CS4349
Ch 3: Recurrences

Recurrence

- Is an equation or inequality that describes a function in terms of its value on smaller inputs.
- Example: Worst case running time T(n) of Merge-Sort is represented with the recurrence:

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

whose solution is $T(n) = \Theta(n \, lgn)$

What Does Recursion Really Mean?

- $T(n) = 2T(n/2) + n$
  - To solve a problem of size n we must solve two subproblems of size n/2 and do n units of additional work.
- $T(n) = T(n/4) + n^2$
  - To solve a problem of size n we must solve 1 subproblem of size n/4 and do n^2 units of additional work.
- $T(n) = 3T(n-1) + n$
  - To solve a problem of size n, we must solve 3 subproblems of size $n - 1$ and do n additional units of work.

3 methods to solve recurrences

- ***Substitution method***, we guess a bound and then use mathematical induction to prove our guess correct.
- ***Recursion-tree method*** converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion.
- ***Master method*** provides bounds for recurrences of the form $T(n) = aT(n/b) + f(n)$ where $a \geq 1$, $b > 1$, and $f(n)$ is a given function.

Substitution method

- Guess the form of the solution.
- Use mathematical induction to find the constants and show that the guess is correct.
- We substitute the guessed solution for the function when applying the inductive hypothesis to smaller values; hence the name “substitution method.”

Substitution method
Example 1

- Prove that $T(n) = 3T(n/3) + n = O(n\log n)$
- Need to show that $T(n) \leq c\, n \log n$ for some c, and sufficiently large n
- Assume above is true for $T(n/3)$, i.e.

  $T(n/3) \leq cn/3 \log (n/3)$

Substitution method Example 1 (cont'd)

$T(n) = 3\,T(n/3) + n$

$\leq 3\,cn/3 \log(n/3) + n$

$\leq cn \log n - cn \log 3 + n$

$\leq cn \log n - (cn \log 3 - n)$

$\leq cn \log n \qquad$ (if $cn \log 3 - n \geq 0$)

$cn \log 3 - n \geq 0$

$\Rightarrow \quad c \log 3 - 1 \geq 0 \qquad$ (for $n > 0$)

$\Rightarrow \quad c \geq 1/\log 3$

$\Rightarrow \quad c \geq \log_3 2 \qquad$ (because $\log_b(c) = 1 / \log_c(b)$)

Therefore, $T(n) = 3\,T(n/3) + n \leq cn \log n$ for $c = \log_3 2$ and $n > 0$.

By definition, $T(n) = O(n \log n)$

Substitution method
Example 2

- Prove that $T(n) = T(n/3) + T(2n/3) + n = O(n\log n)$
- Need to show that $T(n) \leq c\, n \log n$ for some c, and sufficiently large n
- Assume above is true for $T(n/3)$ and $T(2n/3)$, i.e.

  $T(n/3) \leq cn/3 \log (n/3)$

  $T(2n/3) \leq 2cn/3 \log (2n/3)$

Substitution method
Example 2 (cont'd)

$$T(n) = T(n/3) + T(2n/3) + n$$
$$\leq cn/3 \log(n/3) + 2cn/3 \log(2n/3) + n$$
$$\leq cn \log n + n - cn (\log 3 - 2/3)$$
$$\leq cn \log n + n(1 - c\log 3 + 2c/3)$$
$$\leq cn \log n, \text{ for all } n > 0 \text{ (if } 1 - c \log 3 + 2c/3 \leq 0)$$

$$c \log 3 - 2c/3 \geq 1$$
$$\Rightarrow c \geq 1 / (\log 3 - 2/3) > 0$$

Therefore, $T(n) = T(n/3) + T(2n/3) + n \leq cn \log n$ for $c = 1 / (\log 3 - 2/3)$ and $n > 0$. By definition, $T(n) = O(n \log n)$.

Substitution method
Example 2 (cont'd)

$$\frac{cn}{3}\log(n/3) = \frac{cn}{3}\log n - \frac{cn}{3}\log 3 \quad \text{.... I}$$

$$\frac{2cn}{3}\log(2n/3) = \frac{2cn}{3}\log 2n - \frac{2cn}{3}\log 3$$

$$= \frac{2cn}{3}\log 2 + \frac{2cn}{3}\log n - \frac{2cn}{3}\log 3 \quad \text{.... II}$$

$$cn\log n - cn\log 3 + \frac{2cn}{3} + n \quad \text{..... (Adding I and II)}$$

$$= cn\log n + n - cn(\log 3 - 2/3) \quad \text{.... (Reparanthesize)}$$

Recursion Tree Method

- How to draw a Recursion Tree:
- We draw the recursion tree in levels. Each level represents a level of recursion and has 3 parts:
  - I: The problem size
  - II: The recursion tree itself
  - III: The total work done

