Solución a Ecuaciones de Recurrencia

750094M - Fundamentos de Análisis y Diseño de Algoritmos

Antonio José Vélez Quintero
<antonio.velez@correounivalle.edu.co>
Docente Ocasional - Tecnología en Sistemas de Información
Universidad del Valle - Sede Palmira

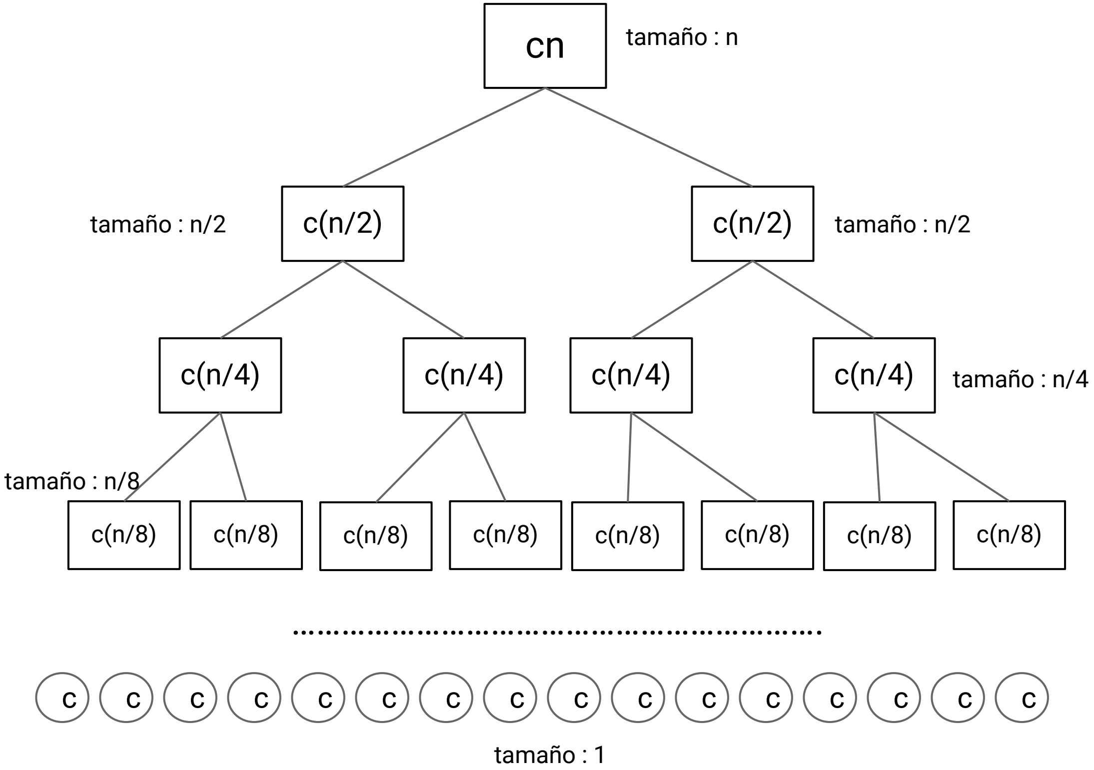
Árbol de Recurrencia

1. Construir el árbol de recurrencia

```
mergeSort (A , p , r )
    if ( p < r ) then
        q = piso ( (p + r) / 2 )
        mergeSort ( A , p , q )
        mergeSort ( A , q + 1, r )
        merge ( A , p , q , r )
```

n : la cantidad de elementos entre los índices p y r (n = r - p + 1)

