k = 0 \\ T(2^{k-1}) + d & k > 0 \end{cases}$$

Ecuaciones de recurrencia: sustitución de variables

Extendiendo la expresión anterior obtenemos:

$$
\begin{aligned}
T(2^k) &= T(2^{k-1}) + d \\
&= (T(2^{k-2}) + d) + d \\
&= T(2^{k-2}) + 2d \\
&= (T(2^{k-3}) + d) + 2d \\
&= T(2^{k-3}) + 3d \\
&= \cdots
\end{aligned}
$$

Ecuaciones de recurrencia: sustitución de variables

Extendiendo la expresión anterior obtenemos:

$$
\begin{aligned}
T(2^k) &= T(2^{k-1}) + d \\
&= (T(2^{k-2}) + d) + d \\
&= T(2^{k-2}) + 2d \\
&= (T(2^{k-3}) + d) + 2d \\
&= T(2^{k-3}) + 3d \\
&= \cdots
\end{aligned}
$$

Deducimos la expresión general para $k - i \geq 0$:

$$T(2^k) \quad = \quad T(2^{k-i}) + i \cdot d$$

Ecuaciones de recurrencia: sustitución de variables

Considerando $i = k$ obtenemos:

$$\begin{aligned} T(2^k) &= T(1) + k \cdot d \\ &= c + k \cdot d \end{aligned}$$

Ecuaciones de recurrencia: sustitución de variables

Considerando $i = k$ obtenemos:

$$\begin{aligned} T(2^k) &= T(1) + k \cdot d \\ &= c + k \cdot d \end{aligned}$$

Dado que $k = \log_2 n$, obtenemos que $T(n) = c + d \cdot \log_2 n$ para n potencia de 2.

- Vamos a definir notación para decir esto de manera formal

Notaciones asintóticas condicionales

Sea P un predicado sobre los números naturales.

Definición

$$O(f \mid P) \;=\; \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(n \in P \to g(n) \leq c \cdot f(n))\}$$

Notaciones asintóticas condicionales

Sea P un predicado sobre los números naturales.

Definición

$$O(f \mid P) \;=\; \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(n \in P \to g(n) \leq c \cdot f(n))\}$$

Las notaciones $\Omega(f \mid P)$ y $\Theta(f \mid P)$ son definidas de manera análoga.

Búsqueda en una lista ordenada: complejidad del algoritmo

Sea $\text{POTENCIA}_2 = \{2^i \mid i \in \mathbb{N}\}$

Para la función T que define la complejidad del procedimiento **encontrar**, tenemos que:

$$T \quad \in \quad \Theta(\log_2 n \mid \text{POTENCIA}_2)$$

¿Podemos concluir que $T \in \Theta(\log_2 n)$?

- Vamos a estudiar este problema, pero antes vamos a ver un segundo ejemplo.

Ordenamiento de una lista

Ahora queremos ordenar una lista L de números naturales

- Utilizamos el algoritmo **mergesort**

Si contamos sólo las comparaciones entre elementos de L, entonces la siguiente expresión define la complejidad de **mergesort**:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + (n-1) & n > 1 \end{cases}$$

Orden de **mergesort**

Nuevamente utilizamos la sustitución $n = 2^k$, obteniendo:

$$T(2^k) \quad = \quad \begin{cases} 0 & k = 0 \\ 2 \cdot T(2^{k-1}) + (2^k - 1) & k > 0 \end{cases}$$

Desarrollando esta expresión obtenemos:

$$\begin{aligned}
T(2^k) \quad &= \quad 2 \cdot T(2^{k-1}) + (2^k - 1) \\
&= \quad 2 \cdot (2 \cdot T(2^{k-2}) + (2^{k-1} - 1)) + (2^k - 1) \\
&= \quad 2^2 \cdot T(2^{k-2}) + 2^k - 2 + 2^k - 1 \\
&= \quad 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k - (1 + 2) \\
&= \quad 2^2 \cdot (2 \cdot T(2^{k-3}) + (2^{k-2} - 1)) + 2 \cdot 2^k - (1 + 2) \\
&= \quad 2^3 \cdot T(2^{k-3}) + 2^k - 2^2 + 2 \cdot 2^k - (1 + 2) \\
&= \quad 2^3 \cdot T(2^{k-3}) + 3 \cdot 2^k - (1 + 2 + 2^2) \\
&= \quad \cdots
\end{aligned}$$

Orden de **mergesort**

Deducimos la expresión general para $k - i \geq 0$:

$$\begin{aligned} T(2^k) &= 2^i \cdot T(2^{k-i}) + i \cdot 2^k - \sum_{j=0}^{i-1} 2^j \\ &= 2^i \cdot T(2^{k-i}) + i \cdot 2^k - 2^i + 1 \end{aligned}$$

Considerando $i = k$ obtenemos:

$$\begin{aligned} T(2^k) &= 2^k \cdot T(1) + k \cdot 2^k - 2^k + 1 \\ &= k \cdot 2^k - 2^k + 1 \end{aligned}$$

Dado que $k = \log_2 n$, concluimos que:

$$T \in \Theta(n \cdot \log_2 n \mid \text{POTENCIA}_2)$$

Técnicas para solucionar ecuaciones de recurrencia

Vamos a estudiar dos técnicas que nos permitan extender la solución a una ecuación de recurrencia encontrada usando sustitución de variables.

