LAUREATE
INTERNATIONAL UNIVERSITIES

UNIVERSIDAD PRIVADA DEL NORTE
Laureate International Universities®

ANÁLISIS Y DISEÑO DE ALGORITMOS

FUNDAMENTOS

CLASE N°1

Leissi M. Castañeda León

lcl@upnorte.edu.pe

https://sites.google.com/site/leissicl/

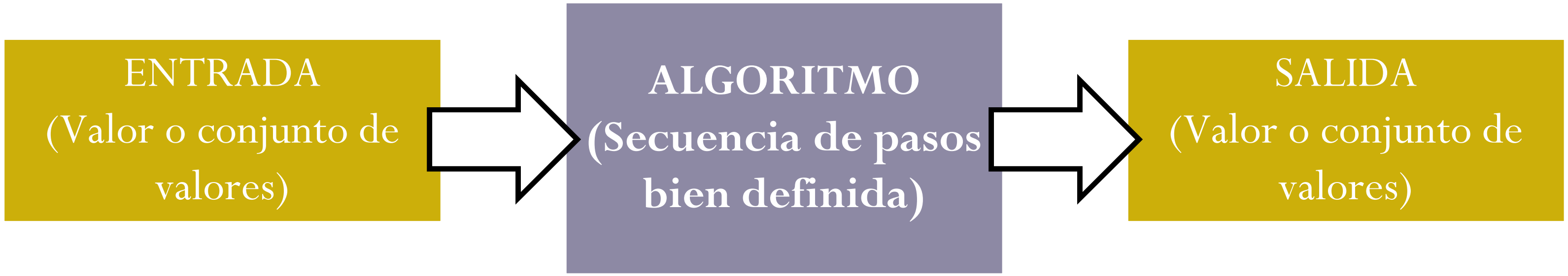
Que veremos hoy?

1. Rol de los Algoritmos en Computación
    a) ¿Que són los algoritmos?
    b) ¿Por qué el estudio de los algoritmos?, ¿vale la pena?
    c) ¿Cuál es la función de los algoritmos en relación a otras tecnologías usadas en computadores?
2. Análisis de los algoritmos
3. Diseño de algoritmos

1. Rol de los Algoritmos en Computación

1.1 Algoritmo

- Procedimiento computacional



- Herramienta para resolver problemas computacionales

Conceptos relacionados

- *Instancia de un problema*: Entrada (que satisface las restricciones del problema) para la que se necesita calcular la solución del problema.

- *Correctitud del algoritmo*: Para cada posible entrada, retorna la salida correcta.

Ejemplo

- Problema del Ordenamiento: Cuando deseamos ordenar en orden creciente una secuencia de números.
  - Entrada: Una secuencia de números <a1,a2,…,an>
  - Procedimiento: *Algoritmo de ordenación*
  - Salida: el reordenamiento (a1',a2',…an') de la secuencia de entrada, tal que a1'≤ a2'≤ … ≤ an'

  - Entrada (Instancia): <31,41,59,26,41,58>
  - Salida: <26,31,41,41,58,59>

¿Qué clase de problemas resuelven los algoritmos?

- Proyecto del Genoma Humano
- Internet: Acceso y recuperación de grandes cantidades de información
- Comercio electrónico: Almacenamiento de información, criptografía de clave pública y firma digital
- Manufactura: optimización del uso de recursos

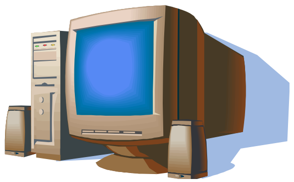
1.2 Algoritmos como tecnología

- Imagine que las computadoras fueran infinitamente rápidas y las memorias fueran gratuitas.

- ¿Cree Usted que existiría alguna razón para estudiar algoritmos?

- Tecnología:
  - Computadores rápidas, pero no infinitamente
  - Memorias baratas, pero no gratuitas
- Algoritmos deben ser eficientes
  - Tiempo de computación es recurso limitado, al igual que el espacio en memoria.
- Algoritmos a nivel de HW : Tecnología
- Desempeño del sistema:

+



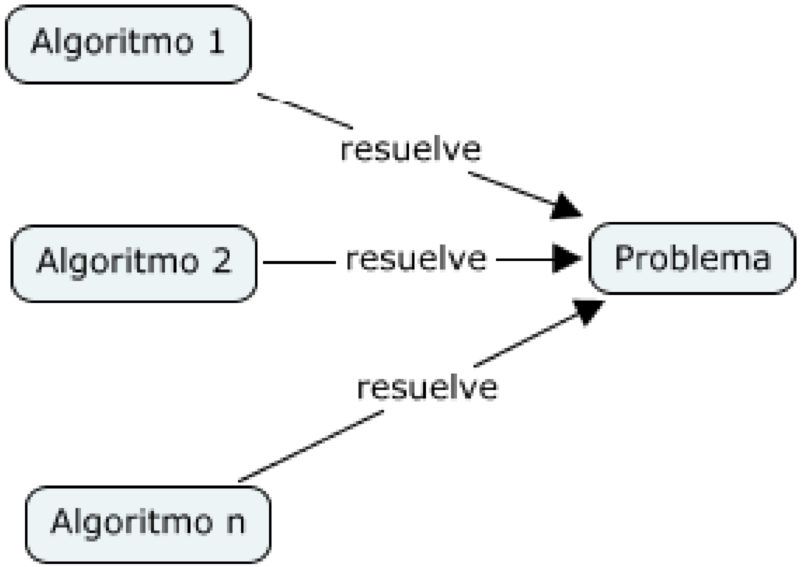
Algoritmos eficientes HW rápido

¿Por qué estudiar algoritmos y sus desempeño?

- Los algoritmos ayudan a entender la escalabilidad.
- Marcan el límite entre lo posible y lo imposible
- Matemáticas algorítmicas brindan un lenguaje para expresar el comportamiento de un programa.
- El desempeño de los programas se puede generalizar a otros recursos computacionales.

Eficiencia

- Diferentes algoritmos para resolver problema
- Diferente grado de eficiencia en cada algoritmo



Ejemplo:

Caso: ordenar un millón de números

- Pc A:
  - Ordenación por Inserción: c1.n^2
  - Más rápida: mil millones de instrucciones por segundo (1 000 000 000 = 10^9)
  - Programado en lenguaje máquina (c1=2)
- Pc B:
  - Ordenación por Merge sort: c2.nlogn
  - Menos rápida: diez millones de instr./seg. (10 000 000 = 10^7) => A 100 veces más rápida que B
  - Programado en alto nivel (c2=50)

Ejemplo (cont.)

Caso: Ordenar un millón de números

- Pc A:

$$\frac{2 \cdot \left(10^6\right)^2 \cdot instrucciones}{10^9 instrucciones / seg} = 2000\ seg$$

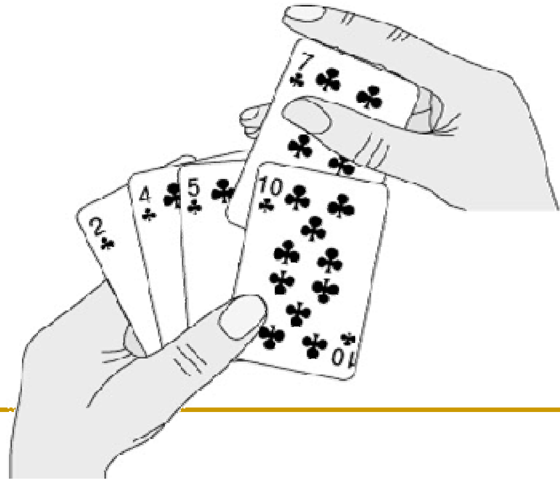
- Pc B:

$$\frac{2 \cdot 10^6 \cdot \log 10^6 \cdot instrucciones}{10^7 instrucciones / seg} \approx 100\ seg$$