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

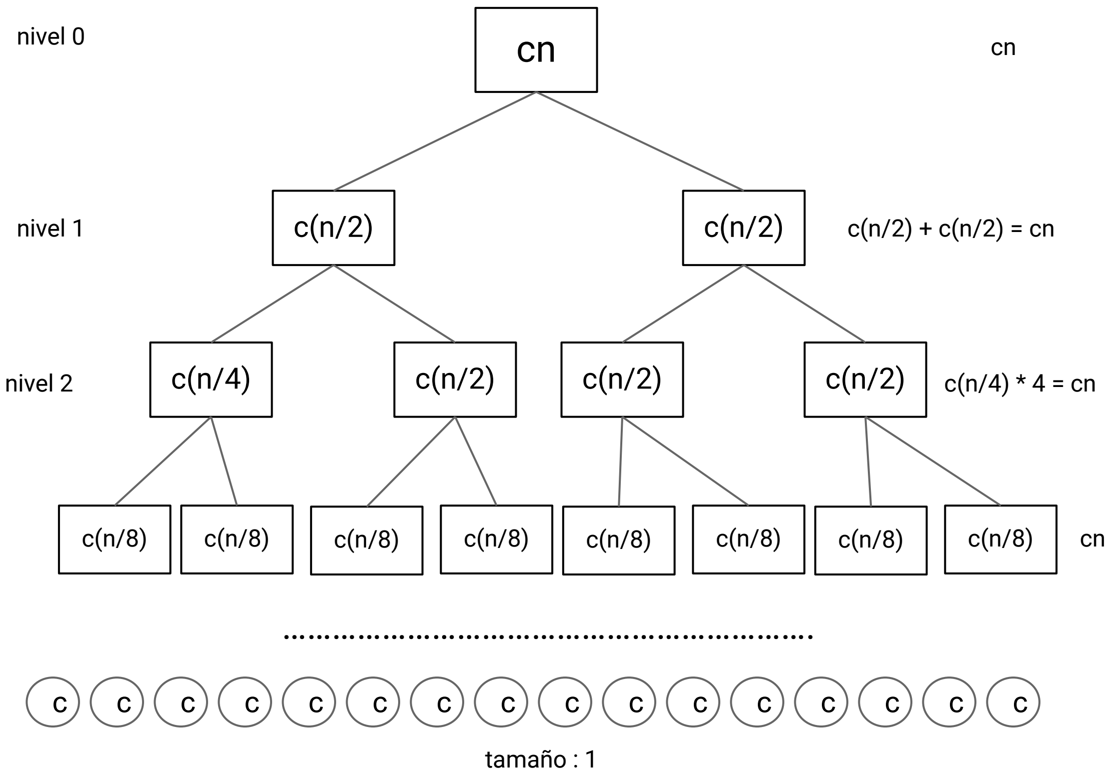


Árbol de Recurrencia

2. Hacer la suma de los niveles

```
mergeSort (A , p , r )
    if ( p < r ) then
        q = piso ( (p + r) / 2 )
        mergeSort ( A , p , q )
        mergeSort ( A , q + 1, r )
        merge ( A , p , q , r )
```

n : la cantidad de elementos entre los índices p y r (n = r - p + 1)

