Capítulo 3 Complejidad de algoritmos recursivos

3.1 Definición de función de recurrencia

Una recurrencia es una ecuación o una desigualdad que describe una función en términos de su propio valor sobre entradas más pequeñas. Por ejemplo, el cálculo del `Factorial` de un número se puede describir con la ecuación de recurrencia $F(n)$.

$$F(n) = \begin{cases} nF(n-1) & if \;\; n > 0 \\ 1 & if \;\; n = 0 \end{cases}$$

La solución del `Factorial` de 4 se puede obtener siguiendo las reglas de la ecuación de recurrencia.

$$F(4) = 4 \cdot F(3)$$
$$F(3) = 3 \cdot F(2)$$
$$F(2) = 2 \cdot F(1)$$
$$F(1) = 1 \cdot F(0)$$
$$F(0) = 1$$
$$F(4) = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 24$$

Una recurrencia es lineal, si cada llamada recursiva genera cuando mucho otra llamada recursiva, en caso contrario es no-lineal.

3.2 Deducción de recurrencias a partir de algoritmos

Los recursos empleados por algoritmos recursivos (por ejemplo, el número de comparaciones de clave y el número de multiplicaciones) se pueden describir con la recurrencia $T(n)$, la cual tiene una estructura inductiva: a) complejidad de los casos base y b) complejidad de los casos progresivos. Los casos base detienen la recursividad, mientras los casos progresivos la reactivan.

$$T(n) = \begin{cases} \text{complejidad de los casos base} & \text{si } n \text{ corresponde a los casos base} \\ \text{complejidad de los casos progresivos} & \text{si } n \text{ corresponde a los casos progresivos} \end{cases}$$

Siguiendo con el ejemplo del `Factorial`, ahora se analiza el desempeño del algoritmo recursivo correspondiente.

```
Algoritmo Factorial (n)
Entradas: un número entero n
Salidas: un número entero correspondiente al factorial de n
1  si n = 0 entonces                    //caso base
2    Factorial  ← 1
3  sino                                 // caso progresivo
4    Factorial  ←  n*Factorial(n-1)
```

Si la operación básica es el número de multiplicaciones, entonces la recurrencia que mide el desempeño de algoritmo `Factorial` se obtiene identificando sus casos y escribiendo las reglas que determinan cada caso y las funciones de desempeño correspondientes. En el caso base (n=0) no se realiza ninguna multiplicación y en el caso progresivo (n>0) se realiza una multiplicación más las multiplicaciones que el algoritmo ejecuta al ser llamado recursivamente con un valor de n-1, este último calculo se denota con $T(n\text{-}1)$.

$$T(n)=\begin{cases}0 & \text{si } n=0 \text{ (caso base)}\\ 1+T(n-1) & \text{si } n>0 \text{ (caso progresivo)}\end{cases}$$

Antes se expandió la recurrencia $F(n)$ para determinar el factorial de 4, ahora se expande la recurrencia $T(n)$ para calcular el número de multiplicaciones.

$T(4) = 1 + T(3)=4$
$T(3) = 1 + T(2)=3$
$T(2) = 1 + T(1)=2$
$T(1) = 1 + T(0)=1$
$T(0) = 0$

$T(4) = 1 + 1 +1 +1 = 4$

Se puede observar que para calcular el factorial de 4, el algoritmo realiza cuatro multiplicaciones: `Factorial(4)=4·3·2·1·1`. Este número de multiplicaciones coincide con el obtenido por la recurrencia $T(4)$=4.

3.2.1 Recurrencia de peor caso del algoritmo de búsqueda secuencial recursivo

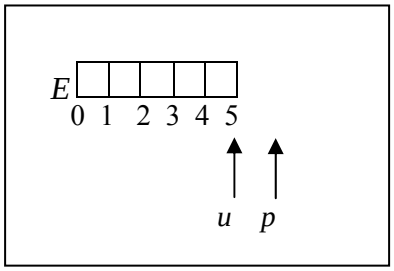
Una versión ineficiente del algoritmo de búsqueda secuencial es su implementación recursiva. A pesar de sus limitaciones, esta versión es sencilla y simplifica la introducción de los casos de un problema al análisis de algoritmos recursivos.

```
Algoritmo busquedaSecRec(E,p,u,k)
Entradas: Arreglo E, e índices p y u que delimitan el subarreglo a
          ordenar contenido en E[p,...,u].
Salidas: E[p,...,u] en orden ascendente.
1  si p>u entonces              //caso base de fracaso
2     respuesta = -1
3  sino si (E[p]=k) entonces    //caso base de éxito
4          respuesta ← p
5       sino
6          respuesta ← busquedaSecRec(E,p+1,u,k)//caso progresivo
7  devolver respuesta
```

El peor desempeño del algoritmo `busquedaSecRec` sucede cuando la búsqueda fracasa. Este comportamiento se puede identificar fácilmente en el algoritmo. Primero la recursividad se activa varias veces porque la llave k no ha sido encontrada (caso progresivo). Finalmente el algoritmo se detiene cuando no existen más datos para analizar (caso base de fracaso). A continuación se detalla el análisis del peor caso.

a) Caso base de fracaso: $p>u$

En el paso 1, cuando todos los elementos de E ya han sido analizados, el índice p rebasa el valor del índice u.



En estas condiciones el algoritmo se detiene porque no tuvo éxito, así que el número de comparaciones es cero porque no se tienen elementos en el segmento de búsqueda ya que $p>u$. Es importante notar que la condición **si $p>u$** equivale a la condición **si $n=0$**.

$$W(n)_{\text{paso 1}} = 0, \quad \text{si } n=0$$

b) Caso progresivo (invocación de recursividad): $p\leq u$ y $E[p]\neq k$

Después que en el paso 1 se encontró que $p\leq u$, en el paso 3 se realiza una comparación no exitosa ya que $E[p]\neq k$. En este paso el subarreglo sí tiene elementos ya que $p\leq u$. De igual manera la condición **si $p\leq u$** equivale a la condición **si $n>0$**.

$$W(n)_{\text{paso 3}} = 1, \quad n>0$$

En el paso 6 se ejecuta una invocación recursiva con un subarreglo disminuido en una unidad; en este paso el subarreglo también tiene elementos.

$$W(n)_{\text{paso 6}} = W(n\text{-}1), \quad n>0$$

c) Recurrencia del peor caso

Considerando todos los casos involucrados (caso base de fracaso y caso progresivo), la recurrencia del peor caso queda como sigue:

$$W(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 + W(n-1) & \text{si } n > 0 \end{cases}$$

Intuitivamente se puede observar que en cada llamada recursiva se realiza una comparación. Dado que se realizan n invocaciones al algoritmo, la solución de la recurrencia es n.

$$W(n) = 1+1+1+1+\ldots+1 = n = \Theta(n)$$

3.2.2 Recurrencia de peor caso del algoritmo de búsqueda binaria

Un algoritmo muy popular es el de búsqueda binaria. La aplicación de este algoritmo requiere un ordenamiento previo de los datos.

```
Algoritmo búsquedaBinaria (E,p,u,k)
Entradas: Arreglo E, e índices p y u que delimitan el subarreglo a
          ordenar contenido en E[p,...,u].
Salidas: E[p,...,u] en orden ascendente.

1  si (p>u)entonces                    //caso base de fracaso
2    devolver -1
3  medio ← ⌊(p + u)/2⌋
4  si k = E(medio) entonces            //caso base de éxito
5    índice ← medio
6  sino si (k < E[medio]) entonces
7         índice ← búsquedaBinaria(E,p,medio-1,k) //caso progresivo
8       sino
9         índice ← búsquedaBinaria(E,medio+1,u,k) //caso progresivo
10 devolver índice
```

El peor desempeño del algoritmo `búsquedaBinaria` sucede cuando la búsqueda fracasa. Primero la recursividad se activa varias veces porque la llave k no ha sido encontrada en la posición media (caso recursivo). Finalmente el algoritmo se detiene cuando no existen más datos para analizar (caso base de fracaso).

a) Caso base de fracaso : $p > u$

Cuando todos los elementos de E ya han sido analizados, el valor de p rebasa al de u; así que el número de elementos del segmento de búsquedas es cero.

$$W(n)_{\text{paso 1}} = 0, \quad \text{si } n=0$$

b) Caso progresivo (invocación de recursividad): $p \leq u$ y $E[medio] \neq k$

En el paso 5 del algoritmo se realiza una comparación no exitosa.

$$W(n)_{\text{paso 5}} = 1, \quad \text{si } n>0$$

En el paso 7 se realiza una comparación para ubicar la siguiente búsqueda.

$$W(n)_{\text{paso 7}} = 1, \quad \text{si } n>0$$

En las líneas 8 y 9 las invocaciones recursivas se realizan con subarreglos de diferentes tamaños. Cuando n es par: el segmento izquierdo es $n/2$ y el derecho es $(n/2)$-1. Cuando n es impar: los segmentos izquierdo y derecho son $(n\text{-}1)/2$. El mayor de cada tipo de tamaño es $n/2$ y $(n\text{-}1)/2$. Ambos tamaños máximos quedan expresados con $\lfloor n/2 \rfloor$.

$$W(n)_{\text{paso 8 o paso 9}} = W(\lfloor n/2 \rfloor), \quad \text{si } n>0$$

c) Recurrencia del peor caso

La siguiente recurrencia expresa el número de comparaciones de clave en el peor caso:

$$W(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2 + W\left(\left\lfloor \frac{n}{2} \right\rfloor\right) & \text{si } n > 0 \end{cases}$$

3.3 Ecuaciones de recurrencia comunes

Muchos algoritmos clásicos han sido diseñados en base a dos técnicas que permiten plantear la solución de un problema en términos de varios subproblemas de menor dimensión.

3.3.1 Algoritmos basados en divide-y-vencerás

Usando la técnica de divide-y-vencerás, si un problema es demasiado grande para resolverlo de una vez, se descompone en varias partes más fáciles de resolver. Con más formalidad, dado un problema a resolver planteado en términos de una entrada de tamaño n, la técnica de divide la entrada en b subproblemas, $1<b< n$. Estos subproblemas se resuelven independientemente y después se combinan sus soluciones parciales para obtener la solución del problema original.

En la función de recurrencia para divide-y-vencerás, n es el tamaño del problema principal, b son los subproblemas ($1<b< n$), cada uno de tamaño n/c ($1<c< n$). Para *dividir* el problema en subproblemas y *combinar* las soluciones de los subproblemas existe cierto costo no recursivo que se expresan en $f(n)$.

$$T(n) = b\ T(\ n/c\ ) + f(n)$$

Número de subproblemas
(factor de ramificación)

Costo de dividir en subproblemas y combinar las soluciones
Costo de resolver un subproblema obtenido por división

El algoritmo de ordenamiento `mergeSort` es un ejemplo de la aplicación de la técnica de divide-y-venceras.

```
Algoritmo mergeSort (E, p, u)
Entradas: Arreglo E, e índices p y u que delimitan el subarreglo a
      ordenar contenido en E[p,...,u].
Salidas: E[p,...,u] en orden ascendente.
1     si (p < u) entonces            //caso progresivo
2         m ← ⌊(p + u)/2⌋
3         mergeSort(E, p, m)
4         mergeSort(E, m + 1, u)
5         fusionar(E, p, m, u)
```

El caso base del algoritmo `mergeSort` no está explicito en el algoritmo, pero sucede cuando $p \geq u$, e implica que no se realizan comparaciones cuando el arreglo es de tamaño 1.

$$W(n)_{\text{paso implicito}} = 0, \quad \text{si } n=1$$

Cuando el arreglo es de tamaño mayor de n, el número de comparaciones se puede derivar aplicando la fórmula para los algoritmos de tipo divide-y-venceras.

$$W(n) = \underbrace{2W\left(\left\lfloor n/2 \right\rfloor\right)}_{\text{Comparaciones para ordenar por separado las parte derecha e izquierda que son de tamaño } \lfloor n/2 \rfloor \ (b=2,\ c=2).} + \underbrace{(n-1)}_{\text{Comparaciones para fusionar dos subarreglos ordenados de tamaño } \lfloor n/2 \rfloor. \text{ En el peor caso se toma alternadamente un elemento de cada subarreglo, requiriendose } n\text{-}1 \text{ comparaciones } (f(n) = n\text{-}1)}, \quad \text{si } n>1$$

Reuniendo en una sola expresión los casos base y recursivo, la complejidad del algoritmo `mergeSort` en el peor caso es como sigue.

$$W(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2W\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + (n-1) & \text{si n} > 0 \end{cases}$$

3.3.2 Algoritmos basados en recorta-y-vencerás

El problema principal de tamaño n se puede *recortar* a b subproblemas $(1<b< n)$, cada uno de tamaño n-c $(1<c< n)$, con costo no recursivo $f(n)$ debido a la tarea de *recortar* y *combinar*.

$$T(n) = b\ T(n\text{-}c) + f(n)$$

Número de subproblemas (factor de ramificación)

Costo de recortar en subproblemas y combinar soluciones

Costo de resolver un problema obtenido por recorte

Si los subproblemas son de distinto tamaño se pueden obtener límites asintóticos que acotan el desempeño.

