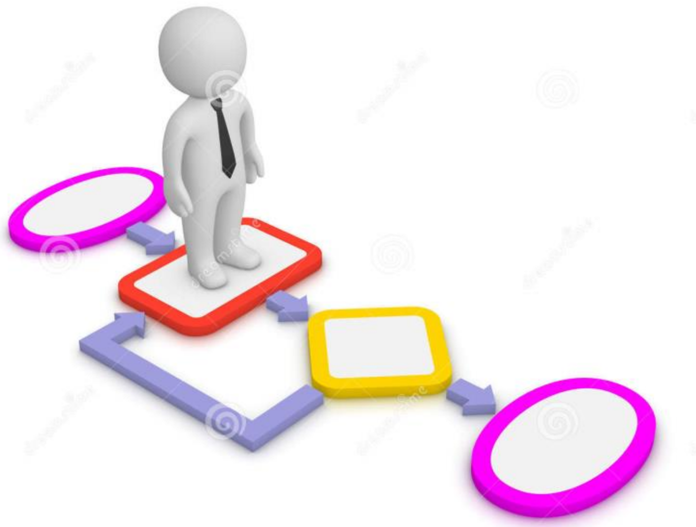


Design & Analysis of Algorithms

Saman Riaz (PhD)
Associate Professor
RSCI, Lahore

Lecture # 05

RECURRENCE

Recurrences - Definition

A ***recurrence is a relation or equation*** that describes a ***function in terms of its lower order arguments***, with the following characteristics

(i) The function is defined over a set of ***natural numbers***

(ii) The definition includes a ***base*** value for the function, called ***boundary condition***

Example(1): The factorial function ***f(n)=n!*** can be *expressed*
By the recurrence

$f(n) = n.f(n-1)$

$f(0)=1$ (boundary condition)

Example(2) The ***Fibonacci Sequence f(n)*** is usually defined as

$f(n) = f(n\text{-}1) + f(n\text{-}2)$

$f(0) = 0, \quad f(1)=1$ (boundary condition)

Recurrences - Examples

The following examples demonstrate the use of recurrence for the running times of common algorithms. Here $T(n)$ denotes the running time for a problem of size n.

Example(1): Here is an example of recurrence relation for ***decrease- and-conquer*** problem.

$$T(n) = T(\underbrace{n-1}_{\text{Subproblem size}}) + \underbrace{cn}_{\text{Cost of decreasing}}$$

Example(2) This example illustrates the recurrence relation for ***divide-and-conquer*** problem.

$$T(n) = \underbrace{8}_{\text{Number of subproblems}} T(n / \underbrace{4}_{\text{Subproblem size}}) + \underbrace{c\, n^2}_{\text{Cost of dividing and combining}}$$

Recurrence Examples

$$s(n)=\begin{cases}0 & n=0\\ c+s(n-1) & n>0\end{cases}$$

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

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

$$T(n)=\begin{cases}c & n=1\\ aT\left(\frac{n}{b}\right)+cn & n>1\end{cases}$$

Methods to solve recurrence

- Substitution method
- Iteration method
- Recursion tree method
- Master Theorem

SUBSTITUTION METHOD

Substitution

The substitution method

- the “making a good guess method”
- Guess the form of the answer, then use induction to find the constants and show that solution works

- Comes with experience

Problem:

Determine a tight asymptotic lower bound for the following recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2.$$

Let us guess that $T(n) = n^2 \lg(n)$. Then our induction hypothesis is that there exists a c and an n_0 such that

$$T(n) \geq cn^2 \lg(n) \; \forall n > n_0 \text{ and } c > 0.$$

For the base case $(n = 1)$, we have $T(1) = 1 > c1^2 \lg 1$. This is true for all $c > 0$.

Now, for the inductive step, assume the hypothesis is true for $m < n$. Then

$$T(m) \geq cm^2 \lg(m).$$

Problem:

So,

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n^2 \\ &\geq 4c\frac{n^2}{4}\lg\frac{n}{2} + n^2 \\ &= cn^2\lg(n) - cn^2\lg(2) + n^2 \\ &= cn^2\lg(n) + (1-c)n^2. \end{aligned}$$

If we now pick as $c < 1$, then

$$T(n) = \Omega(cn^2\lg(n)). \square$$

ITERATION METHOD

Iteration Method

In the ***iteration method*** the recurrence is solved by following the ***top-down approach***.
It involves following steps:

(1) Using definition, equations are set up for arguments n, n-1, n-2.....

(2) On reaching the bottom level the boundary condition is applied.

(3)The equations are summed up.

(4) Finally, the solution is obtained by canceling out identical terms on the left-hand and right- hand sides of the iterated equations

- The iteration method is particularly useful in solving ***decrease-and-conquer*** problems.

In other cases additional efforts are required to cancel out the terms appearing on both sides of the final equation

Iteration Method

Example(1): Here is a recurrence for the linear search. It is based on decrease-and-conquer algorithm:

$$T(0)=0$$

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

Iterating the recurrence:.

$$\begin{aligned}
T(n) &= T(n-1) + c\\
T(n-1) &= T(n-2) + c\\
T(n-2) &= T(n-3) + c\\
&\dots\dots\dots\dots\dots\dots\dots\dots\\
T(3) &= T(2) + c\\
T(2) &= T(1) + c\\
T(1) &= T(0) + c
\end{aligned}$$

Adding both sides of the equations, and canceling equal terms:

$$T(n) = c+ c + \dots\dots\dots + c$$

$Or,\ T(n)=n.c$

It follows that $T(n)=\theta(n)$

Iteration Method

Example(2): In ***selection sort***, the largest element in an array is searched. It is exchanged with the last element in the array. This procedure is repeatedly applied to sub-arrays. The recurrence for selection sort algorithm is as follows:

$$T(0)=0$$

$$\underbrace{T(n)}_{\text{Sorting } n \text{ elements}} = \underbrace{T(n-1)}_{\text{Sorting } n-1 \text{ elements}} + \underbrace{c.n}_{\text{Finding maximum and exchanging}}$$