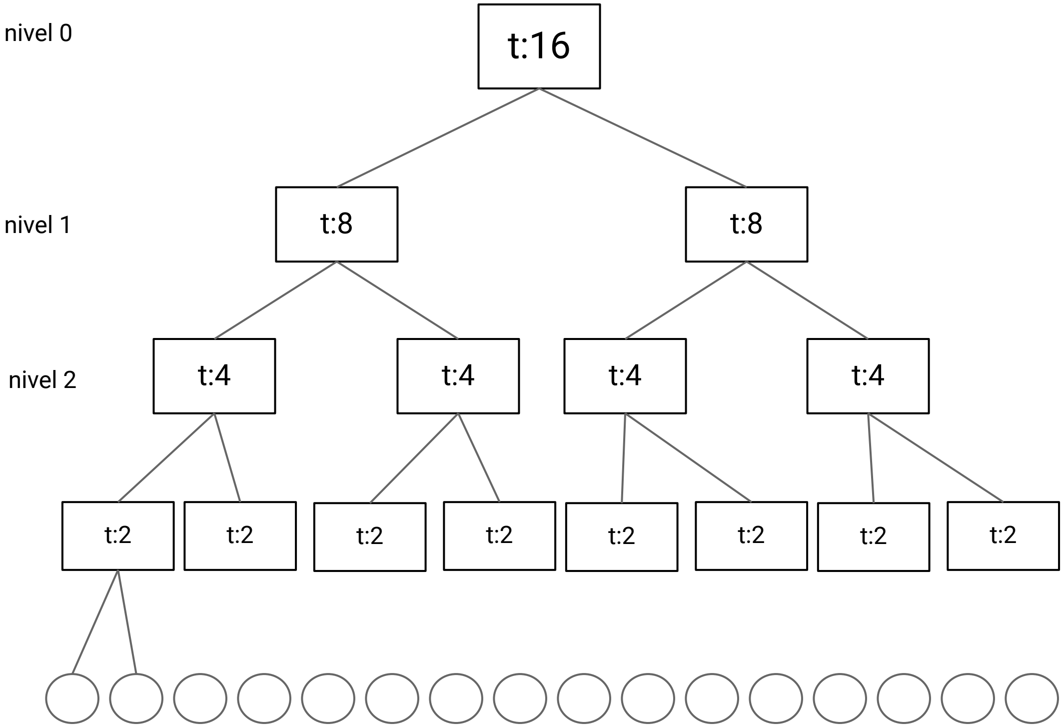


Árbol de Recurrencia

3. Determinar la altura del árbol

```
mergeSort (A , p , r )
    if ( p < r ) then
        q = piso ( (p + r) / 2 )
        mergeSort ( A , p , q )
        mergeSort ( A , q + 1, r )
        merge ( A , p , q , r )
```

n : la cantidad de elementos entre los índices p y r (n = r - p + 1)



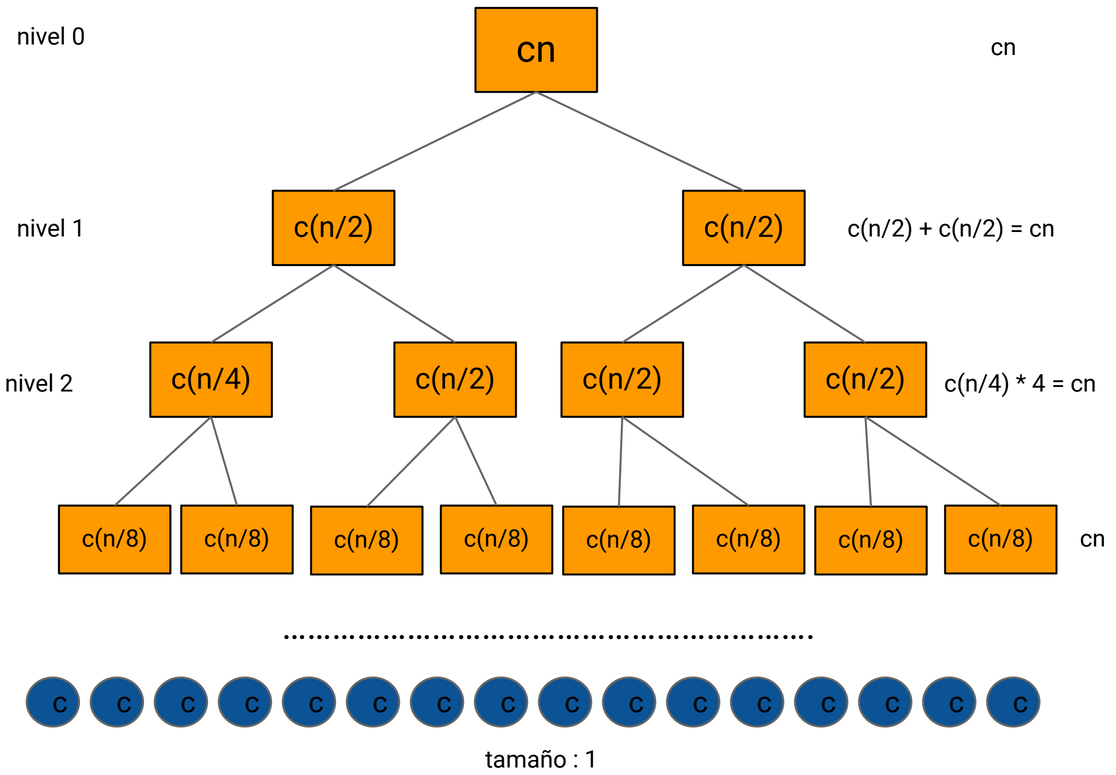
Cúal es índice del último nivel ?
ejemplo : 4
general : $\log_2(n)$

Cuantos nodos de tamaño 1 hay ?
ejemplo : 16
general : $\log_2 16$

Árbol de Recurrencia

4a. Determinar el costo (hacer la suma) de los niveles de llamado recursivo

4b. Determinar el costo (hacer la suma) de los nodos de tamaño 1



Árbol de Recurrencia

4a. Determinar el costo (hacer la suma) de los niveles de llamado recursivo (tamaño > 1)