Cota superior expresada con O(n): utilizando $T(n) \leq b\ T(n - c_{\max}) + f(n)$
Cota inferior expresada con $\Omega(n)$: utilizando $T(n) \geq b\ T(n - c_{\min}) + f(n)$

3.4 Solución de recurrencias mediante el método iterativo

Este método convierte la función de recurrencia en una sumatoria que posteriormente se evalúa para obtener un límite de la función. De manera general el método iterativo consiste en: a) Expansión de la recurrencia para identificar patrones de regularidad y expresarlos como sumas de términos dependientes sólo de n y las condiciones iniciales y, b) Evaluación de la sumatoria.

3.4.1 Pasos del método iterativo

La siguiente secuencia detallada de pasos permite determinar la complejidad ($O(n)$, $\Theta(n)$, o $\Omega(n)$) de un algoritmo recursivo.

Paso 1. Determinar una función $T(n)$ que represente el número de operaciones básicas efectuadas por el algoritmo para el tamaño base y para los casos definidos por recursividad

fuera de la base. Para simplificar la explicación del método usaremos la siguiente estructura de recurrencia que genera en cada nivel b problemas y denotaremos con n_i al tamaño del problema en el nivel de recurrencia i.

$$T(n_i) = \begin{cases} c & \text{si } n_i \text{ corresponde al caso base} \\ bT(n_{i+1}) + f(n_i) & \text{si } n_i \text{ corresponde al caso progresivo} \end{cases}$$

Paso 2. Evaluar la función $T(n)$ del caso progresivo con un conjunto pequeño de valores consecutivos de n.

$$\begin{aligned} T(n_0) &= f(n_0) + bT(n_1) \\ T(n_1) &= f(n_1) + bT(n_2) \\ T(n_2) &= f(n_2) + bT(n_3) \\ T(n_3) &= f(n_3) + bT(n_4) \\ T(n_4) &= f(n_4) + bT(n_4) \end{aligned}$$

Paso 3. Sustituir los resultados del punto anterior en cada expresión antecesora.

$$\mathrm{T}(n_0) = f(n_0) + bf(n_1) + b^2 f(n_2) + b^3 f(n_3) + b^4 T(n_4)$$

Paso 4. Identificar un patrón con el cual pueda generalizarse la representación de los términos del desarrollo.

Paso 5. Expresar en notación de sumatorias el patrón identificado, destacando dos partes en la expresión: un conjunto de términos inductivos y un término llamado de base o de paro, referenciado por m.

Paso 6. Determinar un valor de m para el cual se satisfaga o al menos se aproxime a la igualdad

$$\begin{pmatrix} \text{tamaño de problema} \\ \text{del término} \\ \text{referenciado por } m \end{pmatrix} = \begin{pmatrix} \text{tamaño de problema} \\ \text{que detiene la} \\ \text{recursividad} \end{pmatrix}$$

Paso 7. Sustituir m en la expresión de sumatoria obtenida en el paso 5.

Paso 8. Resolver la sumatoria con una expresión equivalente o aproximada cuya única variable independiente es n. Para esta tarea puede ser de gran ayuda utilizar programas computacionales como Derive.

3.4.2 Ejemplos de recurrencias resueltas con el método iterativo

La siguiente recurrencia se utilizará para ilustrar el método iterativo.

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

Primero se generan cuatro recurrencias para tamaños de problema consecutivos.

$$T(n) = n + 2T\left(n/2\right)$$
$$T\left(n/2\right) = \frac{n}{2} + 2T\left(n/4\right)$$
$$T\left(n/4\right) = \frac{n}{4} + 2T\left(n/8\right)$$
$$T\left(n/8\right) = \frac{n}{8} + 2T\left(n/16\right)$$

Enseguida se usan las recurrencias de anteriores para hacer la expansión de la recurrencia original.

$$T(n) = n + 2\,n/2 + 2\cdot 2T\left(n/4\right)$$
$$T\left(n\right) = n + 2\,n/2 + 2\cdot 2\,n/4 + 2\cdot 2\cdot 2T\left(n/8\right)$$
$$T\left(n\right) = n + 2\,n/2 + 2\cdot 2\,n/4 + 2\cdot 2\cdot 2\,n/8 + 2\cdot 2\cdot 2\cdot 2T\left(n/16\right)$$
$$T\left(n\right) = n + 2^1\,n/2^1 + 2^2\,n/2^2 + 2^3\,n/2^3 + 2^4 T\left(n/2^4\right)$$

Dado que se desconoce en cuántos términos la función $T(n)$ debe ser expandida para alcanzar el caso base, se usará el índice m para hacer referencia al término correspondiente a la condición de paro.

$$T(n) = \left[n + 2^1\,n/2^1 + 2^2\,n/2^2 + 2^3\,n/2^3 + \ \ldots + \text{término}_{m-1}\right] + \left[\text{término}_m\right]$$

Se forma una sumatoria cuyo patrón de regularidad es $2^i\left(n/2^i\right)$ con i= 0…m-1. Posterior a esta sumatoria sigue el término de paro, cuyo coeficiente y tamaño de problema siguen un patrón similar con i=m.

$$T(n) = \left[n + 2^1\,n/2^1 + 2^2\,n/2^2 + 2^3\,n/2^3 + \ \ldots + \ 2^{m-1}\left(n/2^{m-1}\right)\right] + \left[2^m T\left(n/2^{m-1}\right)\right]$$
$$T(n) = \sum_{i=0}^{m-1} 2^i\left(n/2^{m-1}\right) + 2^m T\left(n/2^{m-1}\right)$$

La sumatoria se simplifica al considerar que el último termino corresponde al caso base con n=1 y $T(1)$=1.

$$T(n) = \sum_{i=0}^{m-1} 2^i\left(n/2^i\right) + 2^m T\left(n/2^{m-1}\right) = \sum_{i=0}^{m-1} n + 2^m T\left(1\right) = \sum_{i=0}^{m-1} n + 2^m$$

Encontramos el valor de m igualando las dos condiciones de paro.

$$T\left(n/2^m\right) = T(1)$$
$$n/2^m = 1$$
$$2^m = n$$
$$\log_2(2^m) = \log_2(n)$$
$$m = \log_2(n)$$

La solución de la recurrencia indica que el algoritmo tiene un crecimiento *n*-logarítmico.

$$T(n) = \sum_{i=0}^{\log_2(n)-1} n + 2^{\log n}$$
$$T(n) = n\log n + n^{\log 2} = n\log n + n = \Theta(n\log n)$$

Enseguida, un segundo ejemplo para clarificar el método iterativo.

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

Primero se generan recurrencias para diferentes tamaños de problema.

$$T(n) = n + 3T\left(\left\lfloor n/4 \right\rfloor\right)$$
$$T\left(\left\lfloor n/4 \right\rfloor\right) = \frac{n}{4} + 3T\left(\left\lfloor n/16 \right\rfloor\right)$$
$$T\left(\left\lfloor n/16 \right\rfloor\right) = \frac{n}{16} + 3T\left(\left\lfloor n/64 \right\rfloor\right)$$
$$T\left(\left\lfloor n/64 \right\rfloor\right) = \frac{n}{64} + 3T\left(\left\lfloor n/256 \right\rfloor\right)$$

Enseguida se usan las recurrencias anteriores para hacer la expansión de la recurrencia original.

$$T(n) = n + 3T\left(\left\lfloor n/4 \right\rfloor\right)$$
$$T(n) = n + \frac{3n}{4} + 9T\left(\left\lfloor n/16 \right\rfloor\right)$$
$$T(n) = n + \frac{3n}{4} + \frac{9n}{16} + 27T\left(\left\lfloor n/64 \right\rfloor\right)$$
$$T(n) = n + \frac{3n}{4} + \frac{9n}{16} + \frac{27n}{64} + 81T\left(\left\lfloor n/256 \right\rfloor\right)$$

Dado que se desconoce en cuántos términos la función $T(n)$ debe ser expandida para alcanzar el caso base, se usará el índice m para hacer referencia al término correspondiente a la condición de paro.

$$T(n)=\left(n+\frac{3n}{4}+\frac{9n}{16}+\frac{27n}{64}\ +\ ...\right)+\left[\text{término referenciado por } m\right]$$

Antes de la última llamada recursiva se forma una sumatoria cuyo patrón de regularidad (término general) es $3^i\left\lfloor n/4^i\right\rfloor$ con $i= 0\ldots m\text{-}1$. Después sigue el término asociado a la condición de paro, cuyo coeficiente y tamaño de problema siguen un patrón similar con $i=m$.

$$T(n)=\left(n+\frac{3n}{4}+\frac{9n}{16}+\frac{27n}{64}\ +\ ...+3^{m-1}\left\lfloor \frac{n}{4^{m-1}}\right\rfloor\right)+\left(3^m T\left(\left\lfloor \frac{n}{4^m}\right\rfloor\right)\right)$$

$$T(n)=\sum_{i=0}^{m-1}3^i\left\lfloor \frac{n}{4^i}\right\rfloor+3^m T\left(\left\lfloor \frac{n}{4^m}\right\rfloor\right)$$

El último término de la sumatoria se simplifica ya que éste corresponde al caso base con $n=1$ y $T(1)=\Theta(1)$.

$$T(n)=\sum_{i=0}^{m-1}3^i\left\lfloor \frac{n}{4^i}\right\rfloor+3^m T\left(1\right)$$

Usando el límite $\left\lfloor \frac{n}{4^m}\right\rfloor\leq \frac{n}{4^m}$, encontramos el valor de m que hace referencia al último término de la sumatoria.

$$T\left(\left\lfloor \frac{n}{4^m}\right\rfloor\right)\leq T\left(\frac{n}{4^m}\right),\ \text{como } T\left(\left\lfloor \frac{n}{4^m}\right\rfloor\right)=T(1),\ \text{entonces } T\left(\frac{n}{4^m}\right)\geq T(1)$$

$$\frac{n}{4^m}\geq 1$$

$$4^m\leq n$$

$$\log_4(4^m)\leq \log_4(n)$$

$$m\leq \log_4(n)$$

El valor de m se sustituye en la recurrencia expandida.

$$T(n)\leq \sum_{i=0}^{\log_4(n)-1}3^i\left\lfloor \frac{n}{4^i}\right\rfloor+3^{\log_4(n)}\Theta\left(1\right)$$

$$T(n)\leq \sum_{i=0}^{\log_4(n)-1}3^i\left\lfloor \frac{n}{4^i}\right\rfloor+\Theta\left(3^{\log_4(n)}\right)$$

$$T(n)\leq \sum_{i=0}^{\log_4(n)-1}3^i\left\lfloor \frac{n}{4^i}\right\rfloor+\Theta\left(n^{\log_4(3)}\right)$$

El límite superior de la sumatoria de $T(n)$ se puede sustituir por infinito para utilizar una serie geométrica infinita.

$$\sum_{i=0}^{\infty} r^i = 1/(1-r) \quad \text{con } r = 3/4$$

$$T(n) \le n\sum_{i=0}^{\infty}(3/4)^i + \Theta\left(n^{\log_4(3)}\right)$$

$$T(n) \le n\left(\frac{1}{1-3/4}\right) + \Theta\left(n^{\log_4(3)}\right)$$

$$T(n) \le 4n + \Theta\left(n^{\log_4(3)}\right)$$

$$T(n) = O(n)$$

Un límite superior más ajustado para $T(n)$ se obtiene con una serie geométrica finita.

$$\sum_{i=0}^{k} ar^i = a\left((r^{k+1}-1)/(r-1)\right).$$

Con las series geométricas decrecientes, la suma total es a lo más un factor constante mayor que el primer término. Por lo tanto en la solución prevalece el primer término n que es menor que $4n$ por un factor de 4.