Iterating the recurrence:

$$\begin{aligned}
T(n) &= T(n-1\ ) + c.n \\
T(n-1) &= T(n-2) + c.(n-1) \\
T(n-2) &= T(n-3) + c.(c-2) \\
&\dots \dots \dots \dots \dots \dots \dots \dots \\
T(3) &= T(2) + c.3 \\
T(2) &= T(1) + c.2 \\
T(1) &= T(0) + c.1
\end{aligned}$$

Adding both sides of the equations, and canceling equal terms:

$$T(n) = c(1 + 2 + 3 + \dots \dots \dots + n)$$

Summing the arithmetic series:

$$T(n)=c.n(n+1)/2$$

It follows that

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

Example 1: Iteration Method

$$s(n)=\begin{cases}0 & n=0\\ c+s(n-1) & n>0\end{cases}$$

- s(n) =

  c + s(n-1)

  c + c + s(n-2)

  2c + s(n-2)

  2c + c + s(n-3)

  3c + s(n-3)

  …

  kc + s(n-k) = ck + s(n-k)

Example 1: Iteration Method

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have
  - s(n) = ck + s(n-k)
- What if k = n?

Example 2: Iteration Method

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

- s(n)

=n + s(n-1)

=n + n-1 + s(n-2)

=n + n-1 + n-2 + s(n-3)

=n + n-1 + n-2 + n-3 + s(n-4)

=…

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

Example 2: Iteration Method

- s(n)

=n + s(n-1)

=n + n-1 + s(n-2)

=n + n-1 + n-2 + s(n-3)

=n + n-1 + n-2 + n-3 + s(n-4)

=…

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$= \sum_{i=n-k+1}^{n} i + s(n-k)$$

Example 2: Iteration Method

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

Example 2: Iteration Method

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

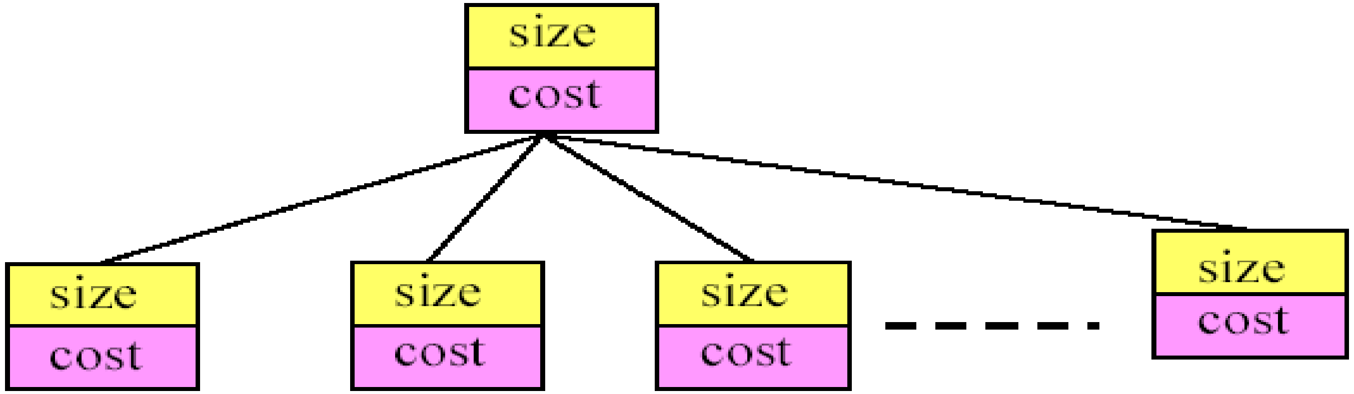
$$\sum_{i=1}^{n} i \quad + \quad s(0) \quad = \quad \sum_{i=1}^{n} i \quad + \quad 0 \quad = \quad n\frac{n+1}{2}$$

$$s(n) \quad = \quad n\frac{n+1}{2}$$

RECURSION TREE

The ***recursion tree*** provides a visual tool for solving recursive equation. It involves following steps

Step # 1 The recurrence is expressed in a ***hierarchical way using a tree structure***, such that each node contains two fields: the ***size field*** and ***cost field*** . The ***number of child nodes*** equals the ***number of subproblems***



Step #2: The ***size field*** of a node is set by plugging the the size of parent node into the relation

Step # 3 :The ***cost field*** is set by substituting node size into ***cost function of the relation***

Step #4: *The solution is found by* ***summing the costs over all nodes*** *of the tree*

Recurrence Relation

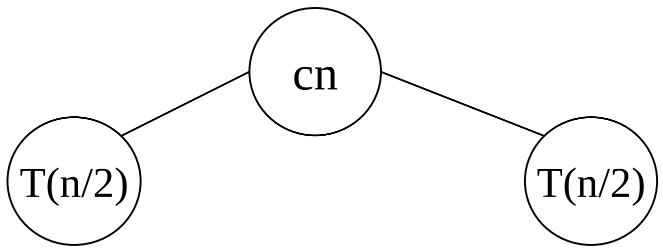
- Recall for Divide and Conquer algorithms