4b. Determinar el costo (hacer la suma) de los nodos de tamaño 1

$$\sum_{i=0}^{(lg n)-1} cn$$

$n * c = cn$

Árbol de Recurrencia

5. Sumar y resolver

$$T(n) = \sum_{i=0}^{(lgn)-1} cn \quad + \quad cn$$

$T(n) = cnlgn + cn$

$cn * lgn + cn * 1$

$T(n) = \Theta(nlgn)$

Solución de Ecuaciones de Recurrencia

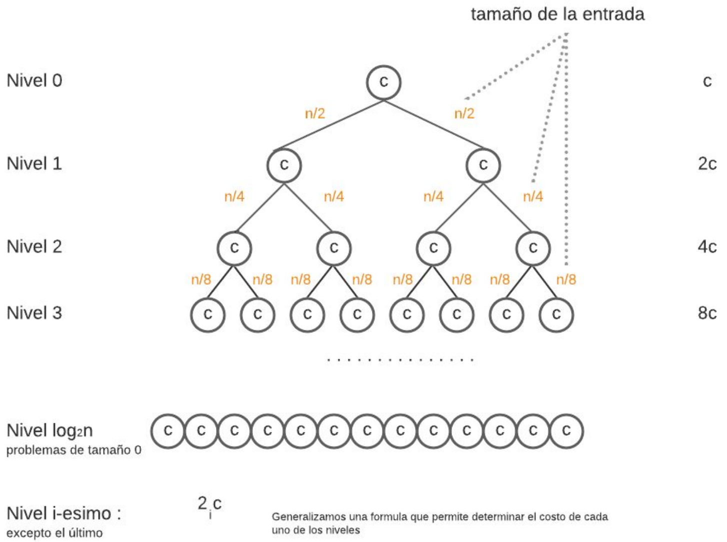
Dado un árbol binario balanceado, contar la cantidad de nodos del árbol

```
contarNodos ( arb )
    if arb is empty then
        return 0
    else
        return  contarNodos(arb.hijoIzq)
                + contarNodos(arb.hijoDer)
                + 1
```

n : la cantidad de nodos que tiene árbol

$$T(n) = \begin{cases} \Theta(1) & \text{para } n = 0 \\ 2T(\text{piso}(n/2)) + \Theta(1) \end{cases}$$

$\Theta(1) = c$



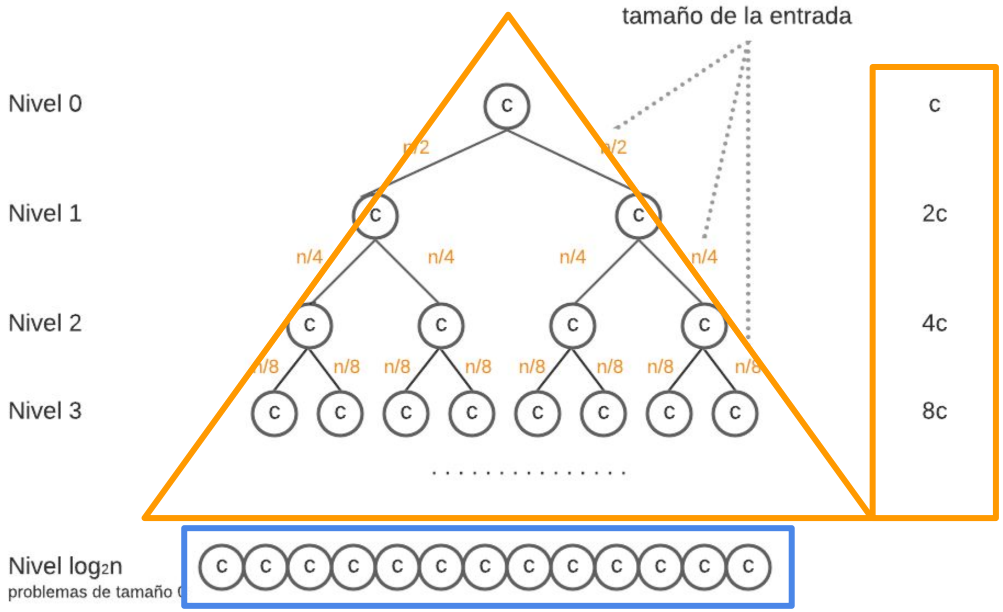
	General	**Problema**
Altura del árbol (índice del último)	$\log_b n$	$\log_2 n$
Cantidad de nodos del último nivel	n ^ $\log_b a$	n