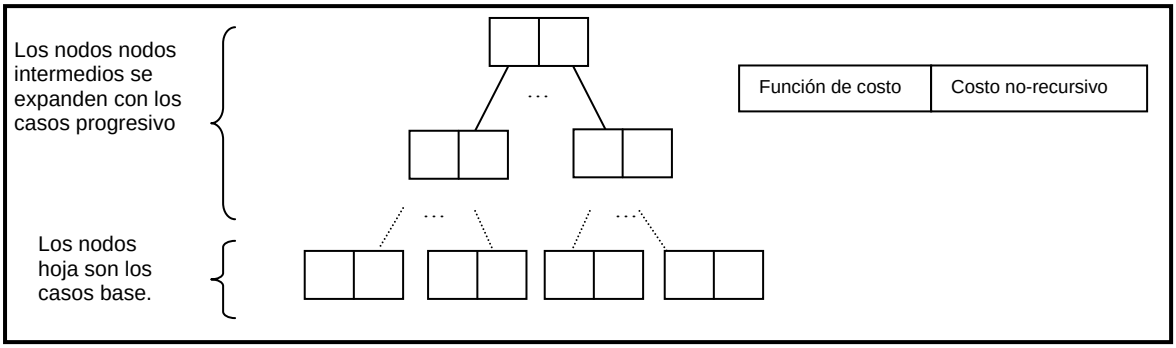
$$n\sum_{i=0}^{\infty}(3/4)^i = n + \frac{3n}{4} + \frac{9n}{16} + \frac{27n}{64} + \ ... = 4n$$

3.5 Solución de recurrencias mediante el método de árbol de recursión

Los árboles de recursión son una herramienta visual para analizar el costo de procedimientos recursivos asociados a una estructura de árbol, por ejemplo los algoritmos de divide-y-venceras.

3.5.1 Descripción del método de árbol de recursión

Se construye el árbol para organizar por niveles las operaciones algebraicas necesarias para resolver la recurrencia. Cada nodo del árbol tiene una estructura de dos componentes: la función de costos y el costo no-recursivo.

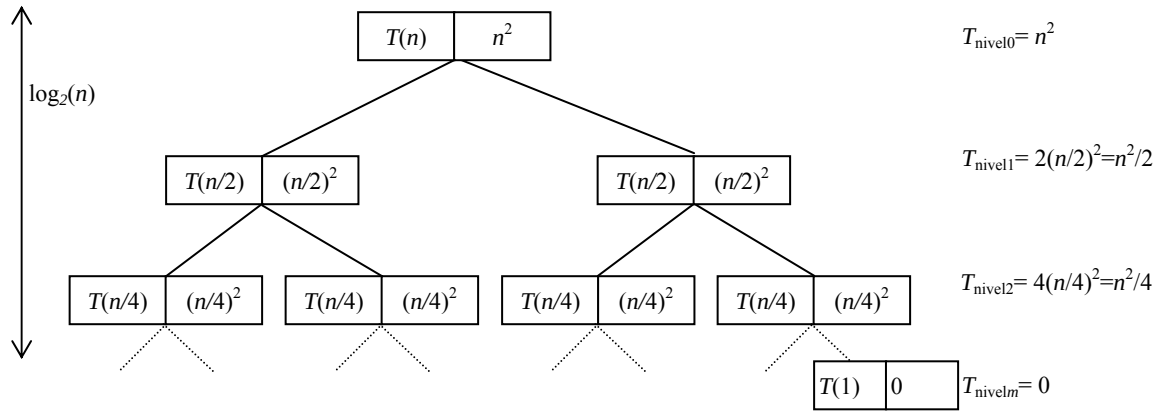


3.5.2 Ejemplos de recurrencias resueltas con el método de árbol de recursión

El siguiente ejemplo ilustra el método de solución de recurrencias basado en árbol de recursión.

$$T(n)=\begin{cases}2T(n/2)+n^2 & n>1\\ 0 & n=1\end{cases}$$

Primero se construye el árbol de recurrencia determinando para cada nodo la función de costos y el costo no recursivo. Para cada nivel se calcula el costo total.



Un análisis visual del árbol revela que en cada nivel el número de subproblemas aumenta en potencias de dos.

$$\left(2^0\right)\rightarrow\left(2^1\right)\rightarrow\left(2^2\right)\rightarrow\ldots\rightarrow\left(2^m\right)$$

En cada nivel el tamaño de los problemas disminuye en potencias de dos.

$$\left(\frac{1}{2^0}n\right)\rightarrow\left(\frac{1}{2^1}n\right)\rightarrow\left(\frac{1}{2^2}n\right)\rightarrow\ldots\rightarrow\left(\frac{1}{2^m}n\right)$$

En cada nivel la complejidad algorítmica total incrementa en potencias de dos. Es importante observar que en el último nivel se tienen 2^m nodos y cada nodo es de tamaño $(1/2^m)n$.

$$\left(\frac{1}{2^0}n^2\right)\rightarrow\left(\frac{1}{2^1}n^2\right)\rightarrow\left(\frac{1}{2^2}n^2\right)\rightarrow\ldots\rightarrow\left(2^m\right)T\left(\frac{1}{2^m}n\right)$$

Usando el patrón de complejidad y las condiciones de terminación, la recurrencia se expresa como una sumatoria.

$$T(n)=\left(n^2+\frac{1}{2}n^2+\frac{1}{4}n^2+\ldots+\frac{1}{2^{m-1}}n^2\right)+2^mT\left(\frac{1}{2^m}n\right)$$

$$T(n)=\left(n^2+\frac{1}{2}n^2+\frac{1}{4}n^2+\ldots+\frac{1}{2^{m-1}}n^2\right)+0=n^2\sum_{i=0}^{m-1}\frac{1}{2^i}+0$$

El valor de *m* se determina igualando las dos expresiones del caso base.

$$T\left(\frac{1}{2^m}n\right) = T(1)$$

$$\frac{1}{2^m}n = 1; \quad 2^m = n; \quad \log 2^m = \log n$$

$$m = \log n$$

Para la solución de $T(n)$ se usa la serie geométrica con $r=1/2$.

$$\sum_{i=0}^{k} ar^i = a\left((r^{k+1}-1)/(r-1)\right)$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

La solución de la recurrencia $T(n)$ revela que el algoritmo tiene una velocidad de crecimiento cuadrática

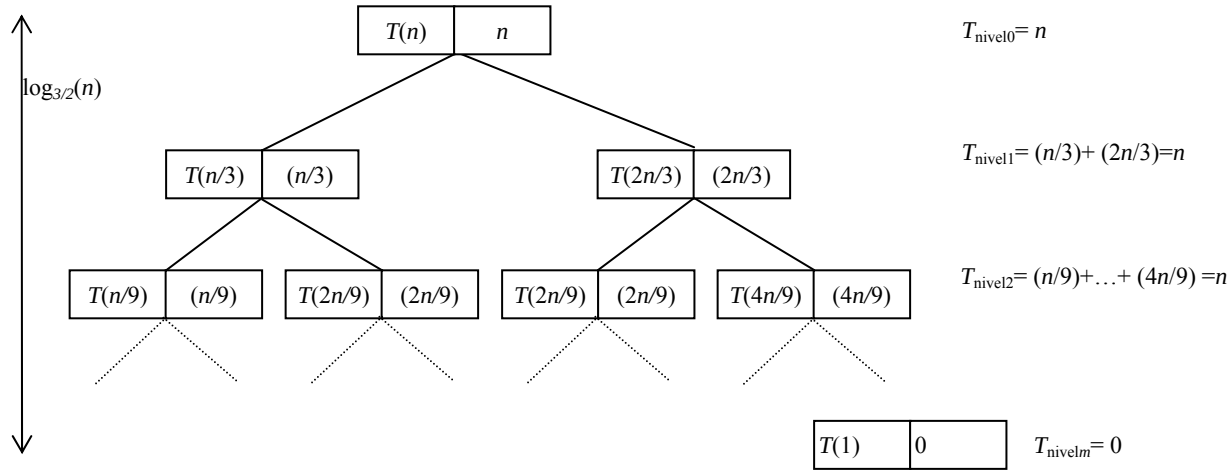
$$T(n) = n^2 \sum_{i=0}^{(\log n)-1} \frac{1}{2^i} = n^2\left(2 - \left(\frac{1}{2^{(\log n)-1}}\right)\right) = n^2\left(2 - \left(\frac{2}{n}\right)\right) = 2n^2 - 2n$$

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

Con un segundo ejemplo se ilustra el método de árbol de recursión.

$$T(n) = \begin{cases} T(n/3) + T(2n/3) + n & n > 1 \\ 0 & n = 1 \end{cases}$$

Primero se construye el árbol de recurrencia determinando para cada nodo la función de costos y el costo no recursivo. Para cada nivel se calcula el costo total.



En cada nivel el número de subproblemas aumenta en potencias de dos.

$$\left(2^0\right) \rightarrow \left(2^1\right) \rightarrow \left(2^2\right) \rightarrow \ldots \rightarrow \left(2^m\right)$$

En cada nivel el tamaño más grande de problema disminuye por un factor de $(2/3)^i$.

$$\left(\frac{2}{3}\right)^0 n \rightarrow \left(\frac{2}{3}\right)^1 n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \ldots \rightarrow \left(\frac{2}{3}\right)^{m-1} n \rightarrow \left(\frac{2}{3}\right)^m n$$

En cada nivel la complejidad algorítmica permanece constante.

$$(n) \rightarrow (n) \rightarrow (n) \rightarrow \ldots \rightarrow \left(2^m\right) T\left(\left(\frac{2}{3}\right)^m n\right)$$

Usando el patrón de complejidad y las condiciones de terminación, la recurrencia se expresa como una sumatoria. Dado que esta sumatoria corresponde a la trayectoria más larga (subproblemas mayores), se obtiene una cota superior de las trayectorias restantes.

$$T(n) \leq n + n + n + \ldots + n + \left(2^m\right) T\left(\left(\frac{2}{3}\right)^m n\right)$$

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

El valor de *m* se determina igualando las dos expresiones que representan la condición de paro.

$$T\left(\frac{2}{3}\right)^m n = T(1)$$

$$\left(\frac{2}{3}\right)^m n = 1$$

$$\left(\frac{3}{2}\right)^m = n$$

$$\log_{3/2}\left(\frac{3}{2}\right)^m = \log_{3/2} n$$

$$m = \log_{3/2} n$$

Para la solución de $T(n)$ se usa una diferencia de límites.

$$T(n) \le \sum_{i=0}^{m-1}(n)$$

$$T(n) \le \sum_{i=0}^{\log_{3/2}(n)-1}(n)$$

$$T(n) \le n\sum_{i=0}^{\log_{3/2}(n)-1}(1)$$

$$T(n) \le n(\log_{3/2}(n)-1+1)$$

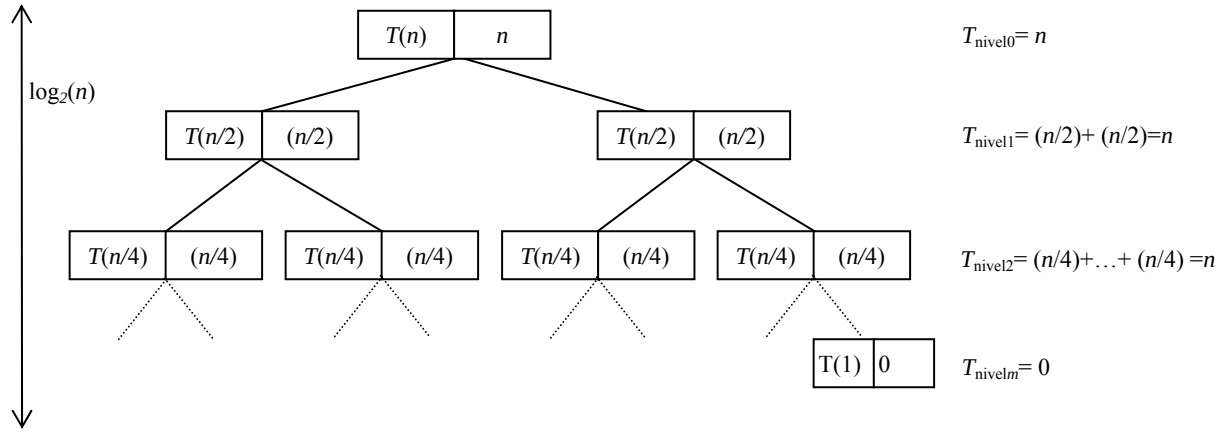
$$T(n) \le n\log_{3/2}(n)$$

$$T(n) = O(n\log n)$$

Con un tercer ejemplo se ilustra el método de árbol de recursión.

$$T(n) = \begin{cases} 2T(n/2)+n & n>1 \\ 0 & n=1 \end{cases}$$

Primero se construye el árbol de recurrencia determinando para cada nodo la función de costos y el costo no recursivo acordes al nivel de recursividad que le corresponda. Para cada nivel se calcula el costo total.