2. Análisis de algoritmos

Caso de Estudio: Ordenación por Inserción

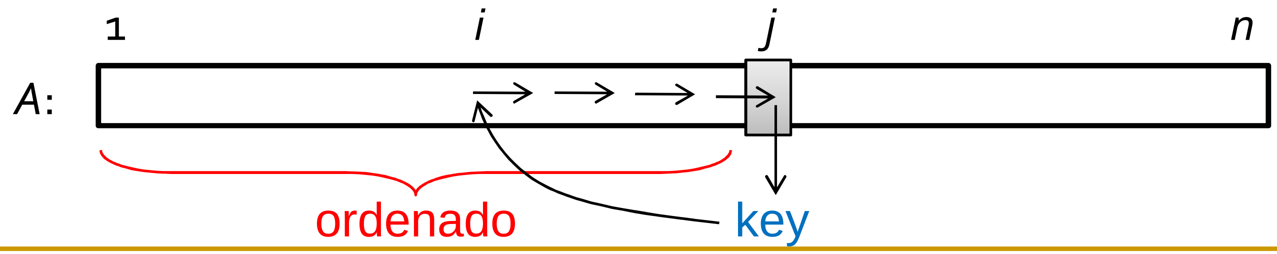
- Los números a ordenar se denominarán *keys.*
- Procedimiento **INSERTION-SORT**
  - **Entrada:** Arreglo *A*[1… *n*] con *n* elementos (n = length[A]).
  - Los números son ordenadas dentro del mismo arreglo
  - **Salida:** Arreglo *A*[1… *n*] con *n* elementos ordenados.
- Simula ordenar un mazo de cartas



Pseudocódigo

- **INSERTION-SORT(*A*)**

```
for j ← 2 to length[A]
    do key ← A[j]
        // Inserta A[j] en la secuencia ordenada A[1 · · j - 1].
        i ← j - 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
                i ← i - 1
        A[i + 1] ← key
```



Ejemplo: Dado $A = \langle 8, 2, 4, 9, 3, 6 \rangle$

8 2 4 9 3 6

8 2 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6
2	3	4	6	8	9

Invariante del Bucle

- En el algoritmo:
  - *j* indica la “carta actual"
  - Al inicio de cada iteración del **for** más externo (indizado por j) ,
    - Subarreglo $A[1 \cdot\cdot j - 1]$ representa las cartas ordenadas
    - Subarreglo $A[j + 1 \cdot\cdot n]$ representa las cartas por ordenar

- **Invariante del Bucle**: elementos $A[1 \ldots j - 1]$ son elementos en las posiciones 1 hasta $j - 1$, pero ordenados.

Invariante del Bucle

- Ayuda a entender por qué un algoritmo es correcto.
- Propiedades:
  1) ***Inicialización***: Es VERDAD antes de la primera iteración del bucle.
  2) ***Mantenimiento***: Si es VERDAD antes de la iteración del bucle, se mantiene VERDAD antes de la siguiente iteración.
  3) ***Terminación***: Cuando el bucle termina, el invariante provee una propiedad útil que demuestra que el algoritmo es correcto.
- 1) y 2) siguen el principio de inducción matemática, son el caso base y el paso inductivo, respectivamente.
- 3) es el más importante, demuestra la correctitud y se diferencia de la inducción matemática (infinita) , la inducción se detiene cuando termina el bucle.

Correctitud de la O. Inserción

- **Inicialización:** Antes de la primera iteración
  - $j = 2$.
  - Subarreglo $A[1 \ldots j - 1]$ = elemento $A[1]$
  - $A[1]$ está ordenado
- **Mantenimiento:**
  - El bucle **for** externo desplaza $A[j - 1]$, $A[j - 2]$, $A[j - 3]$, hasta encontrar la posición correcta para $A[j]$.
  - $A[j]$ se inserta manteniendo el orden.
- **Terminación:**
  - El bucle **for** externo termina cuando $j > n$,($j = n + 1$).
  - Reemplazando $n + 1$ en j del invariante del bucle, el subarreglo $A[1 \ldots n]$ contiene los elementos de $A[1 \ldots n]$, ordenados.
  - Todo el arreglo se encuentra ordenado.

Objetivos del Análisis

- Predecir los recursos que requiere el algoritmo (memoria, ancho de banda, tiempo)
- El análisis de algoritmos permite seleccionar el algoritmo más efectivo que soluciona un problema
- Antes del análisis, definir el modelo de la tecnología de implementación, incluyendo un modelo de recursos y sus costos. Se asumirá una máquina de acceso aleatorio (RAM) de un procesador.

Tamaño de la Entrada

- La noción depende del problema.
- Será el número de elementos en la entrada, en algoritmos como ordenamientos, el cálculo de la DFT.
- Será el número de bits necesarios para representar la entrada, en algoritmos como: multiplicación de dos enteros.
- Casos especiales: representarse con dos números. Ejemplo: Grafos: número de nodos y número de aristas.

Tiempo de Ejecución

- Número de operaciones primitivas o “pasos” ejecutados.
- Se parametriza el tiempo de ejecución por el tamaño de la entrada y se ignoran las constantes dependientes de la máquina.
- Cada línea del pseudocódigo requiere de una cantidad de tiempo constante.
- La ejecución de la i-ésima línea requiere de una cantidad c_i de tiempo, donde c_i es constante.
- Entradas de tamaño más pequeño requerirán de menos tiempo de ejecución que las de tamaño grande.
- Se buscan cotas superiores, todos prefieren tener una garantía.

Tipos de Análisis

- **Peor Caso (Worst-case) :**
  - *T(n) = tiempo máximo* del algoritmo, para entrada de tamaño *n*
  - Cota superior del tiempo de ejecución para cualquier entrada, y garantiza que ninguna entrada tomará más del tiempo indicado.
- **Caso Promedio (Average-case):**
  - *T(n) = tiempo esperado* del algoritmo, para entradas de tamaño *n*
  - Se asumen ciertas características acerca la distribución de probabilidades de las entradas
  - Es usualmente, tan malo como el Peor Caso.
- **Mejor Caso (Best-case) :**
  - Toma ventaja de alguna entrada para la que un algoritmo lento trabaja rápido

Caso: Procedimiento INSERTION-SORT

- El tiempo que tome la O. Inserción depende de la entrada
- Dos secuencias de entrada del mismo tamaño tomarán tiempos diferentes dependiendo de qué tan ordenados se encuentren.
- Se empieza el análisis del algoritmo indicando el “costo de tiempo” del algoritmo y las veces que se repite.
- Para cada $j = 2, 3, \ldots, n$, donde $n = length[A]$, t_j sera el número de veces que se prueba la condición del bucle **while** para el valor de j
- Cuando el bucle de **for** o **while** termina normalmente, la prueba se ejecuta una vez más que el cuerpo del bucle.
- Los comentarios no son instrucciones ejecutables, no requieren de tiempo.

Caso: Procedimiento INSERTION-SORT

	INSERTION-SORT(*A*)	*cost*	*times*
1	**for** $j \leftarrow 2$ **to** *length*[*A*]	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n - 1$
3	// Insert $A[j]$ into the sorted sequence $A[1 \cdot\cdot j - 1]$	0	$n - 1$
4	$i \leftarrow j - 1$	c_4	$n - 1$
5	**while** $i > 0$ and $A[i] >$ key	c_5	$\sum_{j=2}^{n} t_j$
6	**do** $A[i + 1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i \leftarrow i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i + 1] \leftarrow key$	c_8	$n - 1$

Caso: Procedimiento INSERTION-SORT

- El tiempo de ejecución del algoritmo es la suma de los tiempos de cada instrucción ejecutada; una instrucción que toma itera *n* veces y toma un tiempo c_i, contribuirá con $c_i\, n$
- *T(n)* del INSERTION-SORT será :

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) - c_8 (n-1).$$

- Entradas del mismo tamaño pueden tener tiempos de ejecución diferentes.