Solución de Ecuaciones de Recurrencia

Dado un árbol binario balanceado, contar la cantidad de nodos del árbol

```
contarNodos ( arb )
    if arb is empty then
        return 0
    else
        return  contarNodos(arb.hijoIzq)
              + contarNodos(arb.hijoDer)
              + 1
```

n : la cantidad de nodos que tiene árbol



Nivel i-esimo : $2^i c$

excepto el último

Generalizamos una formula que permite determinar el costo de cada uno de los niveles

$$T(n) = \boxed{\sum_{i=0}^{log2n-1} 2^i c} + \boxed{cn}$$

Solución de Ecuaciones de Recurrencia

Dado un árbol binario balanceado, contar la cantidad de nodos del árbol

```
contarNodos ( arb )
    if arb is empty then
       return 0
    else
        return  contarNodos(arb.hijoIzq)
                + contarNodos(arb.hijoDer)
                + 1
```

n : la cantidad de nodos que tiene árbol

$$T(n) = \sum_{i=0}^{log2n-1} 2^i c + cn$$

$$T(n) = c(2^{log2n-1} - 1) + cn$$

$$T(n) = c(2^{log2n-1} - 1) + cn$$

$$T(n) = c2^{log2n} * 1/2 - c + cn$$

$$T(n) = cn^{log2(2)} * 1/2 - c + cn$$

$$T(n) = cn/2 - c + cn$$

$$T(n) = cn(1/2 + 1) - c = 3/2cn - c$$

$$T(n) = \Theta(n)$$

El costo computacional del algoritmo **contarNodos** es Θ(n)

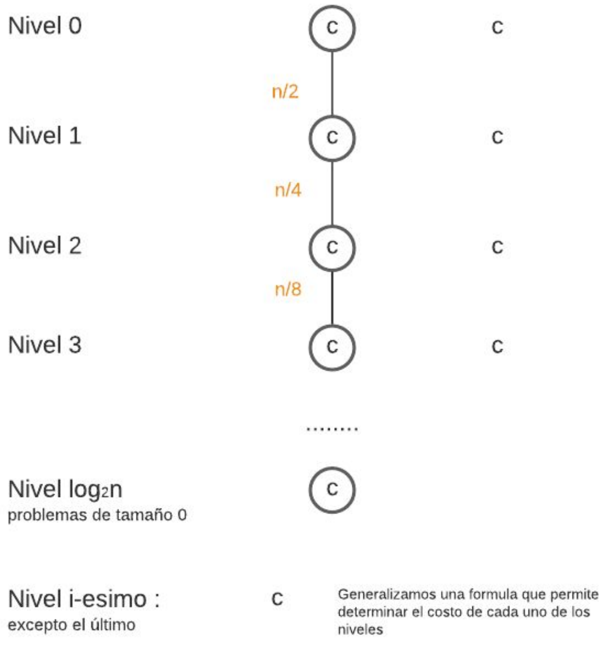
Solución de Ecuaciones de Recurrencia

Dado un árbol binario de búsqueda (ABB), determinar si un elemento se encuentra contenido en el ABB.

```
buscarElemento ( abb , val )
    if arb is empty then
       return false
    if arb.info == val then
        return return true
    if val < arb.info then
        buscarElemento (abb.hijoIzq , val)
    else
        buscarElemento (abb.hijoDer, val)
```

n : la cantidad de nodos que tiene árbol

$$T(n) \begin{cases} \Theta(1) & \text{para } n = 0 \\ T(\text{piso}(n/2)) + \Theta(1) & \end{cases}$$