En cada nivel el número de subproblemas aumenta en potencias de dos.

$$\left(2^0\right) \to \left(2^1\right) \to \left(2^2\right) \to \ldots \to \left(2^m\right)$$

Estableciendo el supuesto que en todos los niveles del árbol *n* es par, en cada nivel el tamaño del problema disminuye en potencias de dos.

$$\left(\frac{1}{2^0}n\right) \to \left(\frac{1}{2^1}n\right) \to \left(\frac{1}{2^2}n\right) \to \ldots \to \left(\frac{1}{2^m}n\right)$$

En cada nivel la complejidad algorítmica permanece constante.

$$(n) \to (n) \to (n) \to \ldots \to \left(2^m\right) T\left(\frac{1}{2^m} n\right)$$

Usando el patrón de complejidad y las condiciones de terminación, la recurrencia se expresa como una sumatoria.

$$T(n) = n + n + n + \ldots + n + \left(2^m\right) T\left(\frac{1}{2^m} n\right) = \sum_{i=0}^{m-1} (n) + 0$$

El valor de *m* se determina igualando las dos expresiones que representan la condición de paro.

$$T\left(\frac{n}{2^m}\right) = 1$$

$$\frac{n}{2^m} = 1$$

$$2^m = n$$

$$\log_2\left(2^m\right) = \log_2 n$$

$$m = \log_2 n$$

Para la solución de $T(n)$ se usa una diferencia de límites.

$$T(n) = n + n + n + \ldots + n + 0$$

$$T(n) = \sum_{i=0}^{m-1} (n) = T(n) = \sum_{i=0}^{\log_2(n)-1} (n) = T(n) = n \sum_{i=0}^{\log_2(n)-1} (1)$$

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

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

3.6 Solución de recurrencias mediante el método maestro

El método maestro es una receta para resolver recurrencias derivadas de aplicar la técnica de divide-y-vencerás.

$$T(n) = b\ T(\ n/c\ ) + f(n)$$

3.6.1 Descripción del método maestro

En la aplicación del método se puede omitir el piso y techo del tamaño del problema ya que no afecta a la solución de la recurrencia porque ésta se expresa con funciones asintóticas.

$$n/c \approx \lfloor n/c \rfloor \approx \lceil n/c \rceil$$

La solución de una recurrencia está sujeta al caso que le corresponda. Se pueden probar cada uno de los casos, o usar el árbol de recursividad para identificar la serie que mejor lo describe.

Caso 1: La complejidad es una serie geométrica creciente

$$\text{si } f(n) = O(n^{\log_c b - \varepsilon}) \text{ para alguna constante } \varepsilon > 0$$

$$\text{entonces } T(n) = \Theta(n^{\log_c b})$$

Caso 2: La complejidad es constante en cada nivel

$$\text{si } f(n) = \Theta(n^{\log_c b})$$
$$\text{entonces } T(n) = \Theta(n^{\log_c b} \lg n)$$

Caso 3: La complejidad es una serie geométrica decreciente

$$\text{si } f(n) = \Omega(n^{\log_c b + \varepsilon}) \text{ para alguna constante } \varepsilon > 0, \text{ y}$$

$$\text{si } bf(n/c) \leq c'f(n) \text{ para alguna constante } c' < 1 \text{ y para } n \text{ suficientemente grande,}$$
$$\text{entonces } T(n) = \Theta(f(n))$$

La segunda condición prueba que $bf(n/c)$, la complejidad del nivel 1, disminuye o se mantiene con respecto a $c'f(n)$, que es la complejidad del nivel 0.

3.6.2 Interpretación del método maestro

En los tres casos se compara $f(n)$, que es el primer término de la sumatoria expandida de la recurrencia original, con $n^{\log_c b}$, que es el último:

Recurrencia

$$T(n) = bT(n/c) + f(n)$$

Sumatoria

$$T(n) = f(n) + bf(n/c) + \ldots + n^{\log_c b}$$

a) En el caso 1 la función $f(n)$ es polinomialmente más pequeña que $n^{\log_c b}$. Esto es, $f(n)$ es asintóticamente más pequeña por un factor de n^{ε}. Esto sucede en las series geométricas crecientes.
b) En el caso 2 la función $f(n)$ es asintóticamente equivalente a $n^{\log_c b}$, esto sucede en las series constantes.

c) En el caso 3 la función $f(n)$ es polinomialmente más grande que $n^{\log_c b}$ por un factor de n^{ε}. Esto es, $f(n)$ es asintóticamente más grande por un factor de n^{ε}. esto sucede en las series geometricas decrecientes.

La superioridad de una función $h(n)$ sobre otra $g(n)$ se dice que es polinomial, si es significativa. Esto es, existe una constante $\varepsilon>0$ tal que $g(n)$ es asintóticamente más pequeña que $h(n)$ por un factor n^{ε}.

$$n^{\varepsilon} g(n) \leq h(n)$$

En el caso 1:

$$h(n) = n^{\log_c b}, g(n) = f(n)$$
$$n^{\varepsilon} f(n) \leq n^{\log_c b}$$
$$f(n) \leq n^{\log_c b - \varepsilon}$$

En el caso 3:

$$h(n) = f(n), \quad g(n) = n^{\log_c b}$$
$$n^{\varepsilon} n^{\log_c b} \leq f(n)$$
$$f(n) \geq n^{\log_c b + \varepsilon}$$

Las recurrencias en las que $f(n)$ es superior o inferior por un factor que no es polinomial, no deben ser resueltas con el método maestro.

3.6.3 Ejemplos de recurrencias resueltas con el método maestro

El siguiente ejemplo ilustra el primer caso correspondiente a una serie geométrica creciente.

$$T(n) = \begin{cases} 9T(n/3) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

De la fórmula general de divide-y-vencerás:

$T(n) = b\ T(\ n/c\ ) + f(n)$
b=9, c=3, $f(n)$ =n,

Para determinar que la recurrencia corresponde al caso 1 del Teorema Maestro, sea:

$$f(n) = n$$
$$g(n) = n^{\log_c b - \varepsilon} = n^{\log_3 9 - \varepsilon} = n^{2-\varepsilon}$$

Se demuestra que $f(n) = O(g(n))$:

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right)=\lim_{x\to\infty}\left(\frac{n}{n^{2-\varepsilon}}\right)$$

$$\lim_{x\to\infty}\left(\frac{n}{n^{2-\varepsilon}}\right)=\frac{1}{n^{1-\varepsilon}}=1 \text{ si } \varepsilon=1$$

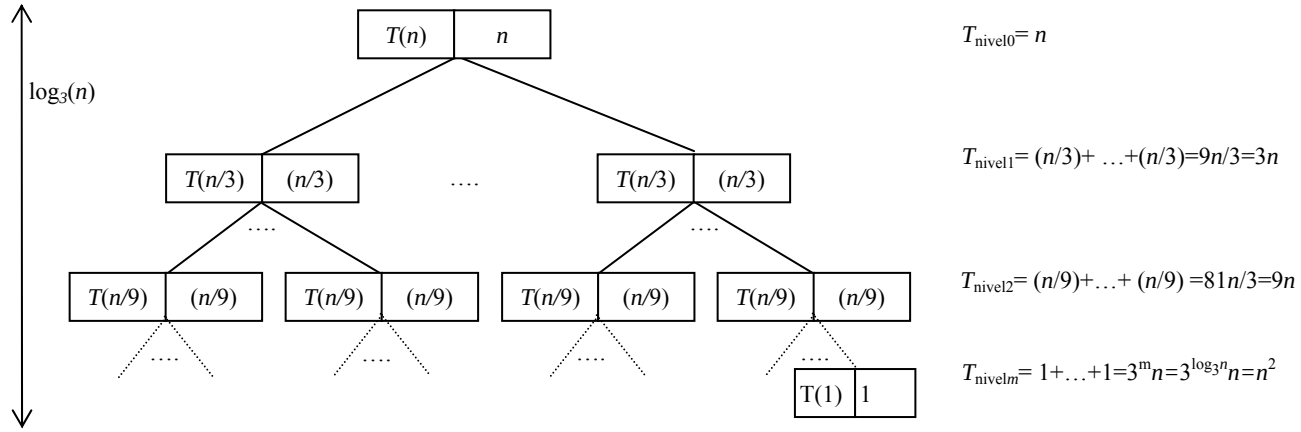
$$\lim_{x\to\infty}\left(\frac{n}{n^{2-\varepsilon}}\right)=\frac{1}{n^{1-\varepsilon}}=0 \text{ si } 0>\varepsilon<1$$

$$\therefore f(n)=\mathrm{O}(n^{\log_c b-\varepsilon}) \text{ para } \varepsilon>0$$

Por lo anterior, entonces el último término de la recurrencia es el dominante,

$$\therefore T(n)=\Theta(n^{\log_c b})=\Theta(n^{\log_3 9})=\Theta(n^2)$$

Alternativamente se puede resolver la recurrencia mediante la construcción del árbol de recursión.



Se obtiene el patrón del tamaño de problema: $n/3^i$

$$\left(\frac{1}{3}\right)^0 n\to\left(\frac{1}{3}\right)^1 n\to\left(\frac{1}{3}\right)^2 n\to\ldots\to\left(\frac{1}{3}\right)^{m-1} n\to\left(\frac{1}{3}\right)^m n$$

En la profundidad $i=m$ del árbol el tamaño del problema es 1

$$\frac{n}{3^m}=1$$

$$3^m=n$$

$$\log_3\left(2^m\right)=\log_3 n$$

$$m=\log_3 n$$

Se obtiene el patrón de complejidad $3^i n$, que da lugar a una serie creciente donde el último término n^2 es el dominante.

$$T(n) = n + 3n + 9n + ... + 3^{m-1}n + 0$$
$$T(n) = n + 3n + 9n + ... + 3^{(\log_3 n)-1}n$$
$$T(n) = n \sum_{i=0}^{(\log_3 n)-1} (3^i)$$

Usando la serie $\sum_{i=0}^{k} ar^i = a\left((r^{k+1}-1)/(r-1)\right)$, se resuelve la recurrencia.

$$T(n) = n\left(\frac{(3^{\log_3(n)}-1)}{3-1}\right) = \frac{n(n-1)}{2} == \frac{n^2-n}{2}$$
$$T(n) = \Theta(n^2)$$

El siguiente ejemplo ilustra el segundo caso correspondiente a una serie constante.