Análisis de Casos: INSERTION-SORT

- MEJOR CASO:
  - Ocurre cuando el arreglo se encuentra ordenado.
  - Para cada $j = 2, 3, \ldots, n$; $A[i] \leq key$ ($i = j - 1$). Así, $t_j = 1$ para $j = 2, 3, \ldots, n$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).
\end{aligned}
$$

  - $T(n)$ puede expresarse como $an + b$ para a y b constantes; por tanto, $T(n)$ es una función lineal de n.

Análisis de Casos: INSERTION-SORT

- PEOR CASO:
  - El arreglo *A* se encuentra en orden inverso (decreciente).
  - Cada elemento A[j] se comparará con cada elemento del subarreglo *A*[1… *j* - 1], tal que $t_j = j$ para $j = 2, 3, \ldots, n$.

$$\sum_{j=2}^{n} j = \frac{n(n-1)}{2} - 1$$

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n-1)}{2} - 1\right)$$

$$+ c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n$$

$$- (c_2 + c_4 + c_5 + c_8).$$

  - T(n) puede expresarse como $an^2 + bn + c$ para *a*, *b*, y *c* constants que dependen de los costos c_i de las instrucciones; *T(n)* es una función cuadrática de *n*.

Análisis de Casos: INSERTION-SORT

- CASO PROMEDIO:
  - Si elegimos aleatoriamente n números y aplicamos INSERTION-SORT, cuánto toma determinar donde insertar $A[j]$ en $A[1 \ldots j-1]$.
  - En promedio, se verificará la mitad del subarreglo, tal que $t_j = j/2$.
  - Se calcula el resultado, se obtendrá una función cuadrática, al igual que en el PEOR CASO.
  - Para realizar la evaluación del CASO PROMEDIO, en algunos casos se necesita Análisis probabilístico, y Algoritmos Aleatorizados.

Tasa de Crecimiento (Θ)

- Considera sólo el término de mayor orden de la fórmula, sin incluir el coeficiente del mismo.
- Asume que los factores constantes y los de menor orden, son insignificantes para el caso de entradas grandes. Por ejemplo, para $an^2 + bn + c$, será n^2.
- En el caso del método INSERTION-SORT, este tiene un tiempo de ejecución del PEOR CASO de $\Theta(n^2)$.
- Permite determinar si un algoritmo es más eficiente que otro (comparar las tasas de crecimiento de sus *T(n)* del PEOR CASO).
- Un algoritmo con tasa de crecimiento $\Theta(n^2)$, será más eficiente que uno con, $\Theta(n^3)$.

3. Diseño de algoritmos

Introducción

- Existen diferentes técnicas para diseñar algoritmos.
- El método INSERTION-SORT aplica una estrategia *incremental*.
- *Estrategia divide y conquista*
  - Algoritmos de estructura recursiva
  - Tres pasos en cada nivel de recursión:
    - DIVIDE el problema en subproblemas
    - CONQUISTA los subproblemas resolviéndolos recursivamente. Si es lo suficientemente pequeño, lo resuelve directamente.
    - COMBINA los resultados de los subproblemas para obtener la solución al problema original.

Caso : Procedimiento MERGE SORT

- DIVIDE: divide la secuencia de n elementos a ser ordenada en dos subsecuencias de n/2 elementos cada una.
- CONQUISTA: ordena las dos subsecuencias recursivamente usando MERGE SORT.
- COMBINA: combina las dos subsecuencias ordenadas para obtener la respuesta ordenada.

- La recursión **empieza a retornar** cuando la secuencia a ser ordenada es de tamaño 1, caso en el que no realiza ningún ordenamiento.

Caso: Procedimiento MERGE SORT

- Operación clave en MERGE SORT es la mezcla de las secuencias ordenadas en el paso COMBINA:
  - Procedimiento auxiliar MERGE(A, p, q, r), donde A es un arreglo y p, q, y r son índices que enumeran elementos del arreglo tal que $p \leq q < r$.
  - Asume que los subarreglos $A[p \dots q]$ y $A[q + 1 \dots r]$ están ordenados; y los mezcla en un solo arreglo ordenado $A[p \dots r]$.

- MERGE toma un tiempo de $\Theta(n)$, donde $n = r - p + 1$ es el número de elementos que son ordenados.
- Se considera necesario mantener un **centinela** (carta) que se usa para simplificar el código, evitando verificar si la pila está vacía en cada paso.

Procedimiento MERGE

MERGE(A, p, q, r)

$n_1 \leftarrow q - p + 1$

$n_2 \leftarrow r - q$

create arrays $L[1 \cdot\cdot\, n_1 + 1]$ and $R[1 \cdot\cdot\, n_2 + 1]$

for $i \leftarrow 1$ **to** n_1

    do $L[i] \leftarrow A[p + i - 1]$

for $j \leftarrow 1$ **to** n_2

    do $R[j] \leftarrow A[q + j]$

$L[n_1 + 1] \leftarrow \infty$

$R[n_2 + 1] \leftarrow \infty$

…Continuación de MERGE

```
i ← 1
j ← 1
for k ← p to r
      do if L[i] ≤ R[j]
            then A[k] ← L[i]
                  i ← i + 1
            else A[k] ← R[j]
                  j ← j + 1
```