$$T(n) = \sum_{i=0}^{\log 2n-1} c \; + c$$

$$T(n) = c \sum_{i=0}^{\log 2n-1} 1 \; + c$$

$$T(n) = c(\log_2 n - 1) + c$$

$$T(n) = c\log_2 n - c + c$$

$$T(n) = \Theta(\log_2 n)$$

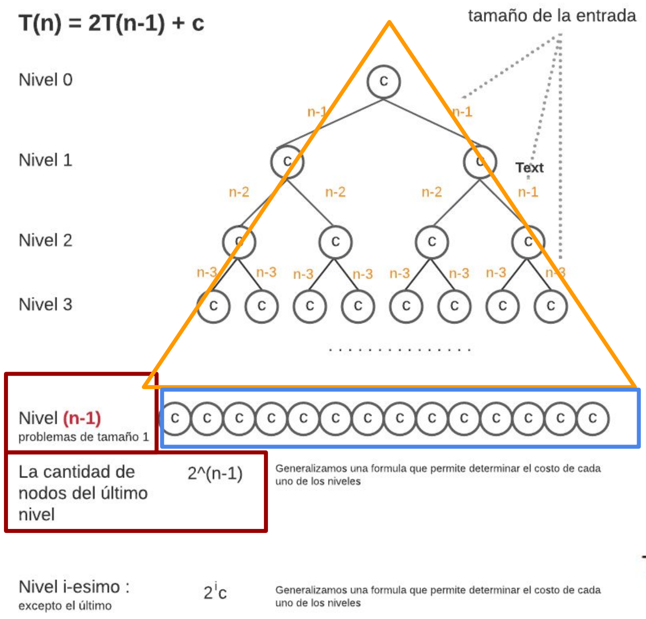
Solución de Ecuaciones de Recurrencia

Dada una lista que contiene número, devolver el número mayor
Aclaración: la lista no será una lista vacía

```
mayor ( lst )
    if lst.next is empty then
        return lst.info
    else
        if lst.info > mayor ( lst.next ) then
            return lst.info
        else
            return mayor ( lst.next )
```

n : cantidad de elementos de la lista

$$T(n) \begin{cases} \Theta(1) & \text{para } n = 0 \quad // \ c1 + c2 \\ 2T(n-1) + \Theta(1) \end{cases}$$



$$T(n) = \boxed{\sum_{i=0}^{n-2} 2^i c} + \boxed{2^{(n-1)} c}$$

Solución de Ecuaciones de Recurrencia

Dada una lista que contiene número, devolver el número mayor
Aclaración: la lista no será una lista vacía

```
mayor ( lst )
    if lst.next is empty then
       return lst.info
    else
       if lst.info > mayor ( lst.next ) then
          return lst.info
       else
          return mayor ( lst.next )
```

n : cantidad de elementos de la lista

$$T(n) \begin{cases} \Theta(1) & \text{para } n = 0 \quad // \text{ c1 + c2} \\ 2T(n-1) + \Theta(1) & \end{cases}$$

$$T(n) = \sum_{i=0}^{n-2} 2^i c + 2^{(n-1)} c$$

$$T(n) = c \sum_{i=0}^{n-2} 2^i + 2^{(n-1)} c$$

$$T(n) = c \sum_{i=0}^{n-2} 2^i + 2^{(n-1)} c \qquad \sum_{k=0}^{n} 2^k = 2^{n+1} - 1$$

$$T(n) = c(2^{(n-2+1)} - 1) + 2^{(n-1)} c$$

$$T(n) = 2^{(n-2+1)} c - c + 2^{(n-1)} c$$

$$T(n) = 2^{(n-1)} c - c + 2^{(n-1)} c$$

$$T(n) = 2^{(n-1)} c - c + 2^{(n-1)} c$$

$$T(n) = (2^{(n-1)} c) \, 2 - c \qquad a^m * a^n = a^{m+n}$$

$$T(n) = (2^{n-1+1} c) - c = T(n) = (2^n c) - c$$

$$T(n) = \Theta(2^n)$$

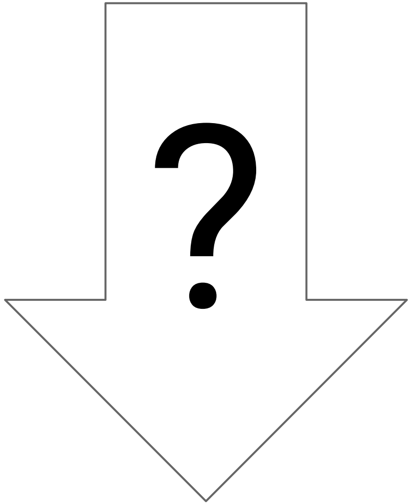
Mejorando el Algoritmo

Dada una lista que contiene número, devolver el número mayor
Aclaración: la lista no será una lista vacía

```
mayor ( lst )
    if lst.next is empty then
        return lst.info
    else
        m = mayor ( lst.next )
        if lst.info > m then
            return lst.info
        else
            return m
```

n : cantidad de elementos de la lista

$$T(n) \begin{cases} \Theta(1) \quad \text{para } n = 0 \quad \text{// c1 + c2} \\ T(n - 1) + \Theta(1) \end{cases}$$