Calculating the Total Cost in Recursion Tree

- In order to calculate the total cost we need the following:
  - number of subproblems
  - size of each subproblem
  - total work done

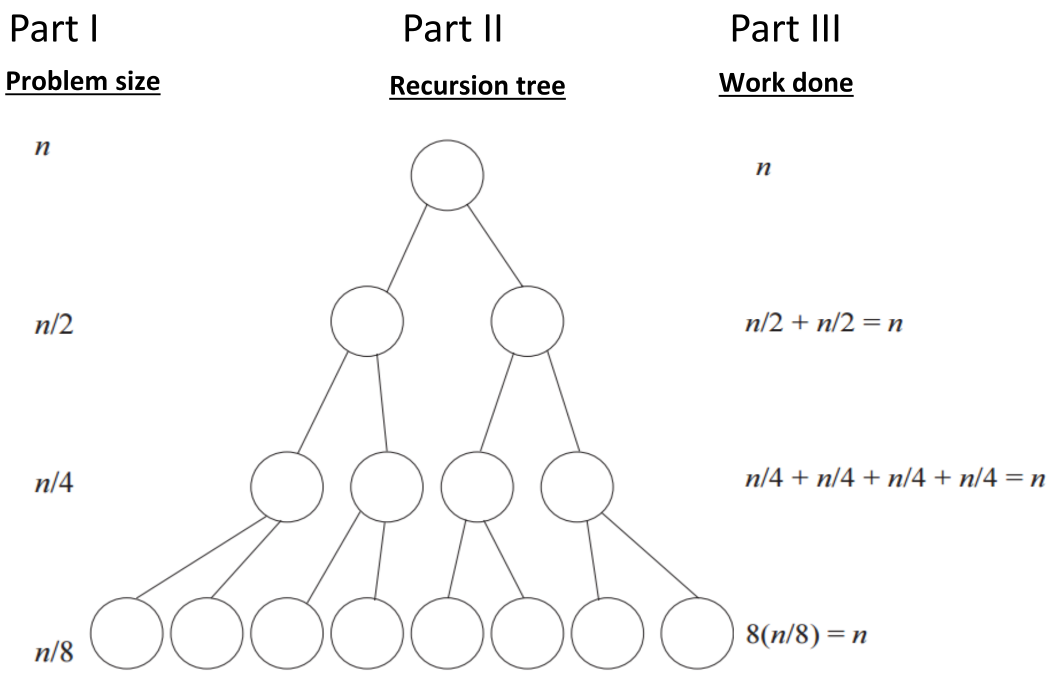
Recursion Tree – Example 1

Draw the recursion tree for the recurrence:

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

- **Level 0:** We have problem of size n, i.e. the original problem. We then draw a root vertex with 2 edges leaving it to show that we are splitting the original problem into 2 problems. We note on the right that we do n units of work in addition to whatever is done on the two new problems we created.