$$T(n) = \begin{cases} T(2n/3)+1 & n>1 \\ 0 & n=1 \end{cases}$$

De la fórmula general de divide-y-vencerás:

$$T(n) = b\ T(\ n/c\ ) + f(n),$$

b=1, c=3/2, f(n) =1,

Para determinar que la recurrencia corresponde al caso 2 del Teorema Maestro, sea,

$$f(n) = 1$$
$$g(n) = n^{\log_c b} = n^{\log_{3/2} 1} = n^0 = 1$$

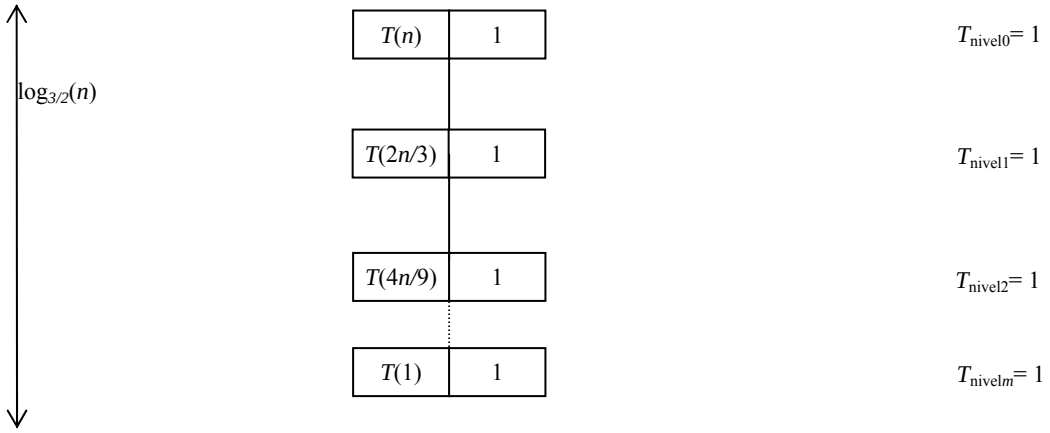
Se demuestra que $f(n) = \Theta(g(n))$:

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{x\to\infty}\left(\frac{1}{1}\right) = 1$$
$$\therefore\ f(n) = \Theta(n^{\log_c b})$$

Como $f(n) = \Theta(n^{\log_c b})$, entonces todos los $\lg n$ términos de la recurrencia son iguales al último término,

$$\therefore T(n) = \Theta(n^{\log_c b} \lg n) = \Theta(n^{\log_{3/2} 1} \lg n) = \Theta(\lg n)$$

Alternativamente se puede resolver la recurrencia mediante la construcción del árbol de recursión.



Se obtiene el patrón del tamaño de problema: $(2/3)^i n$.

$$\left(\frac{2}{3}\right)^0 n \to \left(\frac{2}{3}\right)^1 n \to \left(\frac{2}{3}\right)^2 n \to \ldots \to \left(\frac{1}{3}\right)^{m-1} n \to \left(\frac{2}{3}\right)^m n = 1$$

En la profundidad $i=m$ del árbol, el tamaño del problema es 1.

$$\left(\frac{2}{3}\right)^m n = 1$$

$$\left(\frac{3}{2}\right)^m = n$$

$$\log_{3/2}\left(\frac{3}{2}\right)^m = \log_{3/2} n$$

$$m = \log_{3/2} n$$

Se obtiene como patrón de complejidad una constante que prevalece hasta la profundidad del árbol.

$$T(n) = 1+1+1+\ldots+1$$

$$T(n) = \sum_{i=0}^{\log_{3/2} n} 1$$

$$T(n) = \log_{3/2}(n) + 1$$

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

El siguiente ejemplo ilustra el tercer caso correspondiente a una serie geométrica decreciente.

$$T(n) = \begin{cases} 3T(n/4) + n\lg n & n > 0 \\ 0 & n = 0 \end{cases}$$

De la fórmula general de divide-y-venceras:

$T(n) = b\ T(\ n/c\ ) + f(n),$
b=3, c=4, $f(n) = n\lg n.$

Para determinar que la recurrencia corresponde al caso 3 del Teorema Maestro, sea

$$f(n) = n \lg n$$
$$g(n) = n^{\log_c b+\varepsilon} = n^{\log_4 3+\varepsilon}$$

Se demuestra que $f(n) = \Omega(g(n))$

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{x\to\infty}\left(\frac{n\lg n}{n^{\log_4 3+\varepsilon}}\right)$$
$$\lim_{x\to\infty}\left(\frac{n\lg n}{n^{0.793}n^{\varepsilon}}\right) = \lim_{x\to\infty}\lg n = \infty \quad \text{si } \varepsilon = 0.207$$
$$\lim_{x\to\infty}\left(\frac{n\lg n}{n^{0.793}n^{\varepsilon}}\right) = \infty \quad \text{si } 0<\varepsilon \le 0.207$$
$$\therefore\ f(n) = \Omega(n^{\log_c b+\varepsilon}) \text{ para } \varepsilon > 0$$

Además, se demuestra que $bf(n/c) \le c'f(n)$ para alguna constante $c'<1$ y para n suficientemente grande

$$3\left(\frac{n}{4}\right)\left(\lg\frac{n}{4}\right) \le c'n\lg n \quad \text{si } c' = \frac{3}{4}$$
$$\frac{3}{4}n\lg\frac{n}{4} \le \frac{3}{4}n\lg n$$
$$\therefore bf(n/c) \le c'f(n) \text{ para } c'<1$$

Por lo anterior,

$$\therefore T(n) = \Theta(\ f(n)) = \Theta(n\lg n)$$

3.7 Solución de recurrencias mediante el método de substitución

Se usa cuando las ecuaciones son no lineales. Mediante inducción matemática se prueba que una solución adivinada para $T(n)$ es correcta en condiciones límite.

3.7.1 Pasos del método de substitución

Paso1: Adivinación

Se conjetura que la solución de $T(n)$ pertenece a algún orden de $O(f(n))$.

Paso2: Prueba inductiva

a) Hipótesis: La función adivinada se establece como hipótesis inductiva

$T(n) \leq cf(n)$ para una $c>0$

b) Caso Base: Para n pequeña se prueba $T(n) \leq cf(n)$.

La hipótesis inductiva se prueba con el tamaño de problema más pequeño. Como la solución adivinada se expresa en notación asintótica sólo basta encontrar una n_0, tal que la hipótesis inductiva se cumpla para cualquier $n > n_0$. El valor de n_0 se substituye en la recurrencia para encontrar su solución con n pequeña. La prueba es correcta si la hipótesis y la recurrencia coinciden en alguna constante c.

c) Caso General: Suponiendo $T(n) \leq cf(n)$ verdadera demostrar que $T(n\text{-}1) \leq cf(n\text{-}1)$
La hipótesis inductiva se prueba con un valor de problema grande. Para ello se asume que dicha hipótesis es válida con el tamaño anterior $k_{f\text{-}1}$ al del más grande que es k_f. Esta se substituye en la recurrencia evaluada en k_f, y se considera correcta si alcanza la forma exacta de la hipótesis inductiva.

3.7.2 Ejemplos de recurrencias resueltas con el método de sustitución

El siguiente ejemplo ilustra el método de sustitución.

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

Paso 1: Adivinación

Sea $T(n) = O(n\lg n)$ la solución adivinada

Paso 2: Prueba inductiva

a) Como hipótesis inductiva se establece la función

$T(n) \leq c(n\lg n)$ para una $c>0$.

b) Para probar que la hipótesis inductiva es válida con un tamaño de problema grande $k_f = n$, primero se asume que esta hipótesis es valida con un valor $k_{f\text{-}1} = \lfloor n/2 \rfloor$ anterior al de k_f.

La hipótesis inductiva de la solución adivinada es:

$$T(n) = O(n \log n)$$
$$T(n) \leq c(n \log n)$$

La hipótesis inductiva para $k_{f-1} = \lfloor n/2 \rfloor$ queda:

$$T\left(\lfloor n/2 \rfloor\right) \leq c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$$

La recurrencia original para $k_f = n$ es:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

La hipótesis inductiva que es valida en k_{f-1} se sustituye en la recurrencia especificada en el tamaño de problema más grande k_f. La recurrencia original, con la sustitución de la hipótesis inductiva, queda

$$T(n) \leq 2c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n$$
$$T(n) \leq cn \lg(n/2) + n$$
$$T(n) \leq cn \lg n - cn \lg 2 + n$$
$$T(n) \leq cn \lg n - cn + n$$

Para $c=1$ se encuentra la forma exacta de la hipótesis inductiva:

$$T(n) \leq (1)n \lg n - (1)n + n$$
$$T(n) \leq n \lg n$$

c) Se busca un valor pequeño de n para el cual la hipótesis inductiva es válida.

Para $n_0 = 1$, que es el tamaño de problema más pequeño, la hipótesis inductiva queda:

$$T(n) \leq cn \lg n$$
$$T(1) \leq c(1) \lg(1) = 0$$

La hipótesis es inconsistente con el caso base de la recurrencia ya que no existe una c que pueda obtener su condición límite $T(1)=1$.

$\therefore$ Cuando $n_0=1$ la hipótesis inductiva no se cumple.

Para $n_0 = 2$, la hipótesis inductiva queda:

$$T(n) \leq cn \lg n$$
$$T(2) \leq c(2) \lg(2) = 2c$$

El valor de $n_0 = 2$ se sustituye en la recurrencia:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$
$$T(2) = 2T(\lfloor 2/2 \rfloor) + 2$$
$$T(2) = 2(1) + 2 = 4$$

Para c =2 la hipótesis inductiva corresponde con la recurrencia.

$$T(2) \leq 2c$$
$$T(2) \leq 2(2) = 4$$

$\therefore$ Cuando n_0=2 la hipótesis inductiva se cumple.

Dado que se encontró la forma exacta de la hipótesis con n grande y no se llegó a inconsistencias con n pequeña, queda demostrado que $T(n) = O(n \log n)$.

3.7.3 Consejos para una buena adivinación

a) Usar la solución de una recurrencia similar a la recurrencia con solución conocida.

Recurrencia con solución conocida

$$T(n)_c = 2T(\lfloor n/2 \rfloor) + n$$
$$\mathrm{T}(n)_c = O(n\lg n)$$

Recurrencia con solución desconocida

$$T(n)_c = 2T(\lfloor n/2 \rfloor) + 32 + n$$

Probar la solución

$$\mathrm{T}(n)_c = O(n\lg n)$$

b) Probar límites superiores e inferiores holgados y después reducirlos hasta que converjan.

Para la recurrencia

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Probar como límite inferior el costo no recursivo

$$T(n) = \Omega(\lfloor n \rfloor)$$

Probar como límite superior una potencia del costo no recursivo

$$T(n) = \mathrm{O}(\lfloor n \rfloor)$$

Reducir el límite superior y elevar el límite inferior hasta que converjan en el correcto

$$\mathrm{T}(n) = \Theta(n\lg n)$$

c) Convertir una suposición inductiva débil en una fuerte, restando el término de más bajo orden**.**

Suposición inductiva débil

Recurrencia a resolver:

$$T(n) = 2T(\lfloor n/2 \rfloor) + 2T(\lceil n/2 \rceil) + 1$$

Adivinación de la fórmula de la solución:

$$T(n) = O(n)$$

Hipótesis inductiva: existe una $c>0$ que haga cierto que,

$$T(n) \le c(n)$$

Sustitución en la recurrencia de la hipótesis inductiva evaluada en $k_{f\text{-}1} = n/2$

$$T(n) \le cn/2 + cn/2 + 1$$
$$T(n) \le cn + 1$$

La hipótesis inductiva se rechaza ya que no existe una c que haga cierta la hipótesis en la recurrencia ya que para toda $c>1$, $cn \neq cn+1$

Suposición inductiva fuerte

El término de más bajo orden de la recurrencia es 1, el cual se resta de la hipótesis inductiva como una constante, quedando:

$$T(n) \le c(n) - b, \quad \text{con } b \ge 0$$

Sustituyendo en la recurrencia la hipótesis inductiva evaluada en $k_{f\text{-}1} = n/2$ se tiene:

$$T(n) \le cn/2 - b + cn/2 - b + 1$$
$$T(n) \le cn - 2b + 1$$

Para $b=1$ la recurrencia y la hipótesis coinciden

Recurrencia

$$T(n) \le cn - 2(1) + 1$$

$$T(n) \le cn - 1$$

Hipótesis

$$T(n) \le cn - b$$

$$T(n) \le cn - 1$$

d) Cambio de variables.