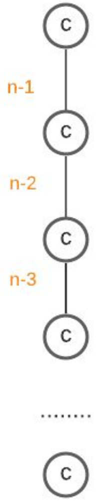
Solución de Ecuaciones de Recurrencia

Sumar los valores entre 1 y n

```
sumar ( n )
    if n == 0
        return 0
    else
        return sumar ( n - 1 ) + n
```

n : cantidad de números a sumar

$$T(n) = \begin{cases} \Theta(1) & \text{para } n = 0 \quad // \; c1 + c2 \\ T(n-1) + \Theta(1) & \end{cases}$$



Nivel 0		c
Nivel 1		c
Nivel 2		c
Nivel 3		c
Nivel **(n-1)** problemas de tamaño 0		
Nivel i-esimo : excepto el último	c	Generalizamos una formula que permite determinar el costo de cada uno de los niveles

$$T(n) = \sum_{i=0}^{n-2} c + 1 * c$$

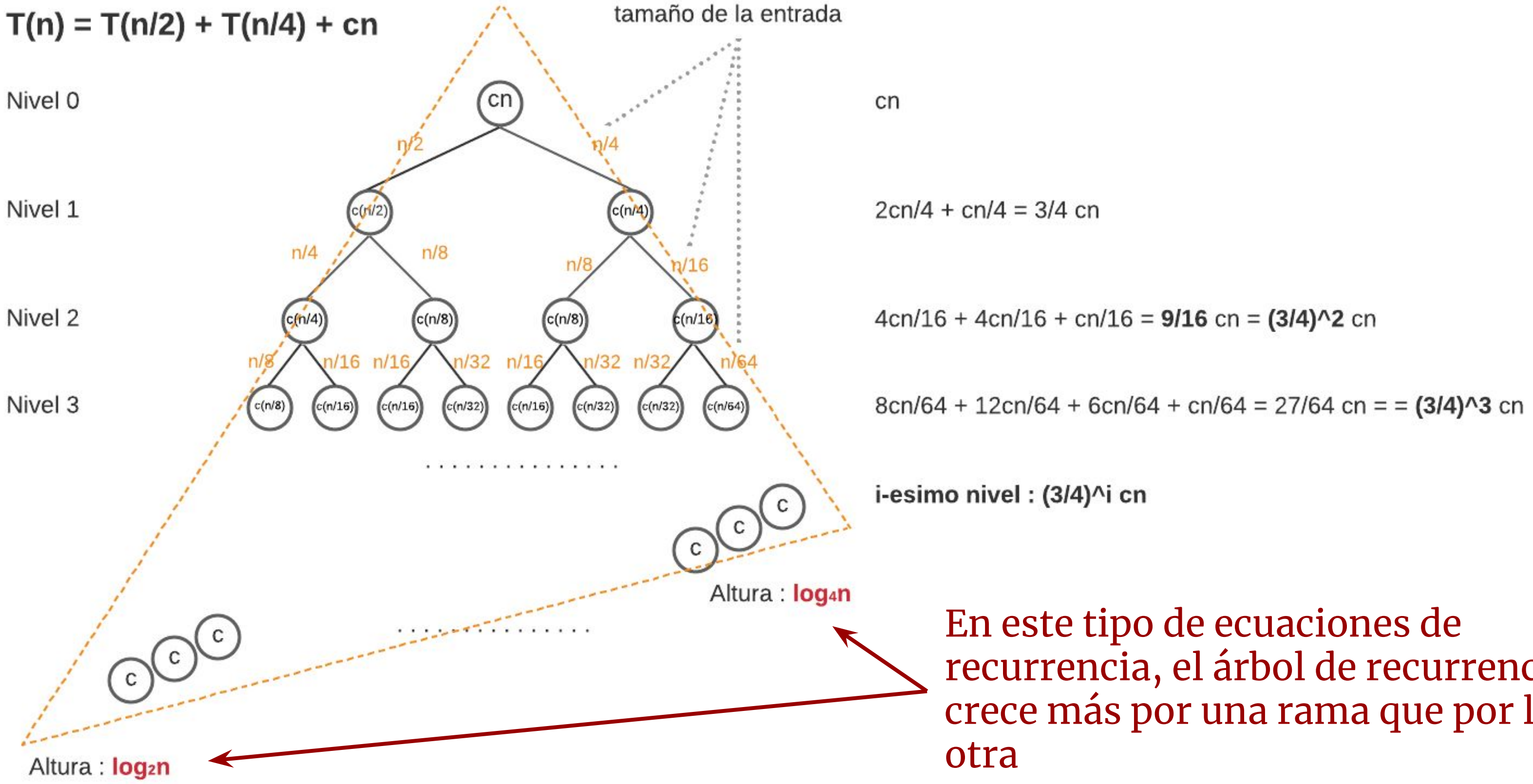
$$T(n) = c \sum_{i=0}^{n-2} 1 + 1 * c$$

$$T(n) = c(n - 2 + 1) + c$$

$$T(n) = cn - c + c$$

$$T(n) = cn$$

$$T(n) = \Theta(n)$$

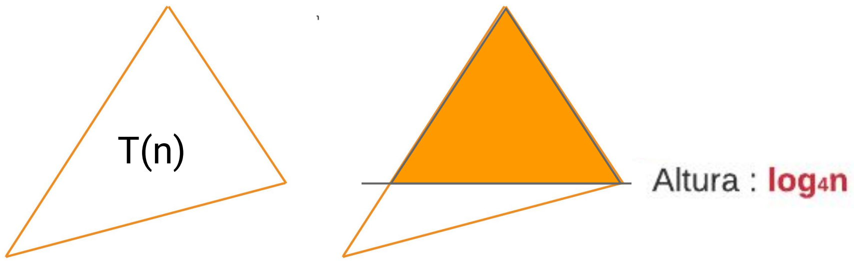


En este tipo de ecuaciones de recurrencia, el árbol de recurrencia crece más por una rama que por la otra

Usar el método de árbol de recurrencia para resolver la siguiente ecuación de recurrencia:

T(n) = T(n/2) + T(n/4) + cn

Cortar el árbol de recurrencia



$$T(n) >= cn \sum_{i=0}^{log4(n)-1} (3/4)^i + 2^{log4(n)}c$$