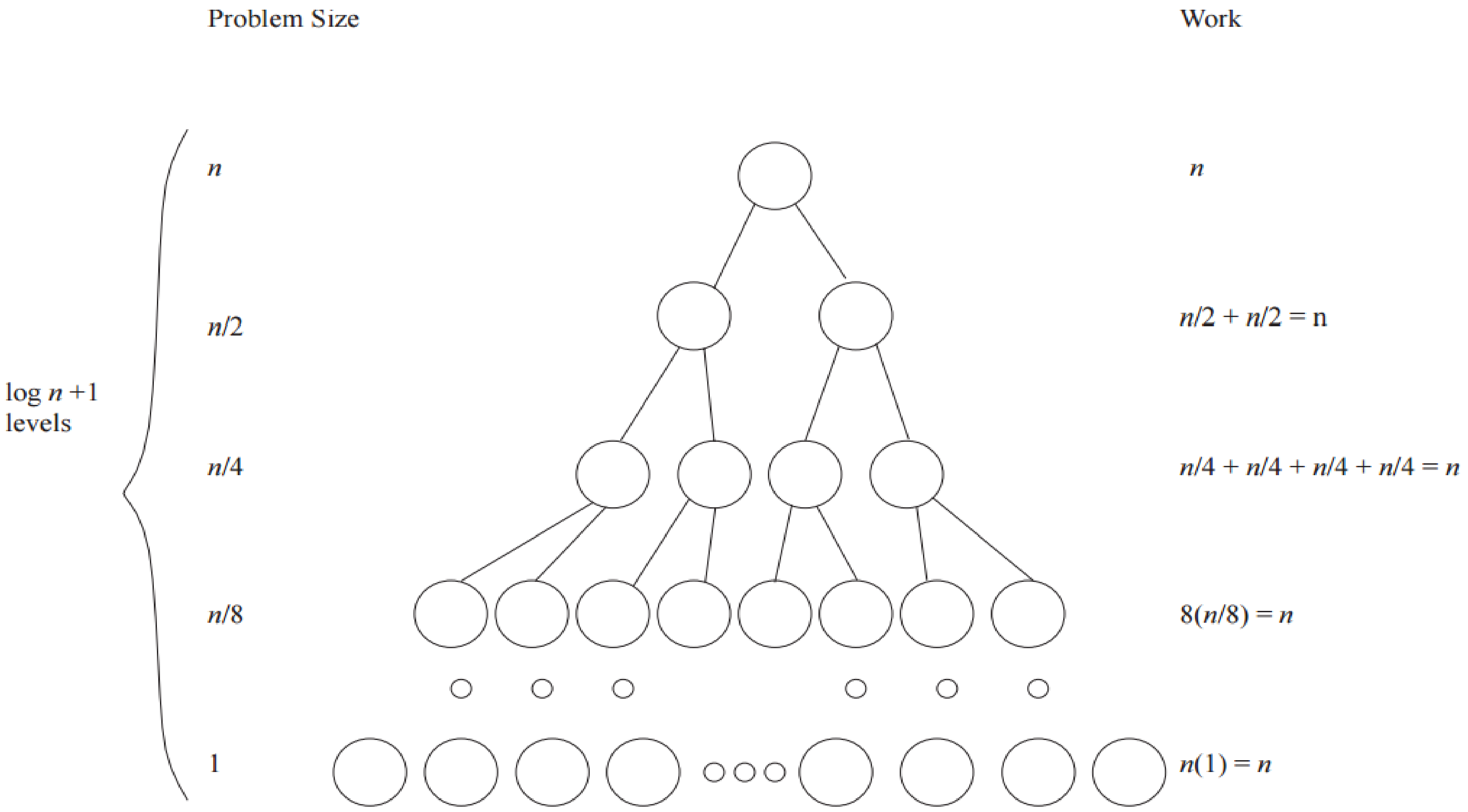
- **Level 1:** We draw 2 vertices in the middle representing the two problems into which we split our main problem and show on the left that each of these problems has size n/2. You can see how the recurrence is reflected in levels 0 and 1 of the recursion tree. The top vertex of the tree represents T(n), an the next level we have two problems of size n/2, giving us the recursive term 2T(n/2) of our recurrence. After we solve these two problems, we return to level 0 and do n additional units of work (to complete the non-recursive term of the recurrence).



Recursion Tree – Example 1 (cont'd)

- At level i, we have 2^i subproblems of size $n/2^i$, totaling a 2^i x $[n/(2^i)]$ = n units of work done per level.
- At each level the problem size is cut in half, and the tree stops when the problem size is 1. Therefore there are $\lg_2 n + 1$ levels of the tree, since we start with the top level and cut the problem size in half $\lg_2 n$ times.
- Since there are $\lg_2 n + 1$ levels, and at each level the amount of work we do is n units, the total amount of work done to solve the recurrence is: $n(\lg_2 n + 1)$, i.e. the solution to the recurrence is: $T(n) = n(\lg_2 n + 1) = n\lg_2 n + n$.
- The additional term "n" can be ignored for large n, therefore making $T(n)= n\lg_2 n$.

Recursion Tree – Example 1 (cont'd)



Recursion Tree – Example 2

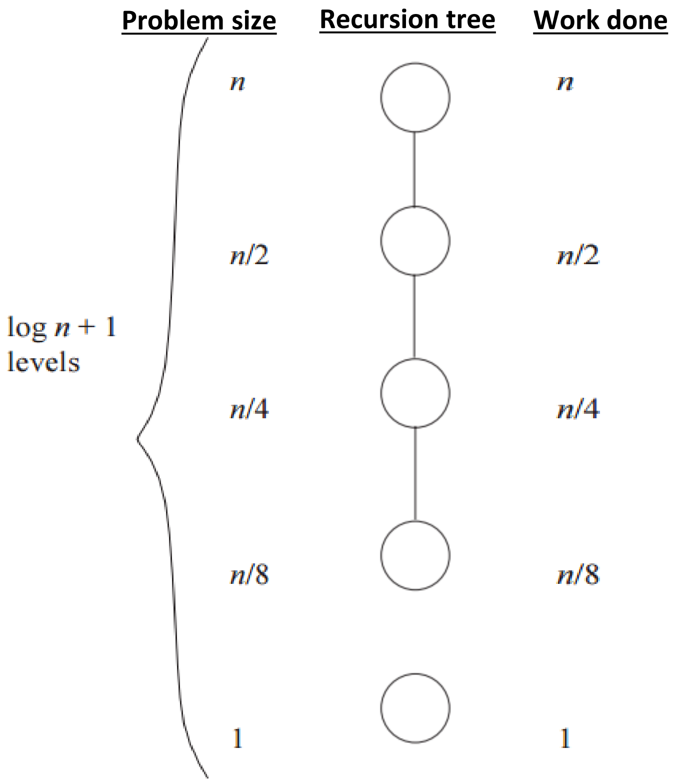
- Calculate the total cost of the following recurrence by using recursion tree method.