Mediante una manipulación algebraica hacer que una recurrencia desconocida sea similar a una conocida

Por ejemplo:

$T(n) = 2T(\sqrt{n}) + \lg n$.

Para eliminar el logaritmo se introduciendo una nueva variable $m=\lg n$.

$$n = 2^m$$

$$T(2^m) = 2T(\sqrt{2^m}) + m$$

$$T(2^m) = 2T(2^{m/2}) + m$$

La recurrencia original T se renombra con S, cuyo tamaño de problema es m.

$$S(m) = T(2^m)$$

$$S(m) = 2S(m/2) + m$$

La nueva recurrencia S es similar a otra recurrencia de solución conocida.

$$T(n) = 2T(n/2) + n$$

$$T(n) = O(n \log n)$$

Por lo tanto, S tiene la misma solución que la recurrencia conocida.

$$S(m) = O(m \log m)$$

Se restablecen los valores originales para obtener la solución de la recurrencia original.

$$T(n) = T(2^m) = S(m)$$

$$T(n) = O(m \lg m) = O(\lg n(\lg(\lg n))),$$

$$T(n) = O(\log n \lg^2 n)$$

3.8 Ejercicios resueltos

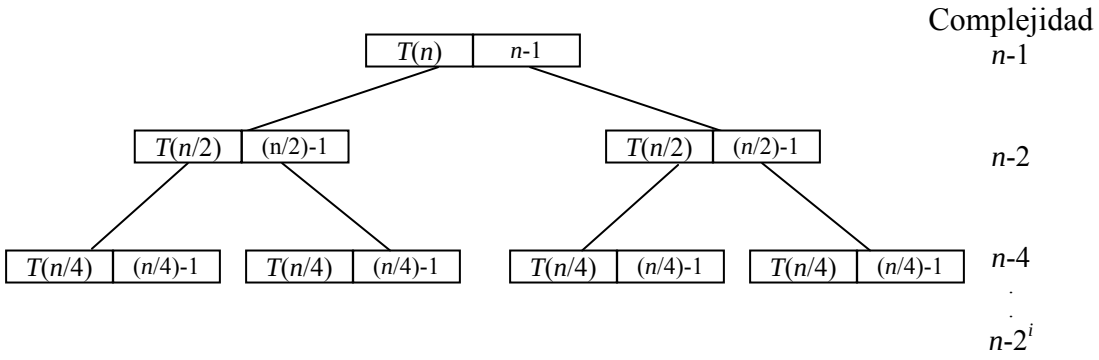
En los siguientes ejercicios, para la función de recurrencia dada, se aplican los métodos de solución indicados. Posteriormente se determine el orden asintótico de la función. Finalmente la solución la solución se comprueba utilizando la herramienta Derive.

3.8.1. Utilice el método maestro y al árbol de recurrencia para resolver la función $T(n)$ definida para todas las potencias de 2.

$$T(n) = n\text{-}1 + 2T\left(n/2\right)$$
$$T(1) = 0$$

a) **Aplicación de métodos de solución de recurrencias**

Mediante un análisis del árbol de recurrencia se observa que la complejidad en los diferentes niveles del árbol no es constante y tampoco es una serie geométrica. Por lo anterior el teorema maestro no se puede utilizar para resolver la recurrencia.



Para facilitar la identificación de patrones, se elabora una tabla que resume la evolución del tamaño del problema, la complejidad de un nodo y la complejidad de un nivel.

Nivel i	Tamaño del problema	Complejidad (de un subproblema)	No. de suproblemas	Complejidad (de todo el nivel)
0	n	n-1	1	$1(n\text{-}1) = n\text{-}1$
1	$n/2$	$n/2$ -1	2	$2(n/2\text{ -}1) = n\text{-}2$
2	$n/4$	$n/4$ -1	4	$4(n/4\text{ -}1) = n\text{-}4$
⋮	⋮	⋮	⋮	⋮
$9i$	$\frac{n}{2^i}$	$\frac{n}{2^i}-1$	2^i	$n-2^i$
⋮	⋮	⋮	⋮	⋮
m	$\frac{n}{2^m}=1$	$T\left(\frac{n}{2^m}\right)=T(1)=0$	2^m	$2^m T\left(\frac{n}{2^m}\right)=2^m T(1)=2^m(0)$

Patrón de Complejidad: $n\text{-}2^i$

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

Para $i = m$, que es el tamaño de problema más pequeño, y aplicando: $b^L = x \;\; ; \;\; \log_b x = L$

$$T\left(\frac{n}{2^m}\right)=T(1)$$

$$\frac{n}{2^m}=1$$

$$2^m=n$$

$$m=\lg n$$

Logrando con ello establecer el límite al cual llegará i, es decir, la profundidad del árbol

Utilizando la serie geométrica $\sum_{i=0}^{k}2^i = 2^{k+1}-1$

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

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

$$T(n)=n\lg n-n+1$$

b) Determinación del orden asintótico

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

c) Comprobación con Derive

GEOMETRIC1 es una función que resuelve ecuaciones de recurrencia linear-geométrica obtenidas de algoritmos de los tipos divide-y-vencerás. La Función $T(n)$ se debe transformar para adquirir la forma esperada por GEOMETRIC1.

$$T(n)=n-1+2T\left(n/2\right),\quad T(1)=0$$

$$T(2n)=2T\left(n\right)+2n-1$$

$$y(kx)=p(x)y(x)+q(x)$$

$$k=2,\quad x=n,\quad p(x)=2,\quad q(x)=2n-1,\quad x_0=1,\quad y_0=0$$

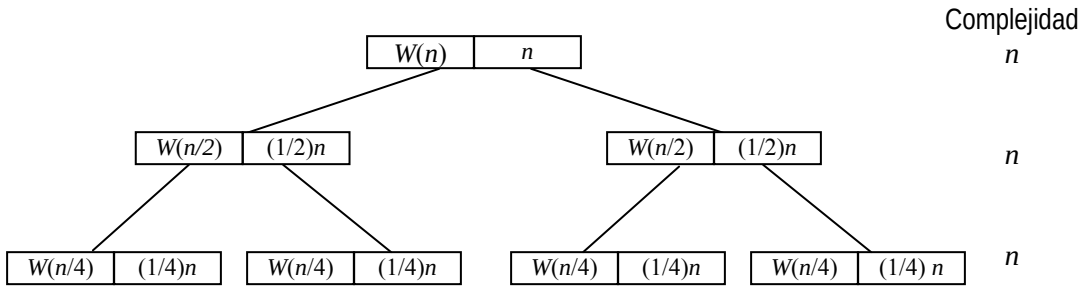
$$\text{GEOMETRIC1}(k,\ p(x),\ q(x),\ x,\ x_0,\ y_0)$$

```
#1: GEOMETRIC1(2, 2, 2n - 1, n, 1, 0)

                                 n·LN(n)
#2:                              ─────── - n + 1
                                 LN(2)
```

3.8.2. Utilice el método maestro para resolver la función W definida por la ecuación de recurrencia y caso base siguientes:

$$W(n) = 2W\left(\lfloor n/2 \rfloor\right) + n$$
$$W(1) = 0$$

a) Aplicación de métodos de solución de recurrencias



Observando el árbol de recurrencia se puede notar que el patrón de complejidad se mantiene constante, correspondiendo al caso 2 del método maestro.

De la formula general de divide-y-vencerás se tiene:

$$T(n) = bT(n/c) + f(n)$$
$$b = 2,\ c = 2,\ f(n) = n$$

Demostrar que $f(n) = \Theta(n^{\log_c b})$, requiere identificar las funciones $f(n)$ y $g(n)$

$$f(n) = n$$
$$g(n) = n^{\lg_2 2} = n$$

Como $f(n) = g(n)$, se demuestra que $f(n) = \Theta(n^{\log_c b})$

Del teorema maestro se obtiene la forma asintótica de la solución

$$W(n) = \Theta(n^{\log_c b} \lg n) = \Theta(n^{\log_2 2} \lg n) = \Theta(n \lg n)$$

b) Comprobación con Derive

$$W(n) = 2W\left(\lfloor n/2 \rfloor\right) + n, W(1) = 0$$
$$W(2n) = 2W\left(n\right) + 2n$$

$$y(kx) = p(x)y(x) + q(x)$$
$$k = 2,\ x = n,\ p(x) = 2,\ q(x) = 2n$$
$$\text{GEOMETRIC1}(k,\ p(x),\ q(x),\ x,\ x0,\ y0)$$

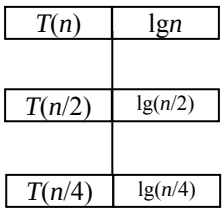
```
#1:   GEOMETRIC1(2, 2, 2n, n, 1, 0)

                                    n·LN(n)
#2:                                 ———————
                                     LN(2)
```

3.8.3. Utilice el método maestro y el árbol de recurrencia para resolver la función T definida por la ecuación de recurrencia y caso base siguientes:

$$T(n) = T(n/2) + \lg n, \quad T(1) = 1$$

a) Aplicación de métodos de solución de recurrencias

Del análisis del árbol de recurrencia se observa que la complejidad no es constante y no es una serie geométrica decreciente, por lo que no posible aplicar el Teorema maestro para resolver la recurrencia.



Complejidad
$\lg n$

$\lg(n/2)$

$\lg(n/4)$

$\lg(n/2^i)$

Nivel i	Tamaño del problema	Complejidad (de un subproblema)	No. de subproblemas	Complejidad (de todo el nivel)
0	n	$\lg n$	1	$\lg n$
1	$n/2$	$\lg(n/2)$	1	$(\lg(n/2))$
2	$n/4$	$\lg(n/4)$	1	$(\lg(n/4))$
⋮	⋮	⋮	1	⋮
i	$n/2^i$	$\lg(n/2^i)$	1	$\lg(n/2^i)$
⋮	⋮	⋮	1	⋮
m	$n/2^m=1$	$T(n/2^m)=T(1)=1$	1	$T(n/2^m)= T(1)=1$

Patrón de complejidad: $\lg\left(\frac{n}{2^i}\right)$

Para $i=m$, que es el tamaño de problema más pequeño, y aplicando: $b^L = x \;\; ; \;\; \log_b x = L$

$$T(n/2^m) = T(1)$$
$$\frac{n}{2^m} = 1$$
$$2^m = n$$
$$m = \lg n$$

Logrando con ello establecer el límite al cual llegará i, es decir la profundidad del árbol

$$T(n) = \sum_{i=0}^{\lg n-1} \lg\left(n/2^i\right) + T(1)$$
$$T(n) = \sum_{i=0}^{\lg n-1} \lg n - \sum_{i=0}^{\lg n-1} \lg 2^i + 1 = \lg n \sum_{i=0}^{\lg n-1} 1 - \sum_{i=0}^{\lg n-1} i + 1$$
$$T(n) = \lg n(\lg n) - \frac{\lg n(\lg n - 1)}{2} + 1 = (\lg n)^2 - \frac{(\lg n)^2 - \lg n}{2} + 1$$
$$T(n) = \frac{2(\lg n)^2 - (\lg n)^2 + \lg n}{2} + 1 = \frac{(\lg n)^2 + \lg n}{2} + 1$$

b) Determinación del orden asintótico

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

c) Comprobación con Derive

$$T(n) = T(n/2) + \lg n, \quad T(1) = 1$$
$$T(2n) = T\left(n\right) + \lg 2n$$

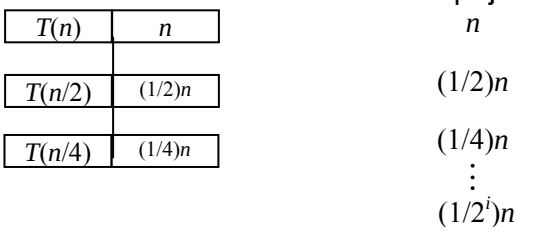
$$y(kx) = p(x)y(x) + q(x)$$
$$k = 2,\ x = n,\ p(x) = 1,\ q(x) = \lg 2n$$
$$\text{GEOMETRIC1}(k,\ p(x),\ q(x),\ x,\ x0,\ y0)$$

```
#1:   GEOMETRIC1(2, 1, LOG(2n, 2), n, 1, 1)

                                  2
                            LN(n)          LN(n)
#2:                         ──────── + ──────── + 1
                                 2      2·LN(2)
                            2·LN(2)
```

3.8.4. Utilice el método maestro para resolver la función T definida por la ecuación de recurrencia y caso base siguientes: $T(n) = T(n/2) + n, \quad T(1) = 1$

a) Aplicación de métodos de solución de recurrencias

Del análisis del árbol de recurrencia se observa que la complejidad es una serie geométrica decreciente, por lo que es recomendable aplicar el caso 3 del Teorema maestro.