  $T(n) = aT(n/b) + D(n) + C(n)$

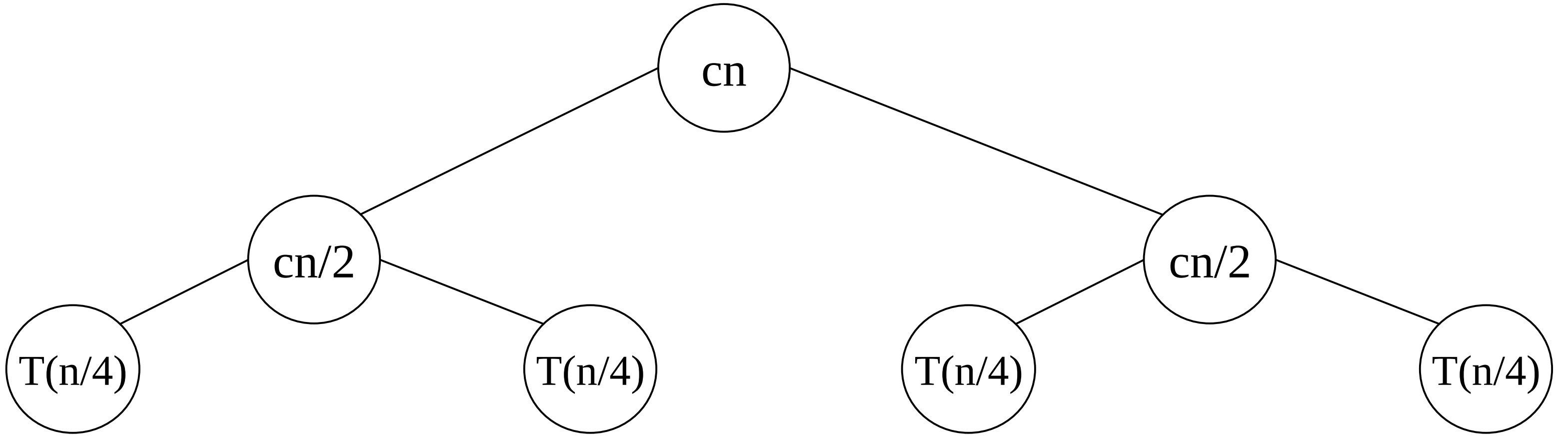
- Here $a=2$, and if we assume n is a power of 2, then each divide step leads to sub-arrays of size $n/2$
- $D(n)=\theta(1)$
- $C(n)= \theta(n)$