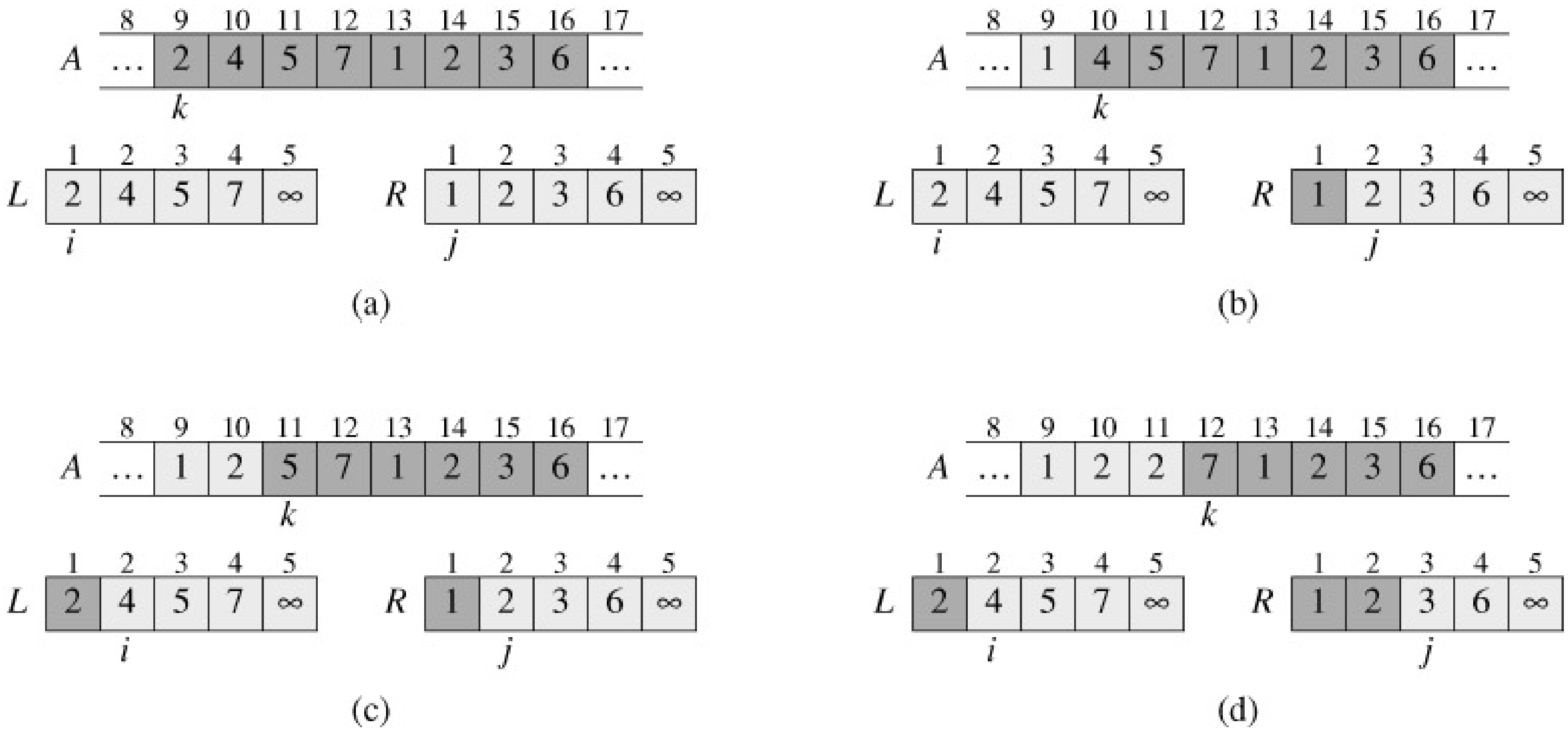
Correctitud de MERGE

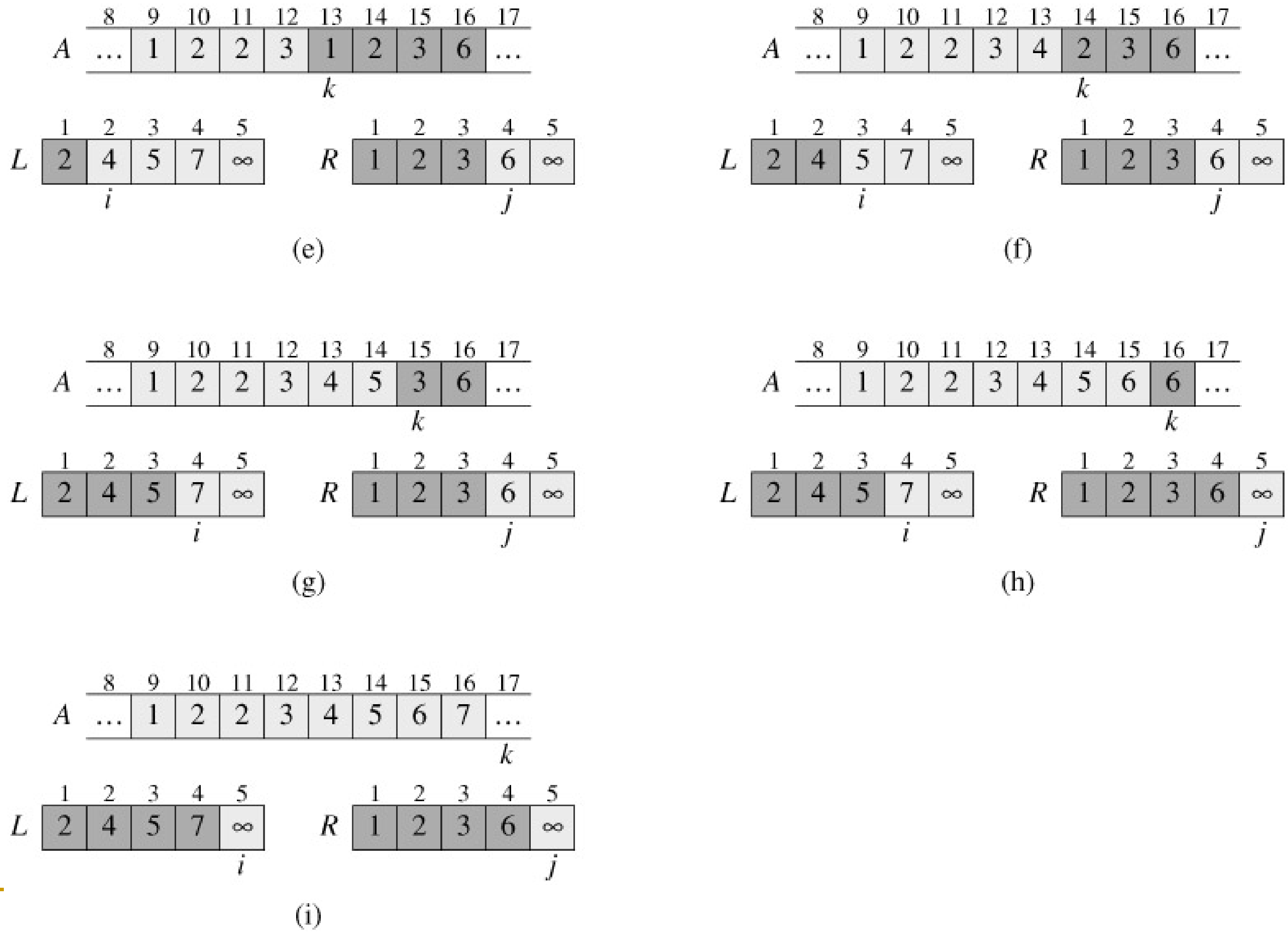
- **Inicialización**: Antes de la 1° iteración
  - k = p A[p… k - 1] está vacía.
  - i = j = 1, L[i] y R[j] elementos más pequeños no copiados en A
- **Mantenimiento**: Verificar cada iteración
  - L[i] ≤ R[j] (L[i], menor elemento aún no copiado en A).
  - A[p … k - 1] tiene k - p menores elementos
  - L[i] en A[k] A[p · · k] tendrá los k - p + 1 menores elementos. (línea 14)
  - Incremento de k e i restablecen el invariante del bucle.

Correctitud de MERGE

- **Terminación**: Al terminar
  - k = r + 1.
  - A[p… k - 1] = A[p… r], tiene los k - p = r - p + 1 menores elementos de $L[1 \cdot\cdot n_1 + 1]$ y $R[1 \cdot\cdot n_2 + 1]$, ordenados.
  - L y R contienen $n_1 + n_2 + 2 = r - p + 3$ elementos.
  - Todos menos los dos más grandes han sido copiados en A, y estos dos elementos son los centinelas. el invariante del bucle es verdadero

Ejecución MERGE





8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(e)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(f)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(g)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(h)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 7 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(i)

Procedimiento MERGE SORT

- El procedimiento MERGE se ejecuta en tiempo $\Theta(n)$, donde $n = r - p + 1$.
- MERGE es subrutina de MERGE-SORT(A, p, r), que ordena los elementos de $A[p \ldots r]$.
- Si $p \geq r$,
  - subarreglo tiene a lo más un elemento y está ordenado.
- Sino, se divide en el índice q, $A[p \ldots r]$ en dos subarreglos:
  - $A[p \ldots q]$, con $\lceil n/2 \rceil$ elementos, y
  - $A[q + 1 \ldots r]$, con $\lfloor n/2 \rfloor$ elementos.
- Para ordenar $A = \langle A[1], A[2], . . . , A[n] \rangle$, llamada será MERGE-SORT(A, 1, *length*[A]), donde *length*[A] $= n$.