Nivel i	Tamaño del problema	Complejidad (de un subproblema)	Complejidad (de todo el nivel)
0	n	n	n
1	$n/2$	$n/2$	$n/2$
2	$n/4$	$n/4$	$n/4$
⋮	⋮	⋮	⋮

De la formula general $T(n) = bT(n/c) + f(n)$ se tiene se tiene $b = 1,\ c = 2,\ f(n) = n$.

Demostrar que $f(n) = \Omega(n^{\log_c b+\varepsilon})$ para $\varepsilon > 0$, requiere identificar las funciones $f(n)$ y $g(n)$

$$f(n) = n$$

$$g(n) = n^{\log_c b+\varepsilon} = n^{\log_2 1+\varepsilon} = (n^{0+\varepsilon}) = (n^{\varepsilon})$$

Como $f(n/c) \le c' g\ (n)$ se demuestra que $f(n) = \Omega(n^{\varepsilon})$ para $\varepsilon = 1$ y $c' = 1$

El Teorema Maestro también pide demostrar que $bf(n/c) \le c'\, f(n)$ para $c' < 1$.

$$bf(n/c) \le n/2$$

$$bf(n) = \tfrac{2}{3}\, n \text{ para } c' = \tfrac{2}{3}$$

Como existe una $c' < 1$, se demuestra que $bf(n/c) \le c'\, f(n)$

Cuando las demostraciones son exitosas, el teorema maestro señala que $T(n) = \Theta\big(f(n)\big)$.

Por tanto, la forma asintótica de la solución queda:

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

b) Comprobación con Derive

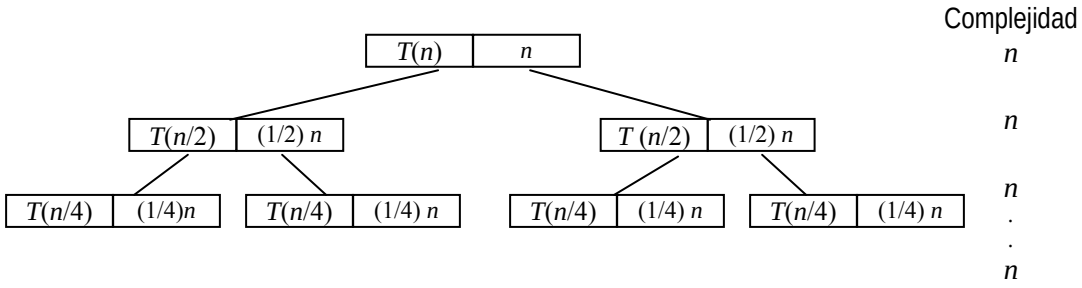
$$T(n) = T(n/2) + n, \quad T(1) = 1$$
$$T(2n) = T(n) + 2n$$

$$y(kx) = p(x)y(x) + q(x)$$
$$k = 2,\ x = n,\ p(x) = 1,\ q(x) = 2n$$
$$\text{GEOMETRIC1}(k,\ p(x),\ q(x),\ x,\ x0,\ y0)$$

```
#1:   GEOMETRIC1(2, 1, 2n, n, 1, 1)

#2:                              2·n - 1
```

3.8.5. Utilice el método maestro para resolver la función ***T*** definida por la ecuación de recurrencia y caso base siguientes: $T(n) = 2T(n/2) + n, \quad T(1) = 1$

a) Aplicación de métodos de solución de recurrencias



Nivel i	Tamaño del problema	Complejidad (de un subproblema)	Complejidad (de todo el nivel)
0	n	n	$n = n$
1	$n/2$	$n/2$	$2(n/2) = n$
2	$n/4$	$n/4$	$4(n/4) = n$
⋮	⋮	⋮	⋮

Observando el árbol de recurrencia se puede notar que el patrón de complejidad se mantiene constante, correspondiendo al caso 2 del método maestro.

De la formula general de divide-y-vencerás $T(n) = bT(n/c) + f(n)$ se tiene
$b = 2,\ c = 2,\ f(n) = n$

Demostrar que $f(n) = \Theta(n^{\log_c b})$, requiere identificar las funciones $f(n)$ y $g(n)$

$$f(n) = n, \quad g(n) = n^{\lg_2 2} = n$$

Como $f(n) = g(n)$,se demuestra que $f(n) = \Theta(n^{\log_c b})$

Del teorema maestro se obtiene la forma asintótica de la solución

$$W(n) = \Theta(n^{\log_c b} \lg n) = \Theta(n^{\log_2 2} \lg n) = \Theta(n \lg n)$$

b) Comprobación con Derive

$T(n) = 2T(n/2) + n, \quad T(1) = 1$

$T(2n) = 2T(n) + 2n$

$y(kx) = p(x)y(x) + q(x)$

$k = 2, x = n, p(x) = 2, q(x) = 2n$

GEOMETRIC1(k, p, q, x, $x0$, $y0$)

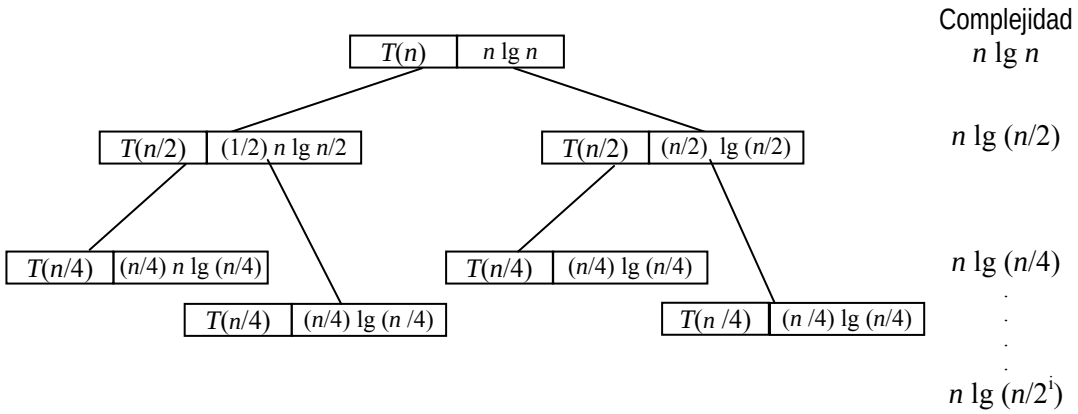
```
#1:   GEOMETRIC1(2, 2, 2n, n, 1, 1)

                                  n·LN(n)
#2:                               ─────── + n
                                  LN(2)
```

3.8.6. Utilice el método maestro y el árbol de recurrencia para resolver la función T definida por la ecuación de recurrencia y caso base siguientes:

$T(n) = 2T(n/2) + n \lg n, \quad T(1) = 1$

a) Aplicación de métodos de solución de recurrencias

Del análisis del árbol de recurrencia se observa que la complejidad no se mantiene constante y no es una serie geométrica decreciente, por lo que no es posible utilizar el Teorema maestro para la solución de la recurrencia



Nivel i	Tamaño del problema	Complejidad (de un subproblema)	No. de subproblemas	Complejidad (de todo el nivel)
0	n	$n \lg n$	2	$1(n \lg n) = n \lg(n)$
1	$n/2$	$(n/2)\lg(n/2)$	4	$2((n/2) \lg(n/2)) = n \lg(n/2)$
2	$n/4$	$(n/4)\lg(n/4)$	8	$4((n/4) \lg(n/4)) = n \lg(n/4)$
⋮	⋮	⋮		⋮
i	$n/2^i$	$(n/2^i)\lg(n/2^i)$	2^i	$n \lg(n/2^i)$
⋮	⋮	⋮		⋮
m	$n/2^m$	$T(n/2^m)=T(1)=1$	2^m	$2^m\ T(n/2^m)= 2^m\ T(1)$

Patrón de complejidad: $n \lg\left(\frac{n}{2^i}\right)$

Para $i = m$, que es el tamaño de problema más pequeño, y aplicando: $b^L = x \ ; \ \log_b x = L$

$$T(n/2^m) = T(1)$$

$$\frac{n}{2^m} = 1$$

$$2^m = n$$

$$m = \lg n$$

Logrando con ello establecer el límite al cual llegará i, es decir, la profundidad del árbol

$$T(n) = \sum_{i=0}^{m-1} n \lg\left(\frac{n}{2^i}\right) + 2^m T(1) = \sum_{i=0}^{\lg n-1} n \lg\left(\frac{n}{2^i}\right) + 2^{\log n} T(1)$$

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

$$T(n) = n \lg n(\lg n) - n \frac{\lg n(\lg n - 1)}{2} + n$$

$$T(n) = n \lg^2 n - \frac{1}{2} n \lg^2 n + \frac{1}{2} n \lg n + n$$

$$T(n) = \frac{1}{2} n \lg^2 n + \frac{1}{2} n \lg n + n$$

b) Determinación del orden asintótico

$$T\left(n\right) = \Theta\left(n \log^2 n\right)$$

c) Comprobación con Derive

$$T(n) = 2T(n/2) + n\lg n, \quad T(1) = 1$$
$$T(2n) = 2T(n) + 2n(\lg 2n)$$

$$y(kx) = p(x)y(x) + q(x)$$
$$k = 2,\ x = n,\ p(x) = 2,\ q(x) = 2n\lg 2n$$
$$\text{GEOMETRIC1}(k,\ p(x),\ q(x),\ x,\ x0,\ y0)$$

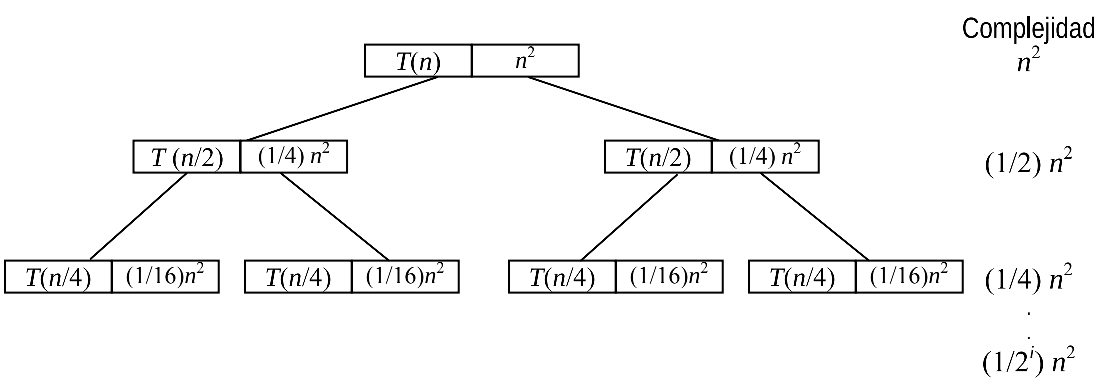
```
#1:  GEOMETRIC1(2, 2, 2nLOG(2n, 2), n, 1, 1)

                                2
                         n·LN(n)           n·LN(n)
#2:                     ───────────── + ───────────── + n
                                  2        2·LN(2)
                           2·LN(2)
```

3.8.7. Utilice el método maestro y el árbol de recurrencia para resolver la función *T* definida por la ecuación de recurrencia y caso base siguientes:

$$T(n) = 2T(n/2) + n^2, \quad T(1) = 1$$

a) Aplicación de métodos de solución de recurrencias

Del análisis del árbol de recurrencia se observa que la complejidad es una serie geométrica decreciente, por lo que para la solución de la recurrencia se utilizará el caso 3 del Teorema Maestro.