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

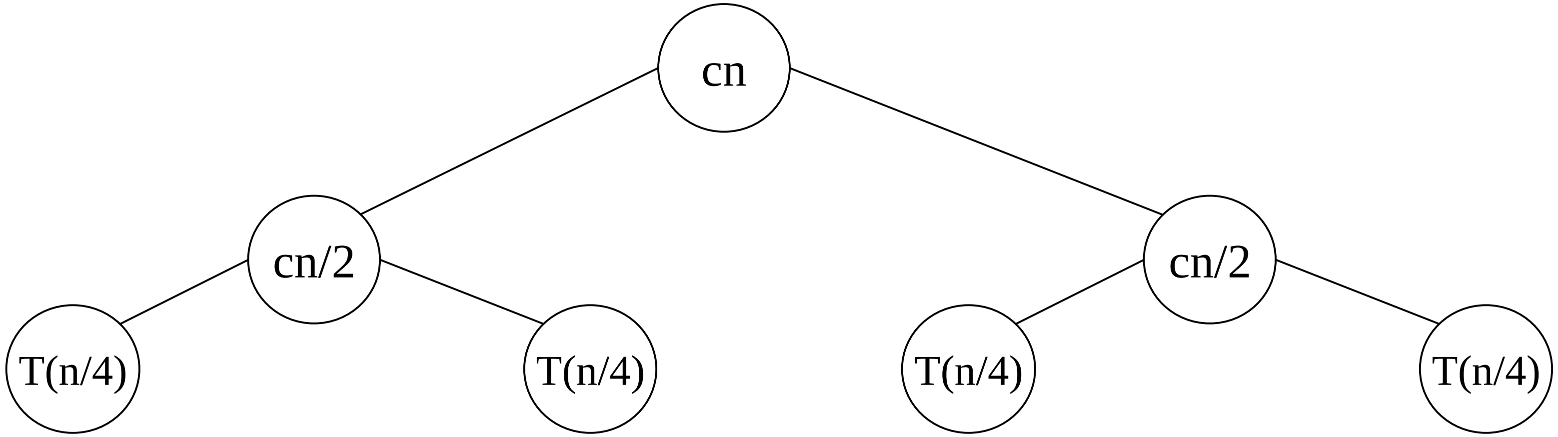
Recursion Tree for Merge Sort



Recursion Tree for Merge Sort

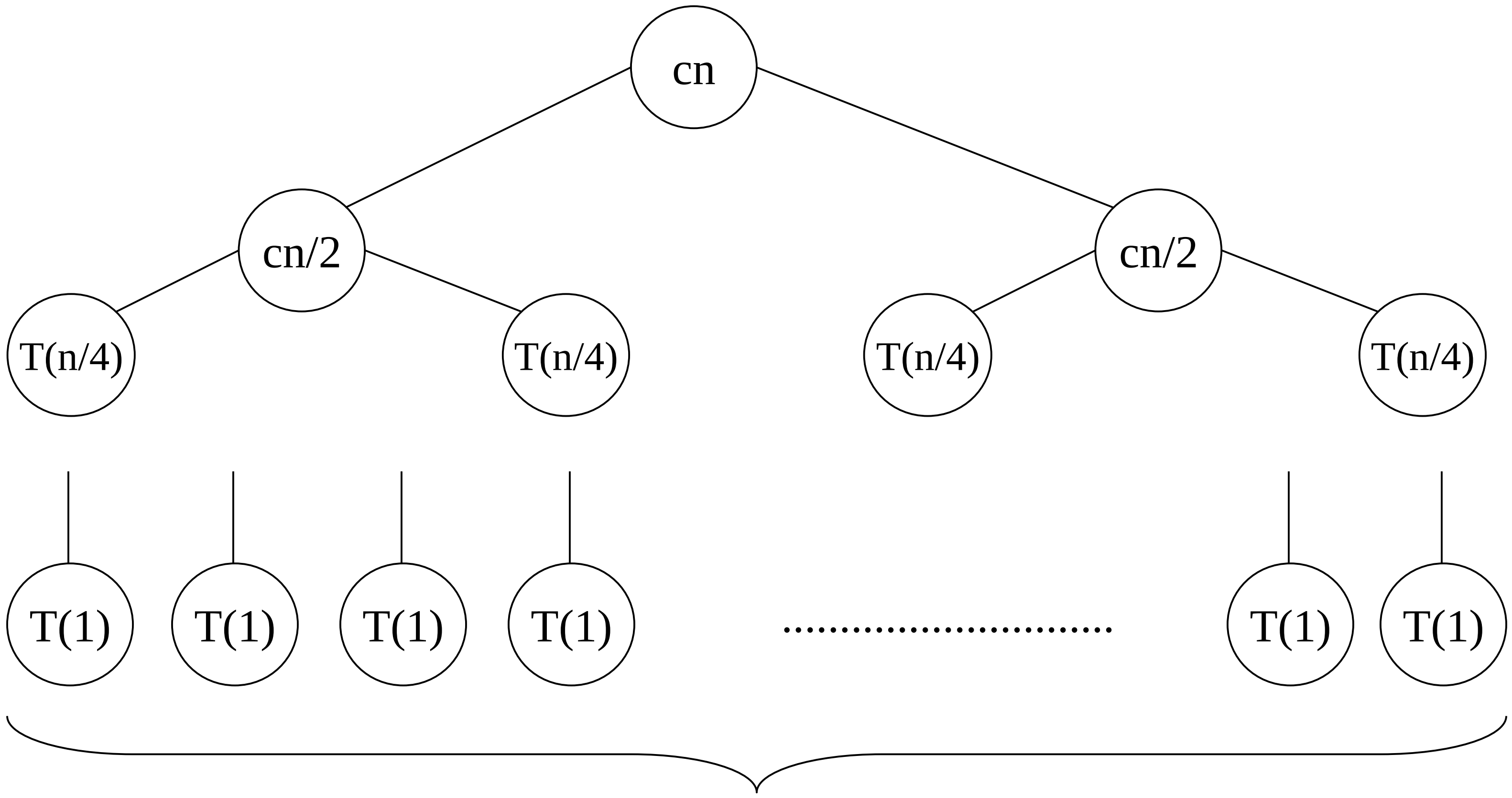


Recursion Tree for Merge Sort



Eventually, the input size (the argument of T) goes to 1, so...