Procedimiento MERGE SORT

MERGE-SORT(A, p, r)

if $p < r$

  then $q \leftarrow \lfloor (p + r)/2 \rfloor$

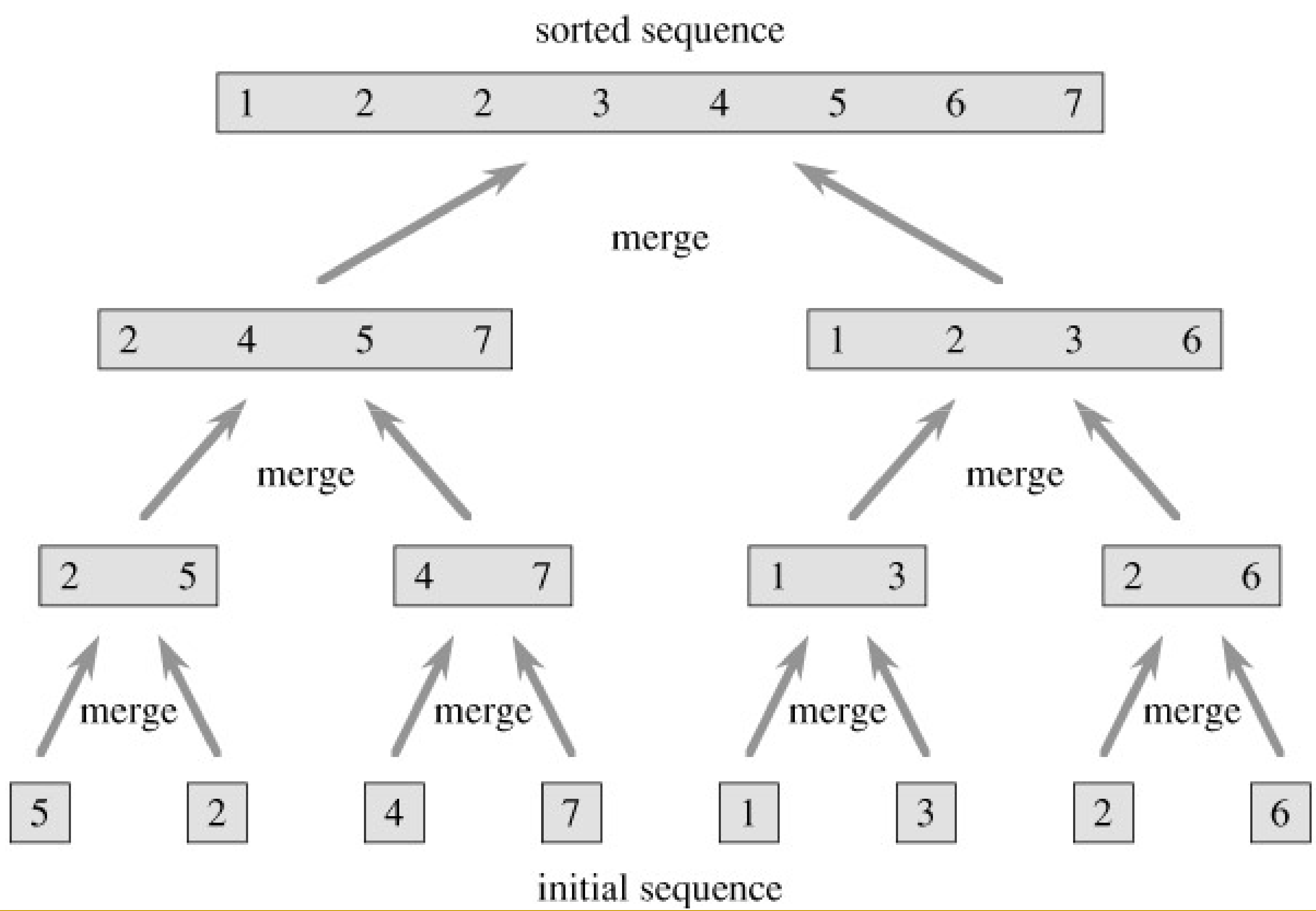
    MERGE-SORT(A, p, q)

    MERGE-SORT(A, $q + 1$, r)

    MERGE(A, p, q, r)

Mayor entero menor o igual a x

Ejecución MERGE SORT



Análisis de Divide y conquista

- Ecuación de recurrencia o recurrencia: describe el *T(n)* de algoritmo recursivo.
  - Describe *T(n)* en función del *T(n'), n'*< *n*
  - Se basa en los tres pasos del paradigma básico.
  - Si $n \leq c$ (constante), la solución puede ser descrita por un tiempo constante Θ(1).
  - Si se divide el problema en *a* sub-problemas, de tamaño 1/*b* del problema original, donde puede ser $a \neq b$, *D(n)*: tiempo de división, y *C(n)*: tiempo para combinar las soluciones

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{em caso contrário.} \end{cases}$$

Análisis: Caso MERGE SORT

- Asumir :
  - *n* potencia de 2: tamaño del problema original
  - División: dos subsecuencias de *n/2*
  - No afecta la tasa de crecimiento de la recurrencia.
- Se describirá la recurrencia de *T(n)* para el Peor Caso:
  - Ordenar un elemento toma un tiempo constante $\Theta(1)$.
  - Si n > 1, el tiempo de ejecución se describe:
    - Divide: calcula la posición media del arreglo, $D(n) = \Theta(1)$.
    - Conquista: resuelve recursivamente dos subproblemas de tamaño n/2, implica $2T(n/2)$
    - Combina: MERGE a un arreglo de n elementos toma un tiempo $\Theta(n)$, $C(n) = \Theta(n)$

Análisis: Caso MERGE SORT

- Agregando $D(n)$ y $C(n)$ al análisis de MERGE SORT, se agrega una función $\Theta(n)$ y otra $\Theta(1)$.
- La suma es una función lineal de n, $\Theta(n)$.
- Sumando el término $2T(n/2)$ de la fase de conquista, la recurrencia para el Peor Caso es:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ 2T(n/2) + \Theta(n) & \text{se } n > 1. \end{cases}$$

  - Equivalente a:

$$T(n) = \begin{cases} c & \text{se } n = 1, \\ 2T(n/2) + cn & \text{se } n > 1. \end{cases}$$

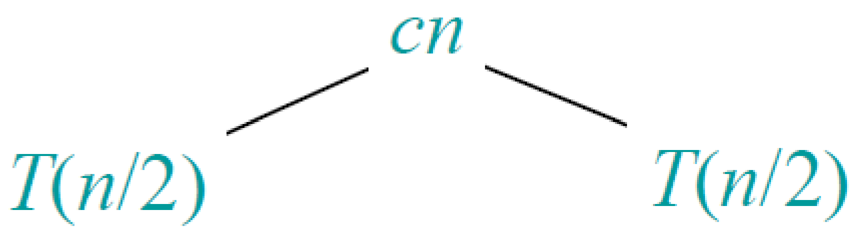
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0