- Nos vamos a concentrar en la notación O

Primera técnica:

- Utilizamos una sustitución de variable para encontrar una solución en un caso particular.
  - Por ejemplo: $T \in O(n \cdot \log_2 n \mid \text{POTENCIA}_2)$
- Usamos inducción para demostrar que la solución es válida en el caso general

Construcción de soluciones usando inducción

Consideremos la ecuación de recurrencia para el algoritmo de búsqueda en listas ordenadas:

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

Sabemos que $T \in O(\log_2 n \mid \text{POTENCIA}_2)$

- Queremos demostrar entonces que:

$$(\exists e \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)\,(T(n) \leq e \cdot \log_2 n)$$

Construcción de soluciones usando inducción

Veamos algunos valores de la función T para estimar e y n_0:

$$\begin{array}{rcl} T(1) & = & c \\ T(2) & = & c + d \\ T(3) & = & c + d \\ T(4) & = & c + 2d \end{array}$$

En este caso necesitamos que $n_0 \geq 2$ y $e \geq (c + d)$

- ¿Por qué?

Intentamos entonces con $n_0 = 2$ y $e = (c + d)$

- Por demostrar: $(\forall n \geq 2)\,(T(n) \leq (c + d) \cdot \log_2 n)$

Inducción fuerte

¿Cuál es el principio de inducción adecuado para el problema anterior?

- Tenemos n_0 como punto de partida
- n_0 es un caso base, pero podemos tener otros
- Dado $n > n_0$ tal que n no es un caso base, suponemos que la propiedad se cumple para todo $k \in \{n_0, \ldots, n-1\}$

Inducción fuerte: Ejemplo

Queremos demostrar que $(\forall n \geq 2)\,(T(n) \leq (c + d) \cdot \log_2 n)$

Inducción fuerte: Ejemplo

Queremos demostrar que $(\forall n \geq 2)\,(T(n) \leq (c+d) \cdot \log_2 n)$

- 2 es el punto de partida y el primer caso base

Inducción fuerte: Ejemplo

Queremos demostrar que $(\forall n \geq 2)\,(T(n) \leq (c+d) \cdot \log_2 n)$

- 2 es el punto de partida y el primer caso base
- También 3 es un caso base ya que $T(3) = T(1) + d$ y para $T(1)$ no se cumple la propiedad

Inducción fuerte: Ejemplo

Queremos demostrar que $(\forall n \geq 2)\,(T(n) \leq (c + d) \cdot \log_2 n)$

- 2 es el punto de partida y el primer caso base
- También 3 es un caso base ya que $T(3) = T(1) + d$ y para $T(1)$ no se cumple la propiedad
- Para $n \geq 4$ tenemos que $T(n) = T(\lfloor \frac{n}{2} \rfloor) + d$ y $\lfloor \frac{n}{2} \rfloor \geq 2$, por lo que resolvemos este caso de manera inductiva
  - Suponemos que la propiedad se cumple para todo $k \in \{2, \ldots, n - 1\}$

Inducción fuerte: Ejemplo

Vamos a demostrar entonces que para

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

se tiene que $(\forall n \geq 2)\,(T(n) \leq (c + d) \cdot \log_2 n)$

Inducción fuerte: Ejemplo

Vamos a demostrar entonces que para

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

se tiene que $(\forall n \geq 2)\,(\,T(n) \leq (c+d) \cdot \log_2 n)$

Casos base:

$$\begin{array}{ccccc} T(2) & = & c + d & = & (c+d) \cdot \log_2 2 \\ T(3) & = & c + d & < & (c+d) \cdot \log_2 3 \end{array}$$

Inducción fuerte: Ejemplo

Vamos a demostrar entonces que para

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

se tiene que $(\forall n \geq 2)\,(T(n) \leq (c + d) \cdot \log_2 n)$

Casos base:

$$\begin{array}{rclcl} T(2) & = & c + d & = & (c + d) \cdot \log_2 2 \\ T(3) & = & c + d & < & (c + d) \cdot \log_2 3 \end{array}$$

Caso inductivo: Suponemos que $n \geq 4$ y para todo $k \in \{2, \ldots, n - 1\}$:

$$T(k) \leq (c + d) \cdot \log_2 k$$

Inducción fuerte: Ejemplo

Tenemos que:

$$
\begin{aligned}
T(n) &= T(\lfloor \frac{n}{2} \rfloor) + d \\
&\leq (c+d) \cdot \log_2 \lfloor \frac{n}{2} \rfloor + d \\
&\leq (c+d) \cdot \log_2 \frac{n}{2} + d \\
&= (c+d) \cdot \log_2 n - (c+d) + d \\
&= (c+d) \cdot \log_2 n - c \\
&< (c+d) \cdot \log_2 n
\end{aligned}
$$

Para definir esta segunda técnica necesitamos definir algunos términos.