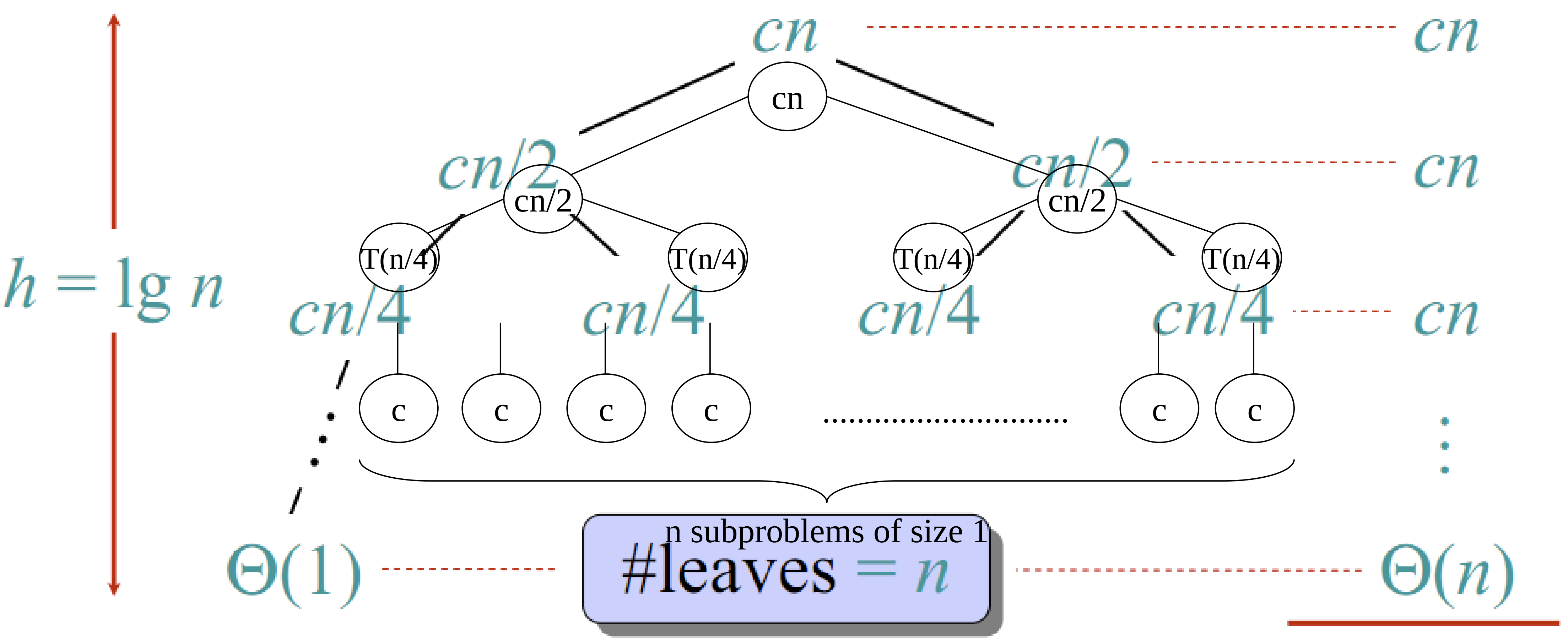
Recursion Tree for Merge Sort



n sub problems of size 1, but T(1) = c
by boundary condition

Recursion Tree for Merge Sort

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



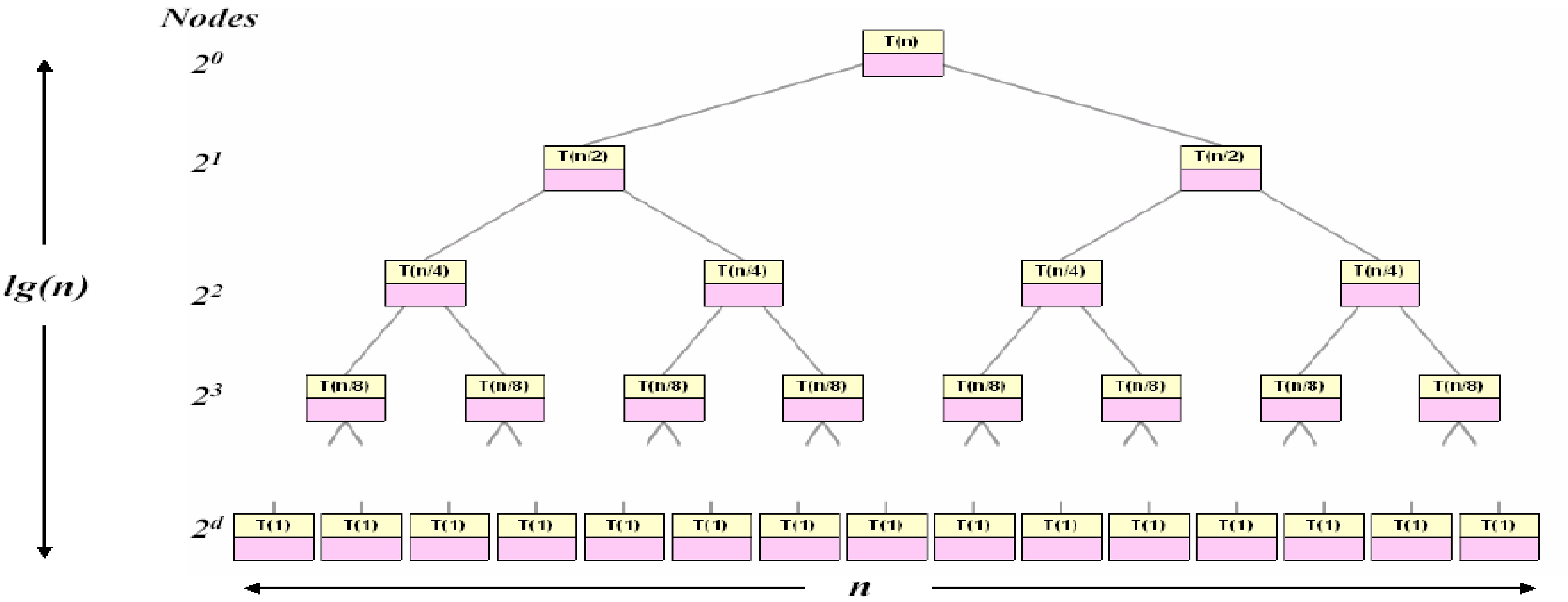
Total $= \Theta(n \lg n)$

Recursion Tree

Example(1): $T(n)= 2T(n/2) + cn,\ n>1, \quad T(1)=c$

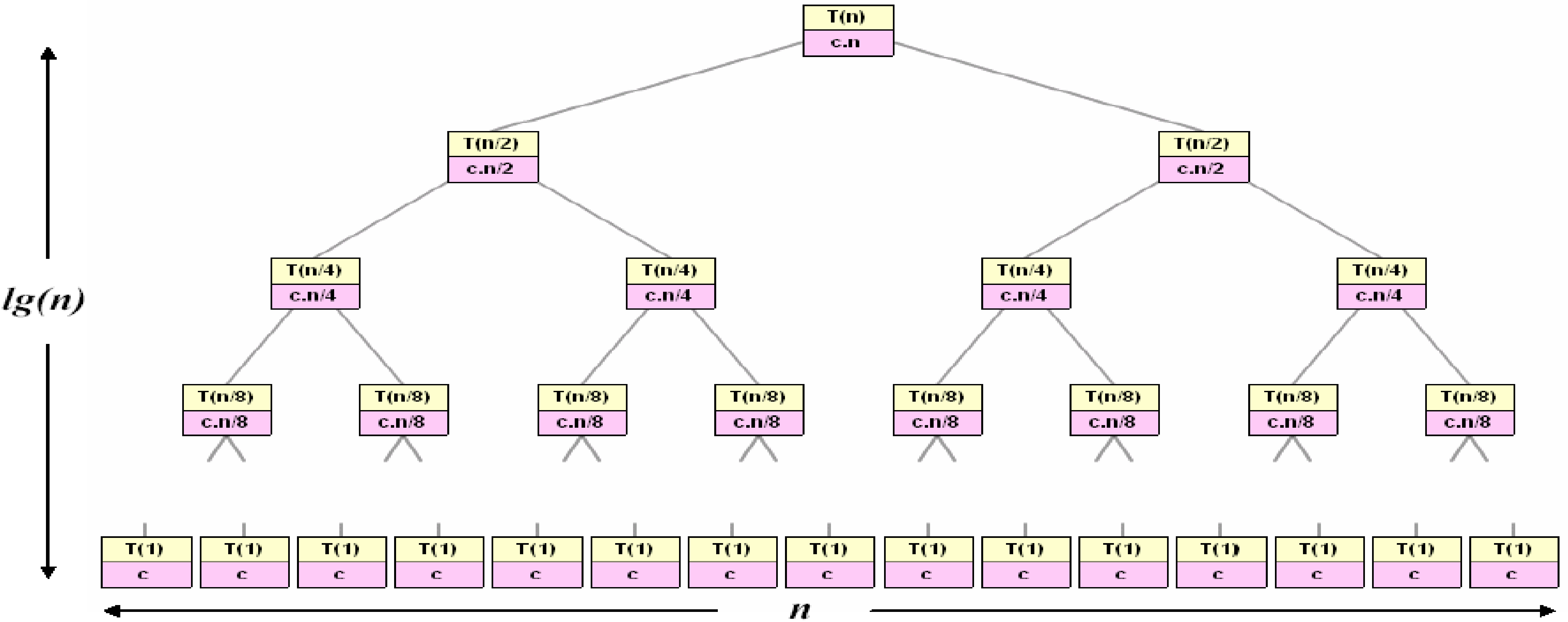
Step #1: Constructing tree structure

The fully expanded recursion tree is shown below. It has 2^d nodes at the bottom level (called ***leaves***) where ***d*** is the ***tree depth.*** Since at the bottom level $T(n/2^d)=T(1)$, it follows that $n/2^d=1$, or $2^d= n$. i.e $d= \lg n$.
Thus, ***tree depth*** $= \lg n$. , and ***number of leaves is*** $2^d=n$