$$T(n)$$

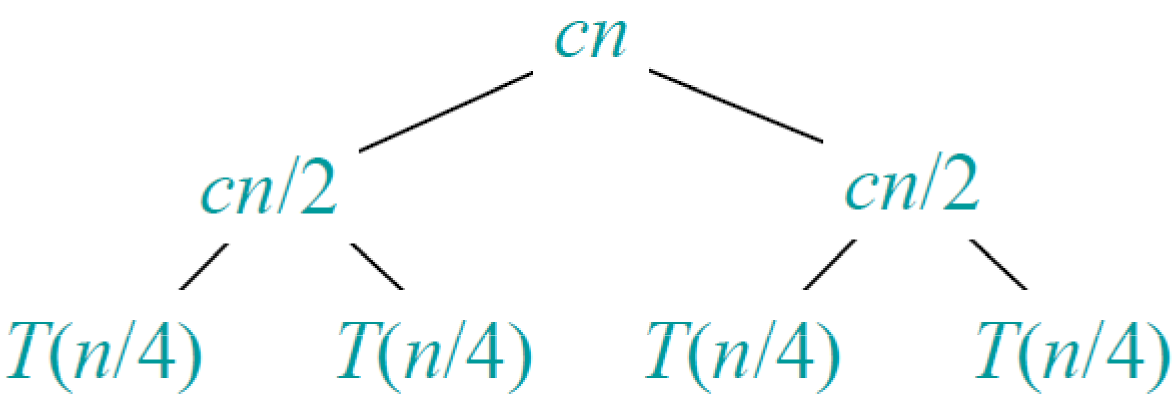
Análisis: Árbol de Recurrencia

- Resolver: $T(n) = T(n/2) + cn$, donde $cn > 0$



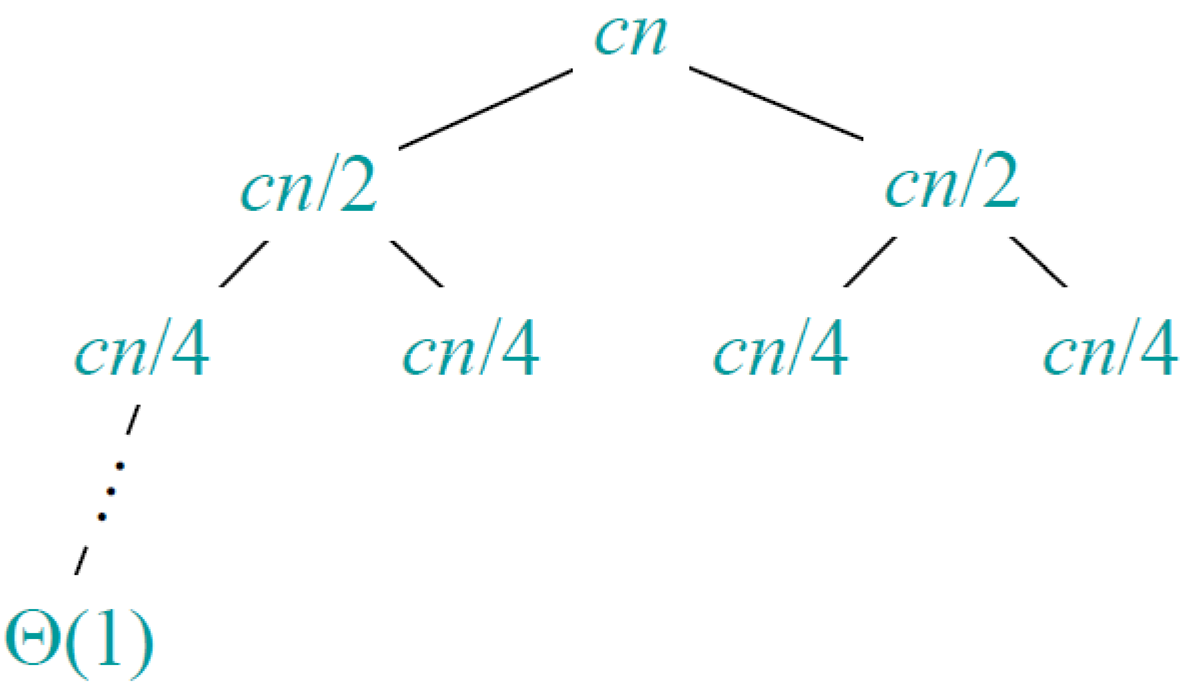
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



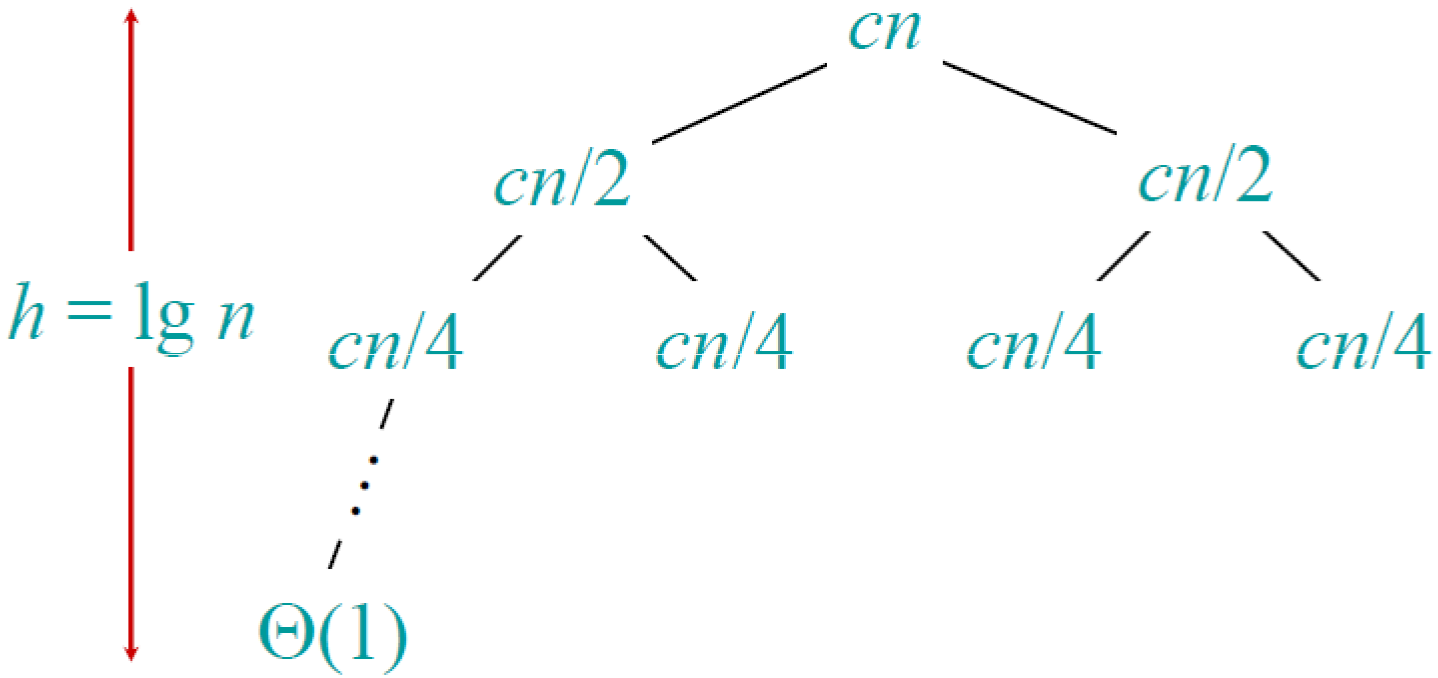
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



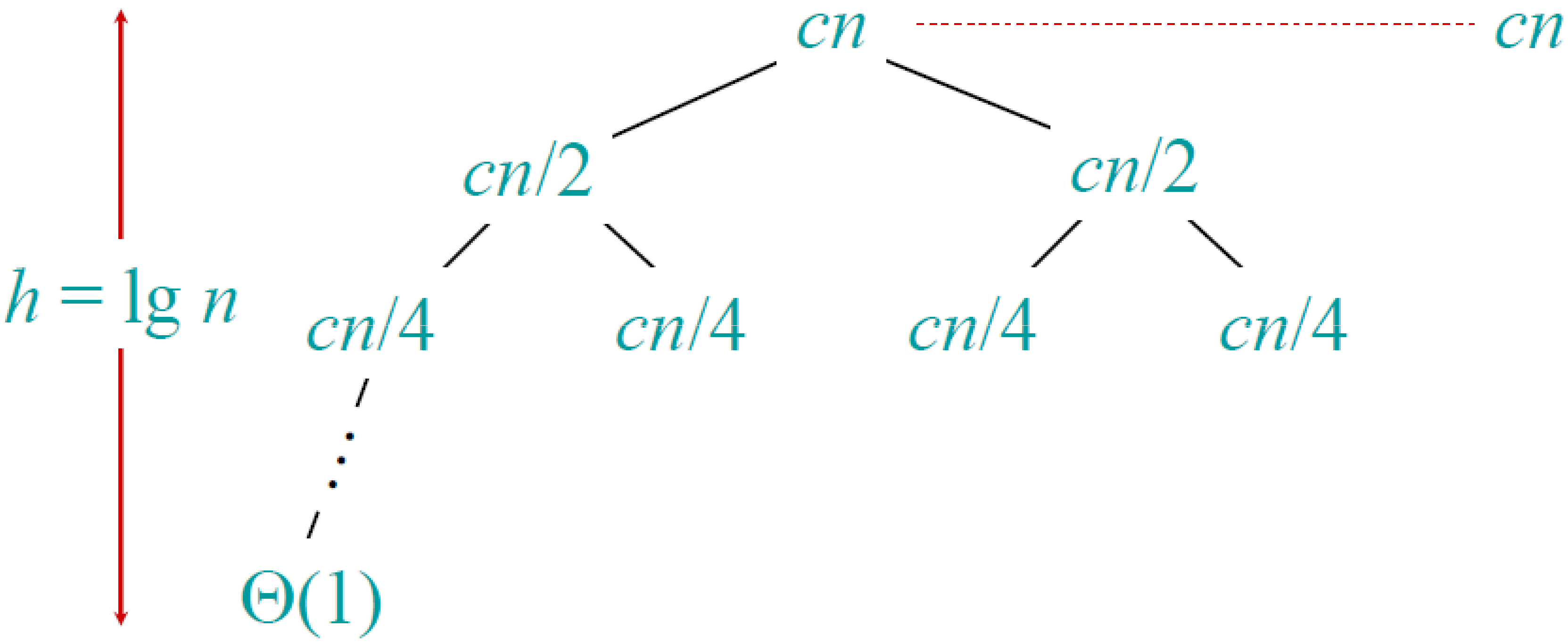
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