$$T(n) >= cn \sum_{i=0}^{log4(n)-1} (3/4)^i + 2^{log4(n)}c$$

$$\boxed{\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad para|\,x\,|<1}$$

$$T(n) >= cn \sum_{i=0}^{log4(n)-1} (3/4)^i + 2^{log4(n)}c \quad <= cn \sum_{i=0}^{INF} (3/4)^i + 2^{log4(n)}c$$

$$\boxed{a^{\log_b n} = n^{\log_b a}}$$

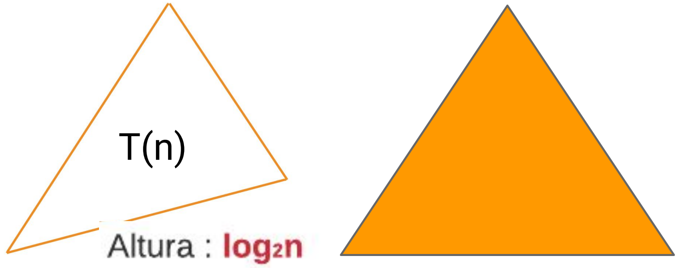
$$T(n) >= cn \sum_{i=0}^{INF} (3/4)^i + 2^{log4(n)}c$$

$$T(n) >= cn(1/(1-3/4)) + n^{log4(2)}c$$

$$T(n) >= 4cn + n^{1/2}c$$

$$T(n) \in \Omega(n)$$

Completar el árbol de recurrencia



$$T(n) <= cn \sum_{i=0}^{log2(n)-1} (3/4)^i + 2^{log2(n)}c$$

$$\boxed{\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad para|\,x\,|<1}$$

$$T(n) <= cn \sum_{i=0}^{log2(n)-1} (3/4)^i + 2^{log2(n)}c$$

$$T(n) <= cn \sum_{i=0}^{log2(n)-1} (3/4)^i + 2^{log2(n)}c \quad <= cn \sum_{i=0}^{INF} (3/4)^i + 2^{log2(n)}c$$

$$\boxed{a^{\log_b n} = n^{\log_b a}}$$

$$T(n) <= cn(1/(1-1/2)) + n^{log2(2)}c$$

$$T(n) <= 2cn + nc <= 3cn$$

$$T(n) \in O(n)$$

Usar el método de árbol de recurrencia para resolver la siguiente ecuación de recurrencia:

T(n) = T(n/2) + T(n/4) + cn