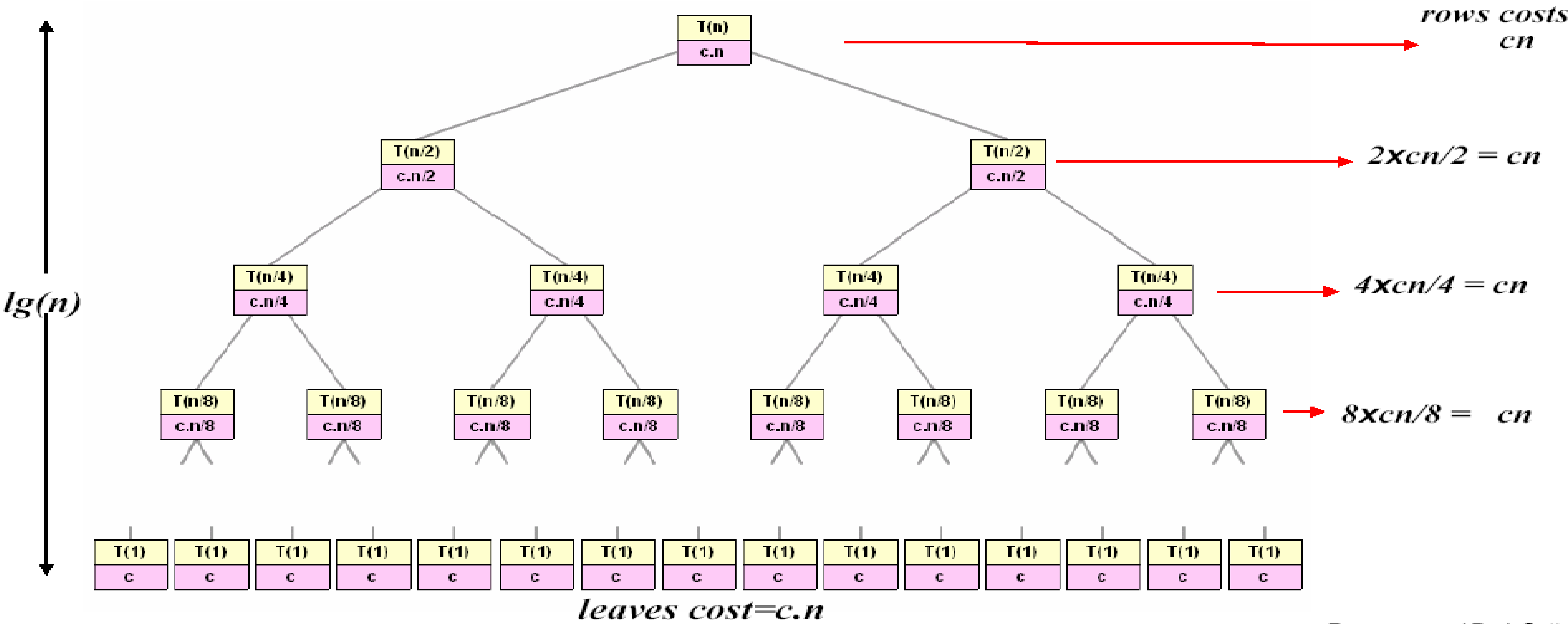
Recursion Tree

The root has associated size n and cost cn. Each child of root has size $n/2$ and associated cost $cn/2$
At the next level the costs are reduced by a factor of 2. This reduction is continued up to the bottom level.
Each leaf has associated cost of c.



Recursion Tree

Each row contributes total cost cn. Since there are ***lg n-1 rows of internal nodes*** and ***one root node***, total cos associated with all nodes is ***cn.(lg n-1)+cn=c.nlg n***. There are ***n leaves***, each having cost c. Thus, total contribution of leaves is ***c.n*** Hence, the recurrence has the solution ***T(n)=cn.lg n + cn =θ(n lg n)***