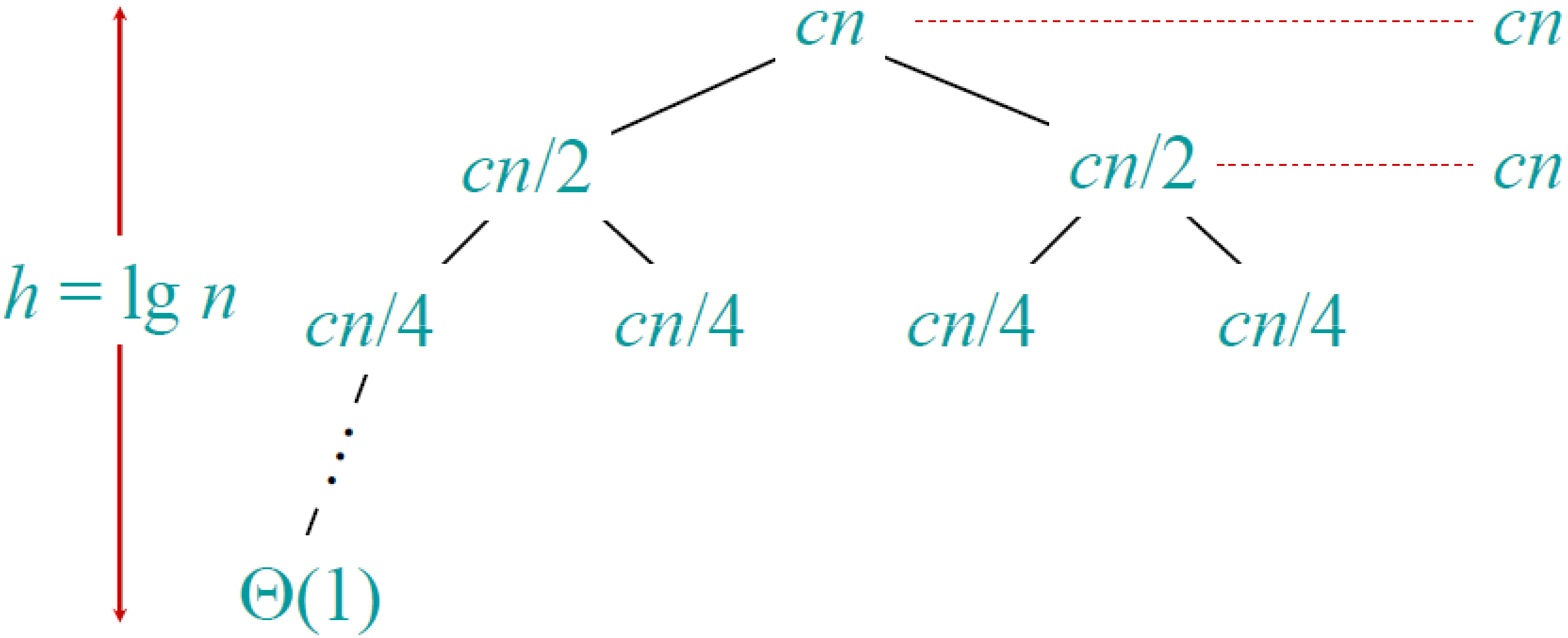
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



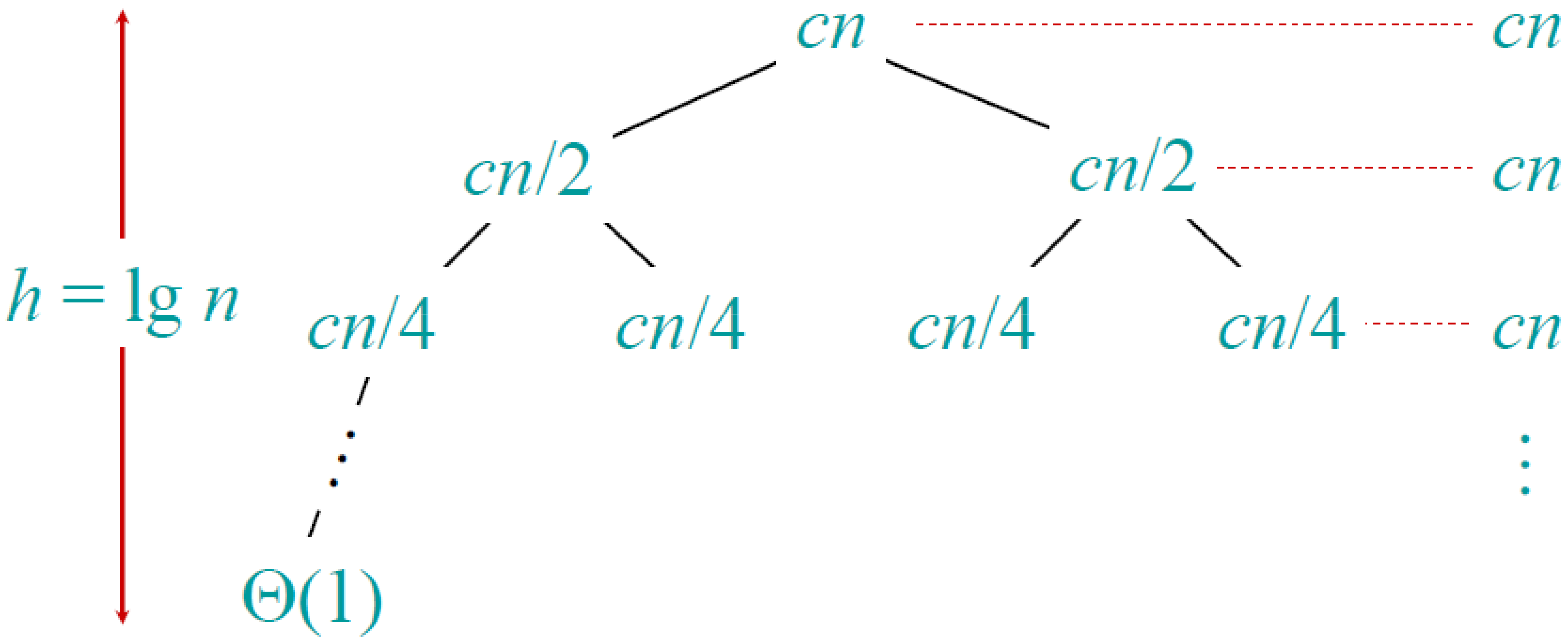
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