Una función $f : \mathbb{N} \to \mathbb{R}_0^+$ es asintóticamente no decreciente si:

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

Para definir esta segunda técnica necesitamos definir algunos términos.

Una función $f : \mathbb{N} \to \mathbb{R}_0^+$ es asintóticamente no decreciente si:

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

Ejemplo

Las funciones $\log_2 n$, n, n^5 y 2^n son asintóticamente no decrecientes

Funciones b-armónicas

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$ y $b \in \mathbb{N}$ tal que $b > 0$

- f es una función b-armónica si $f(b \cdot n) \in O(f(n))$

Funciones b-armónicas

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$ y $b \in \mathbb{N}$ tal que $b > 0$

- f es una función b-armónica si $f(b \cdot n) \in O(f(n))$

Ejemplo

Las funciones $\log_2 n$, n y n^5 son b-armónicas para cualquier b ($b > 0$). La función 2^n no es 2-armónica.

Una segunda técnica: extensión de soluciones

Sean $f, g : \mathbb{N} \to \mathbb{R}_0^+$, $b \in \mathbb{N}$ tal que $b > 0$, y

$$\text{POTENCIA}_b = \{b^i \mid i \in \mathbb{N}\}$$

Una segunda técnica: extensión de soluciones

Sean $f, g : \mathbb{N} \to \mathbb{R}_0^+$, $b \in \mathbb{N}$ tal que $b > 0$, y

$$\text{POTENCIA}_b = \{b^i \mid i \in \mathbb{N}\}$$

Teorema

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

Una segunda técnica: demostración del teorema

Como f es asintóticamente no decreciente:

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

Como g es asintóticamente no decreciente:

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

Como $f \in O(g \mid \text{POTENCIA}_b)$:

$$(\exists c \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)$$
$$(n \in \text{PONTENCIA}_b \rightarrow f(n) \leq c \cdot g(n))$$

Una segunda técnica: demostración del teorema

Como g es b-armónica:

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

Sea $n_4 = \max\{1, n_0, n_1, n_2, n_3\}$ y $n \geq n_4$

Como $n \geq 1$, existe $k \geq 0$ tal que:

$$b^k \;\leq\; n \;<\; b^{k+1}$$

Como $n \geq n_0$ y f es asintóticamente no decreciente:

$$f(n) \;\leq\; f(b^{k+1})$$

Una segunda técnica: demostración del teorema

Como $n \geq n_2$ y $f \in O(g \mid \text{POTENCIA}_b)$:

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

Dado que $b^k \leq n$, se tiene que $b^{k+1} \leq b \cdot n$. Así, dado que $n \geq n_1$ y g es asintóticamente no decreciente:

$$g(b^{k+1}) \;\; \leq \;\; g(b \cdot n)$$

Finalmente, dado que $n \geq n_3$ y g es b-armónica:

$$g(b \cdot n) \;\; \leq \;\; d \cdot g(n)$$

Una segunda técnica: demostración del teorema

Combinando los resultados anteriores obtenemos:

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

Por lo tanto: $(\forall n \geq n_4)\,(f(n) \leq (c \cdot d) \cdot g(n))$

Concluimos que $f \in O(g)$.

Una segunda técnica: el caso de **mergesort**

La siguiente expresión define la complejidad de **mergesort**:

$$T(n) \quad = \quad \begin{cases} 0 & n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + (n-1) & n > 1 \end{cases}$$

Ya demostramos que $T \in O(n \cdot \log_2 n \mid \text{POTENCIA}_2)$

- Usando el último teorema, vamos a demostrar que $T \in O(n \cdot \log_2 n)$

Una segunda técnica: el caso de **mergesort**

Ejercicio

1. Demuestre que $T(n)$ y $n \cdot \log_2 n$ son asintóticamente no decrecientes
   - Puede usar inducción fuerte para demostrar que $T(n)$ es asintóticamente no decreciente
2. Demuestre que $n \cdot \log_2 n$ es 2-armónica
3. Use los ejercicios anteriores para concluir que $T \in O(n \cdot \log_2 n)$