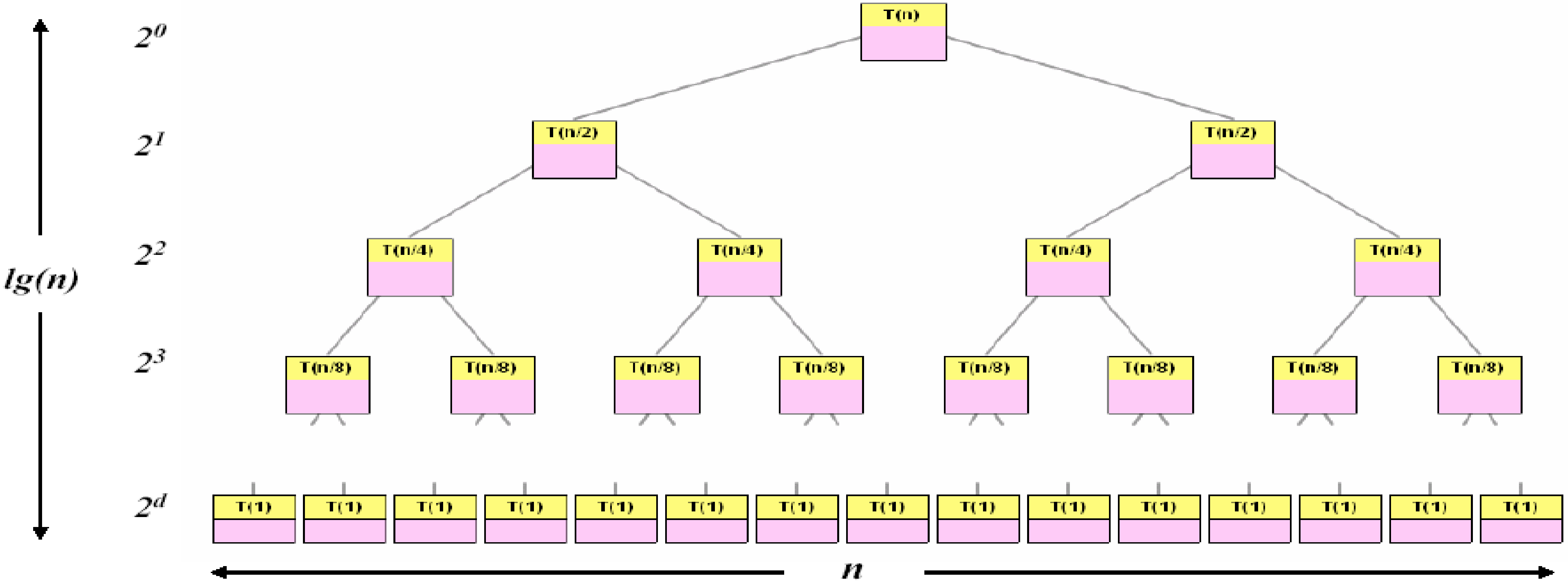


Recursion Tree

Example(2): $T(n) = 2T(n/2) + cn^2,\ n>1,\quad T(1)=c$

Step #1: Constructing tree structure

The fully expanded binary recursion tree is shown below. Tree has depth ***lg n***, and ***n leaves.***

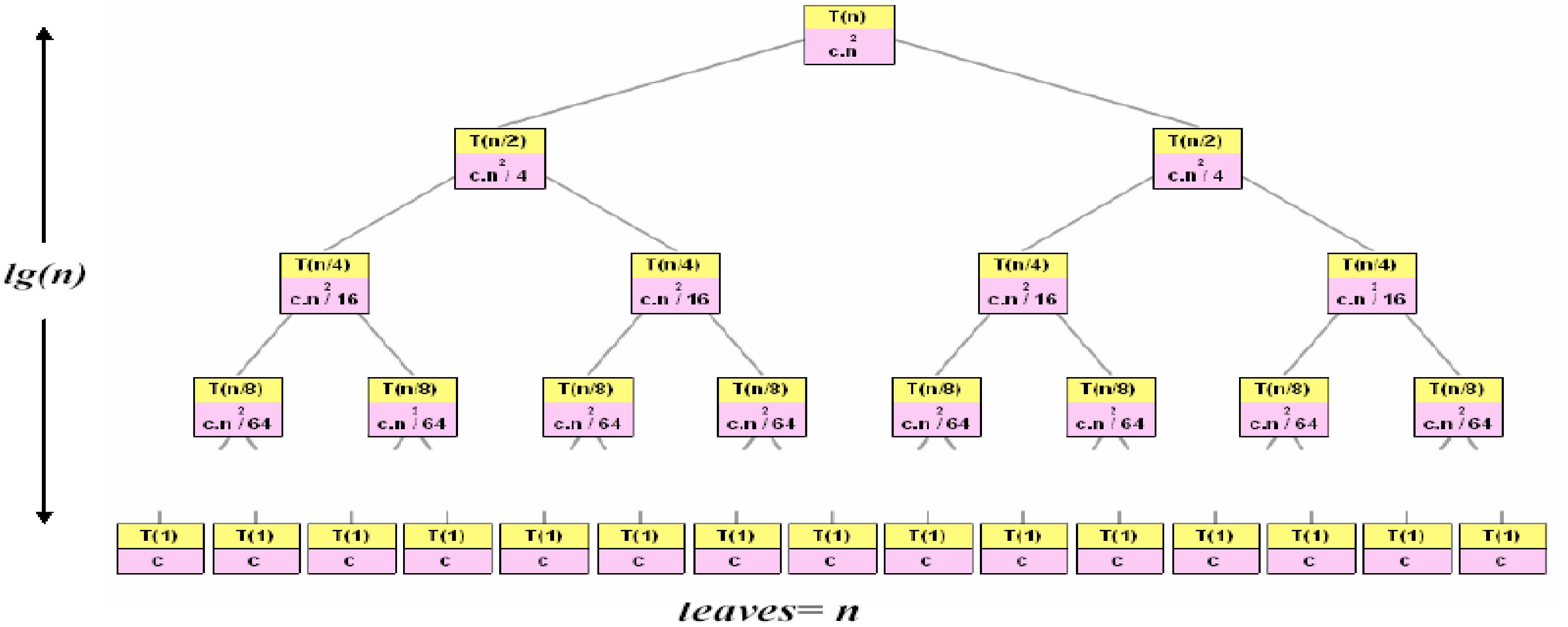


Recursion Tree

Example(2): $T(n) = 2T(n/2) + cn^2$, $n>1$, $T(1)=c$

Step#2: Inserting costs

The root has associated size of n and cost of cn^2. Each child of root has size $n/2$ and associated cost $cn^2/4$. At the next level the costs are reduced by a factor of 4. This reduction is continued up to the bottom level. Each leaf has associated cost of c.