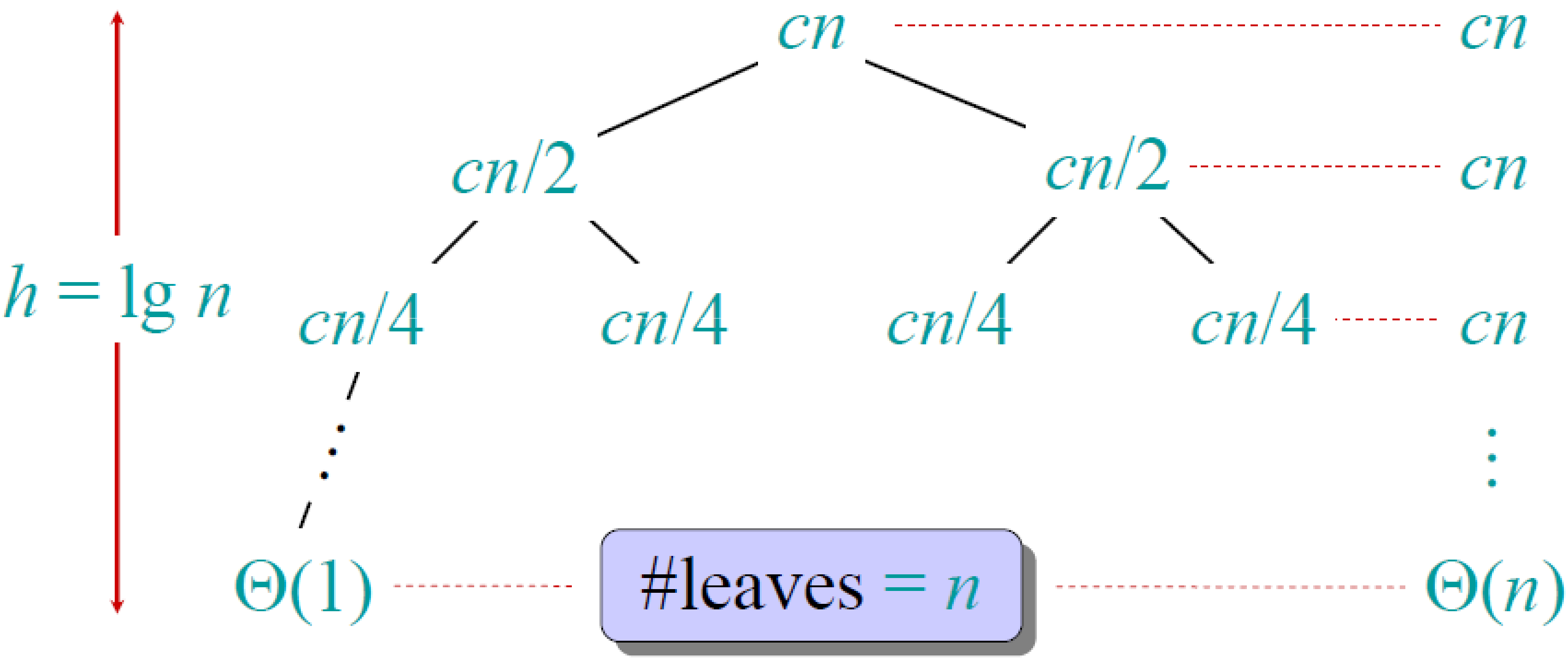
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



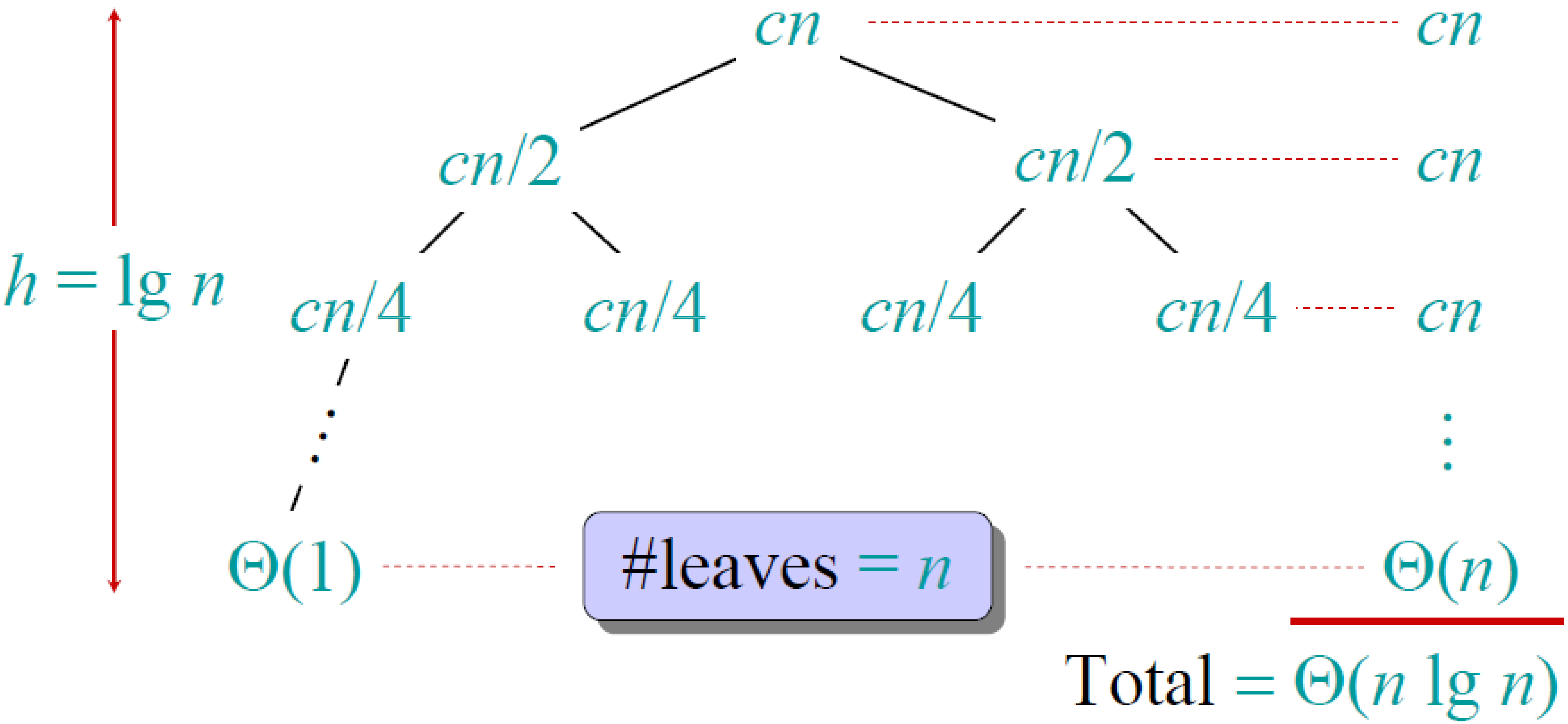
Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



Análisis: Árbol de Recurrencia

- Resolver: T(n) =T(n/2)+cn, donde cn>0



Análisis: Caso MERGE SORT

- Existen log n +1 niveles (altura = log n):
- Prueba usando inducción
  - Caso base:
    - n =1 : log 1 +1 = 0+1=1
  - Hipótesis Inductiva
    - Número de niveles del árbol para 2^i nodos : lg $2^i + 1 = i + 1$.
  - Tesis inductiva:
    - Un problema de tamaño 2^{i+1} *tiene un nivel mas que* 2^i
    - lg $2^{i+1} + 1 = i + 1$.

- Cálculo del costo total:
  - Suma costos de todos los niveles
  - $cn(\lg n + 1) = cn \lg n + cn$.
  - Ignorar los términos de menor orden se obtiene Θ(n lg n).

- $T(n) = \Theta(n \lg n)$
  - $\lg n$ representa $\log_2 n$

- MERGE SORT se desempeña mejor que INSERTION SORT, cuyo $T(n) = \Theta(n^2)$, para el Peor Caso.