Nivel i	Tamaño del problema	Complejidad (de un subproblema)	Complejidad (de todo el nivel)
0	n	n^2	$1\ n^2 = (1/1)\ n^2$
1	$n/2$	$(n/2)^2$	$2\ (n/2)^2 = (1/2)\ n^2$
2	$n/4$	$(n/4)^2$	$4\ (n/4)^2 = (1/4)\ n^2$
3	$n/8$	$(n/8)^2$	$8\ (n/8)^2 = (1/8)\ n^2$
⋮	⋮	⋮	⋮

De la formula general $T(n) = bT(n/c) + f(n)$ se tiene $b = 2, \quad c = 2, \quad f(n) = n^2$

Demostrar que $f(n) = \Omega(n^{\log_c b+\varepsilon})$ para $\varepsilon > 0$, requiere identificar las funciones $f(n)$ y $g(n)$

$$f(n) = n^2$$
$$g(n) = (n^{\log_c b+\varepsilon}) = (n^{1+\varepsilon})$$

Como $f(n) \le c' g(n)$, se demuestra que $f(n) = \Omega(n^{1+\varepsilon})$ para $\varepsilon = 1$ y $c' = 1$

El Teorema Maestro también pide demostrar que $bf(n) \le c' f(n)$ para $c' < 1$.

$$bf\left(n/c\right) = 2\left(n/2\right)^2 = n^2/2$$
$$c' f\left(n\right) = \left(2/3\right)n^2 \quad \text{para } c' = 2/3$$

Como existe una $c' < 1$, se demuestra que $bf(n) \le c' f(n)$

Cuando las demostraciones son exitosas, el teorema maestro señala que $T\left(n\right) = \Theta\left(f\left(n\right)\right)$. Por tanto, la forma asintótica de la solución queda:

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

b) Comprobación con Derive

$$T(n) = 2T(n/2) + n^2, \quad T(1) = 1$$
$$T(2n) = 2T\left(n\right) + 4n^2$$

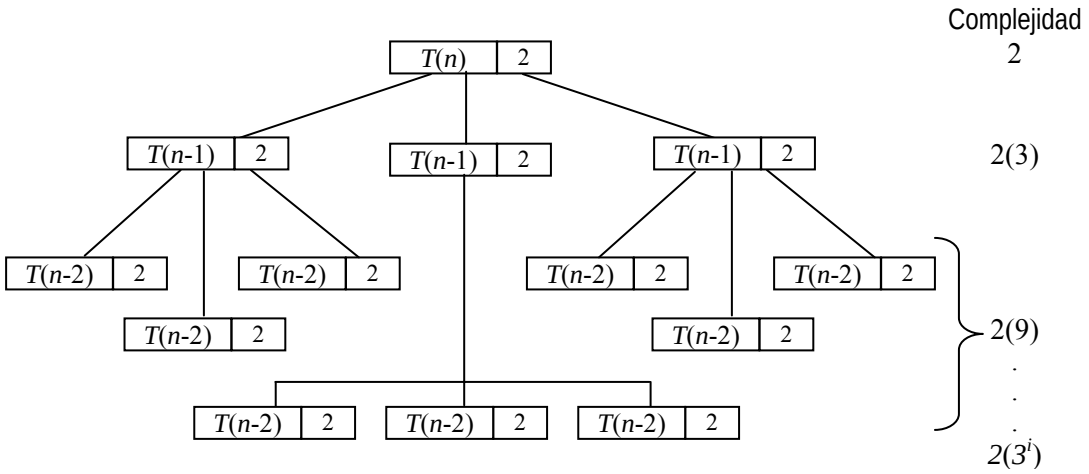
$$y(kx) = p(x)y(x) + q(x)$$
$$k = 2, x = n, \ p(x) = 2, q(x) = 4n^2$$
$$\text{GEOMETRIC1}(k,\ p(x),\ q(x),\ x,\ x0,\ y0)$$

```
#1:  GEOMETRIC1(2, 2, 4n^2 , n, 1, 1)
                                  2
#2:                          2·n – n
```

3.8.8. Utilice el método de árbol de recurrencia para resolver la función:

$$T(1) = 1,\ T(n) = 3T(n-1) + 2$$

a) Aplicación de métodos de solución de recurrencias



Nivel i	Tamaño del problema	Complejidad (de un subproblema)	No. de subproblemas	Complejidad (de todo el nivel)
0	n-1	2	3	3(2)
1	n-2	2	9	9(2)
2	n-3	2	27	27(2)
⋮	⋮	⋮		⋮
i	n-i	2	3^i	3^i (2)
⋮	⋮	⋮		⋮
m	n-m	T(n-m)=T(1)=1	3^m	$3^m T(n\text{-m})=3^m\ T(1) =3^m(1)$

Patrón de Complejidad: $2(3^i)$

$$T(n) = 2\sum_{i=0}^{m-1} 3^i + 3^m(1)$$

Para $i=m$, que es el tamaño de problema más pequeño, y aplicando: $b^L = x \ ;\ \log_b x = L$

$T(n\text{-}m)=T(1)$

$n\text{-}m=1$

$m = n - 1$

Logrando con ello establecer el límite al cual llegará i, es decir, la profundidad del árbol

$$T(n)=2\sum_{i=0}^{n-2}3^i+3^{n-1}T(1)=2\left(\frac{3^{n-2+1}-1}{3-1}\right)+3^{n-1}=2\left(\frac{3^{n-1}-1}{2}\right)+3^{n-1}$$
$$T(n)=3^{n-1}-1+3^{n-1}=2\left(3^{n-1}\right)-1=\Theta(3^n)$$

b) Comprobación con Derive

LIN1_DIFFERENCE es una función que resuelve ecuaciones de recurrencia de primer orden obtenidas de algoritmos de los tipos recorta-y-vencerás. La Función $T(n)$ se debe transformar para adquirir la forma esperada por LIN1_DIFFERENCE.

$T(n) = 3T(n-1) + 2, \quad T(1)=1$
$T(n+1)=3T(n)+2$

$y(x+1)=p(x)y(x)+q(x)$
$x=n,\ p(x)=3,\ q(x)=2$
$\mathrm{LIN1\_DIFFERENCE}(p(x),\ q(x),\ x,\ x0,\ y0)$

```
#1:   LIN1_DIFFERENCE(3, 2, n, 1, 1)

                                    n - 1
#2:                              2·3      - 1
```

3.8.9. Utilice el método de expansión para resolver la siguiente recurrencia:

$T(1)=8$, $T(n)=3T(n-1)-15$

a) Aplicación de métodos de solución de recurrencias

La ecuación inductiva se expresa con diferentes tamaños de problema

$T(n) = 3T(n-1)-15$
$T(n-1)=3T(n-2)-15$
$T(n-2)=3T(n-3)-15$

La ecuación inductiva se expande

Expansión con $T(n)$
$T(n)=3T(n-1)-15$

Expansión con $T(n-1)$
$T(n)=3\left(3T(n-2)-15\right)-15=(3)(3)T(n-2)-3(15)-15=-15-3(15)+3^2T(n-2)$

Expansión con $T(n-2)$

$$T(n)=(3)(3)\Big(\big(3T(n-2)-15\big)\Big)-(3)15-15=-15-3(15)-3^2(15)+3^3T(n-3)$$

La ecuación inductiva se expresa con sumatorias

$$T(n)=-15\sum_{i=0}^{m-1}3^i+3^mT(n-m)$$

En el caso base, $T(1)=8$, por lo tanto $n-m=1$ y $m=n-1$

$$T(n)=-15\sum_{i=0}^{(n-1)-1}3^i+3^{n-1}T(1)$$

$$T(n)=-15\sum_{i=0}^{n-2}3^i+3^{n-1}T(1)$$

$$T(n)=-(15)\left(\frac{3^{n-1}-1}{2}\right)+(3^{n-1})(8)$$

$$T(n)=3^{n-1}\left(8-\tfrac{15}{2}\right)+\tfrac{15}{2}$$

$$\boxed{T(n)=\frac{3^{n-1}+15}{2}}$$

b) Comprobación con Derive

$$T(n)=3T(n\text{-}1)\text{-}15,\quad T(1)=8$$

$$T(n+1)=3T\left(n\right)-15$$

$$y(x+1)=p(x)y(x)+q(x)$$

$$x=n,\ p(x)=3,\ q(x)=-15$$

$$\mathrm{LIN1\_DIFFERENCE}(p(x),\ q(x),\ x,\ x0,\ y0)$$

```
#1:   LIN1_DIFFERENCE(3, -15, n, 1, 8)

                                  n - 1
                                 3          15
#2:                              ------  +  --
                                   2         2
```

3.8.10. Utilice el método de expansión para resolver la siguiente recurrencia:

$$T(1)=2,\ T(n)=T(n-1)+n-1$$

a) Aplicación de métodos de solución de recurrencias

La ecuación inductiva se expresa con diferentes tamaños de problema

$$T(n) = T(n-1) + n - 1$$
$$T(n-1) = T(n-2) + n - 2$$
$$T(n-2) = T(n-3) + n - 3$$

La ecuación inductiva se expande

Expansión con $T(n)$

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

Expansión con $T(n-1)$

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

Expansión con $T(n-2)$

$$T(n) = (n-1) + (n-2) + (T(n-3) + n - 3) = (n-1) + (n-2) + (n-3) + T(n-3)$$

La ecuación inductiva se expresa con sumatorias

$$T(n) = \sum_{i=1}^{m} (n-i) + T(n-m)$$

En el caso base, $T(1) = 2$, por lo tanto $n - m = 1$ y $m = n - 1$

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

b) Comprobación con Derive

$$T(n) = T(n-1) + n - 1, \quad T(1) = 2$$
$$T(n+1) = T(n) + n$$

$$y(x+1) = p(x)y(x) + q(x)$$
$$x = n,\ p(x) = 1,\ q(x) = n$$
$$\text{LIN1\_DIFFERENCE}(p(x),\ q(x),\ x,\ x0,\ y0)$$

```
#1:   LIN1_DIFFERENCE(1, n, n, 1, 2)

                                    2
                                   n       n
#2:                                ──  -  ──  + 2
                                   2       2
```

3.9 Ejercicios propuestos

3.9.1 Usando la notación vista en el curso de Análisis y Diseño de Algoritmos II, escriba la definición de cada tipo de función de recurrencia de la lista de abajo. Para cada tipo de función, también presente un ejemplo y su solución con alguno de los métodos vistos en el curso de análisis y diseño de Algoritmos I.

a) **Recurrencia homogénea lineal de primer-orden con coeficientes constantes**
b) **Recurrencia homogénea lineal de segundo-orden con coeficientes constantes**
c) **Recurrencia no-homogénea de primer-orden**
d) **Recurrencia no-homogénea de segundo-orden**
e) **Recurrencia no-lineal**

3.9.2 Use el árbol de recurrencia para resolver las recurrencias de los ejercicios 3.8.2 y 3.8.4, 3.8.5, 3.8.7

3.9.3 Resuelva las siguientes recurrencias usando el método maestro, árbol de recurrencia y expansión. Verifique la solución con la herramienta Derive.

$$T(n)=\begin{cases}3T(n/2)+n^2+n & n\geq 2\\ 1 & n=1\end{cases}$$

3.9.4 Resuelva las siguientes recurrencias usando el método maestro, árbol de recurrencia y expansión. Verifique la solución con la herramienta Derive.

$$T(n)=\begin{cases}T(n-2)+3n+4 & n\geq 3\\ 1 & n=1\\ 6 & n=2\end{cases}$$

3.9.5 Encuentre un ejemplo de recurrencia de solución conocida en el que las siguientes reglas maestras obtienen la misma solución que las reglas utilizadas en los ejercicios resueltos.

Para recurrencias con la forma general $T(n)=bT(n/c)+an^k$, la solución de la recurrencia se obtiene con las siguientes reglas:

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } b < c^k \\ \Theta(n^k \log n) & \text{si } b = c^k \\ \Theta(n^{\log_c b}) & \text{si } b > c^k \end{cases}$$

Bibliografía

1. CormenT.H., LeisersonC.E., RivestR.L: Introduction to Algorithms, MIT Press 1990.
2. Sara Basse, Allen Van Gelder. Algoritmos Computacionales: Introducción al análisis y diseño. Tercera Edición. AddisonWesley. 2002.
3. Ian Parberry, William Gasarch. Problems on Algorithms. Prentice Hall, 1994