Recursion Tree

Step #3: Summing up rows and leaves costs

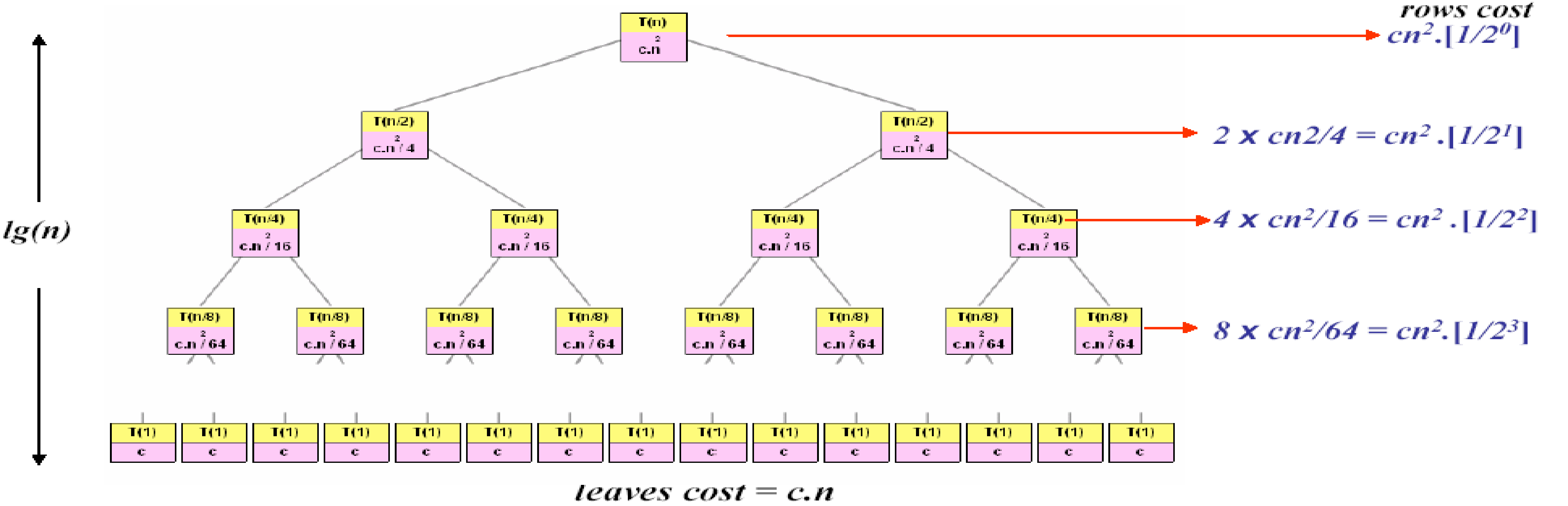
Summing the costs associated with the internal nodes and leaves:

$$T(n) = cn^2 \cdot [1/2^0 + 1/2^1 + 1/2^3 + \ldots\ldots\ldots 1/2^{\lg n-1}] + cn$$

The asymptotic behavior of the series is determined by the ***largest term,*** which is *1*. Thus,

$$1/2^0 + 1/2^1 + 1/2^3 + \ldots\ldots\ldots 1/2^{\lg n-1} = \theta(1).$$

Therefore, $T(n)=cn^2.\theta(1)+cn= \theta(n^2)$ (n^2 being the dominant term in the sum)



Example:

- $T(n) = 3T(n/4) + \Theta(n^2)$?

MASTER THEOREM

- Let T(n) be <u>a monotonically increasing</u> function that satisfies

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

$$T(1) = c$$

where $a \geq 1$, $b \geq 2$, $c>0$. If $f(n)$ is $\Theta(n^d)$ where $d \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{If } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

- You cannot use the Master Theorem if
  - $T(n)$ is not monotone, e.g. $T(n) = \sin(x)$
  - $f(n)$ is not a polynomial, e.g., $T(n)=2T(n/2)+2^n$
  - b cannot be expressed as a constant, e.g.

$$T(n) = T(\sqrt{n})$$

- Note that the Master Theorem does not solve the recurrence equation
- Does the base case remain a concern?

Example 1:

Let $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$. What are the parameters?

$$
\begin{aligned}
a &= \\
b &= \\
d &=
\end{aligned}
$$

Therefore which condition?

Let $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$. What are the parameters?

$$\begin{aligned} a &= 1 \\ b &= 2 \\ d &= 2 \end{aligned}$$

Therefore which condition?

Since $1 < 2^2$, case 1 applies.

Thus we conclude that

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

Example : 2

Let $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$. What are the parameters?

$$
\begin{aligned}
a &= \\
b &= \\
d &=
\end{aligned}
$$

Therefore which condition?

Let $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$. What are the parameters?

$$\begin{aligned} a &= 2 \\ b &= 4 \\ d &= \frac{1}{2} \end{aligned}$$

Therefore which condition?

Since $2 = 4^{\frac{1}{2}}$, case 2 applies.

Thus we conclude that

$$T(n) \in \Theta(n^d \log n) = \Theta(\sqrt{n} \log n)$$

Example: 3

Let $T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1$. What are the parameters?

$$\begin{aligned} a &= \\ b &= \\ d &= \end{aligned}$$

Therefore which condition?

Let $T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1$. What are the parameters?

$$\begin{aligned} a &= 3 \\ b &= 2 \\ d &= 1 \end{aligned}$$

Therefore which condition?

Since $3 > 2^1$, case 3 applies. Thus we conclude that

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$$

Fourth condition

- Recall that we cannot use the Master Theorem if f(n), the non-recursive cost, is not a polynomial
- There is a limited 4th condition of the Master Theorem that allows us to consider polylogarithmic functions

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$. Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Example :4

Say that we have the following recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

Clearly, $a = 2, b = 2$ but $f(n)$ is not a polynomial. However,

$$f(n) \in \Theta(n\log n)$$

for $k = 1$, therefore, by the 4-th case of the Master Theorem we can say that

$$T(n) \in \Theta(n\log^2 n)$$

Examples

- $T(n) = 9T(n/3)+n;$
  –
  –
  –
  –
- $T(n) = T(2n/3)+1$
  –
  –
  –

- $T(n) = 3T(n/4)+n\lg n;$

Practice Problems

For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

1. $T(n) = 3T(n/2) + n^2$

2. $T(n) = 4T(n/2) + n^2$

3. $T(n) = 3T(n/3) + \sqrt{n}$

4. $T(n) = 2^n T(n/2) + n^n$

5. $T(n) = 16T(n/4) + n$

6. $T(n) = 2T(n/2) + n \log n$

7. $T(n) = 2T(n/2) + n/\log n$

8. $T(n) = 2T(n/4) + n^{0.51}$

9. $T(n) = 0.5T(n/2) + 1/n$

10. $T(n) = 16T(n/4) + n!$

11. $T(n) = \sqrt{2}T(n/2) + \log n$

12. $T(n) = 3T(n/2) + n$