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

- The recurrence means to solve a problem of size n, we must solve one subproblem of size n/2 and do n units of additional work.

Recursion Tree – Example 2 (cont'd)

- The problem sizes are the same as in Example-1. However, the number of subproblems does not double, rather it remains at one at each level. Consequently, the amount of work halves at each level. There are still log n + 1 levels. So at level i, we have 1 subproblem of size $n/2^i$, for total work of $n/2^i$ units.



Total amount of work done is:

$$n + n/2 + n/4 + \cdots + 2 + 1 = n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \left(\frac{1}{2}\right)^{\log_2 n}\right)$$

which is n times a **geometric series**.
The value of a geometric series in which the largest term is one is Θ(1).
Therefore, the total work done for the problem is T(n) = Θ(n).

Some useful series

Arithmetic series

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

Harmonic Series

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$$

Geometric Series

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} \quad (x \neq 1)$$

$$\sum_{k=0}^{n-1} x^k = \frac{x^n - 1}{x - 1} (x \neq 1)$$

Special Cases of Geometric Series:

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

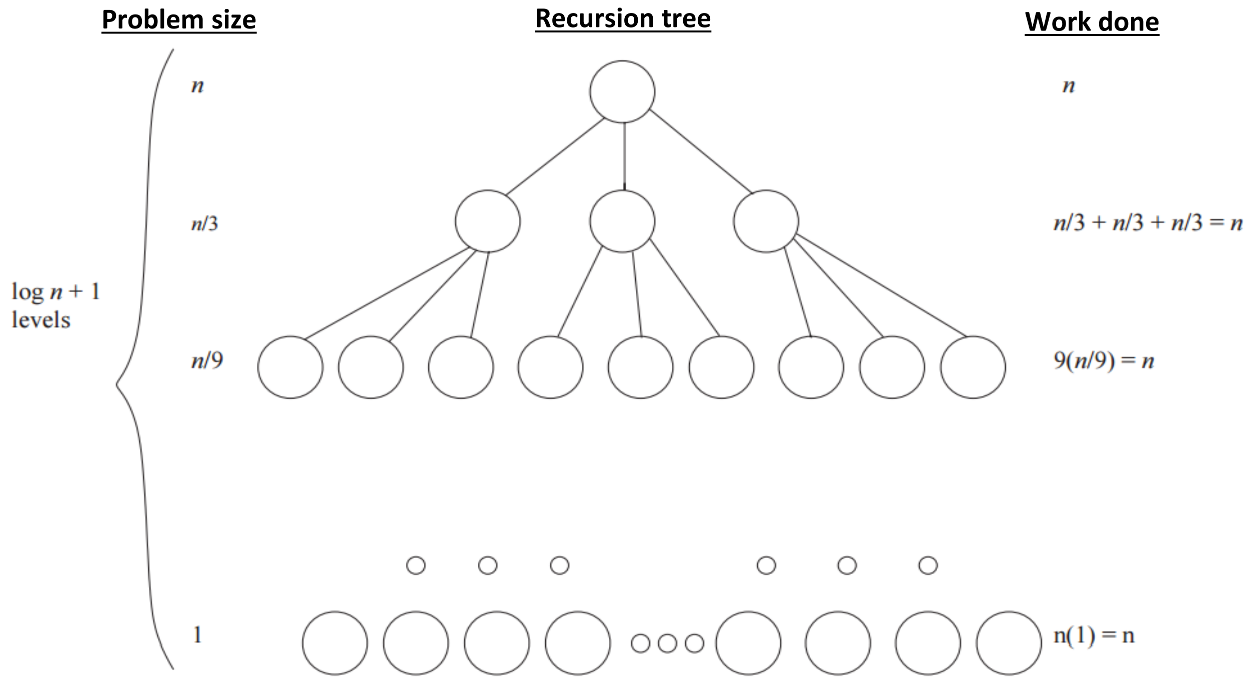
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} \qquad \text{if } x < 1$$

Recursion Tree – Example 3

- Find a Θ bound for the solution to the recurrence by using a recurrence tree. Assume that n is a power of 3.

$$T(n) = \begin{cases} 3T(n/3) + n & \text{if } n \geq 3 \\ 1 & \text{if } n < 3 \end{cases}$$

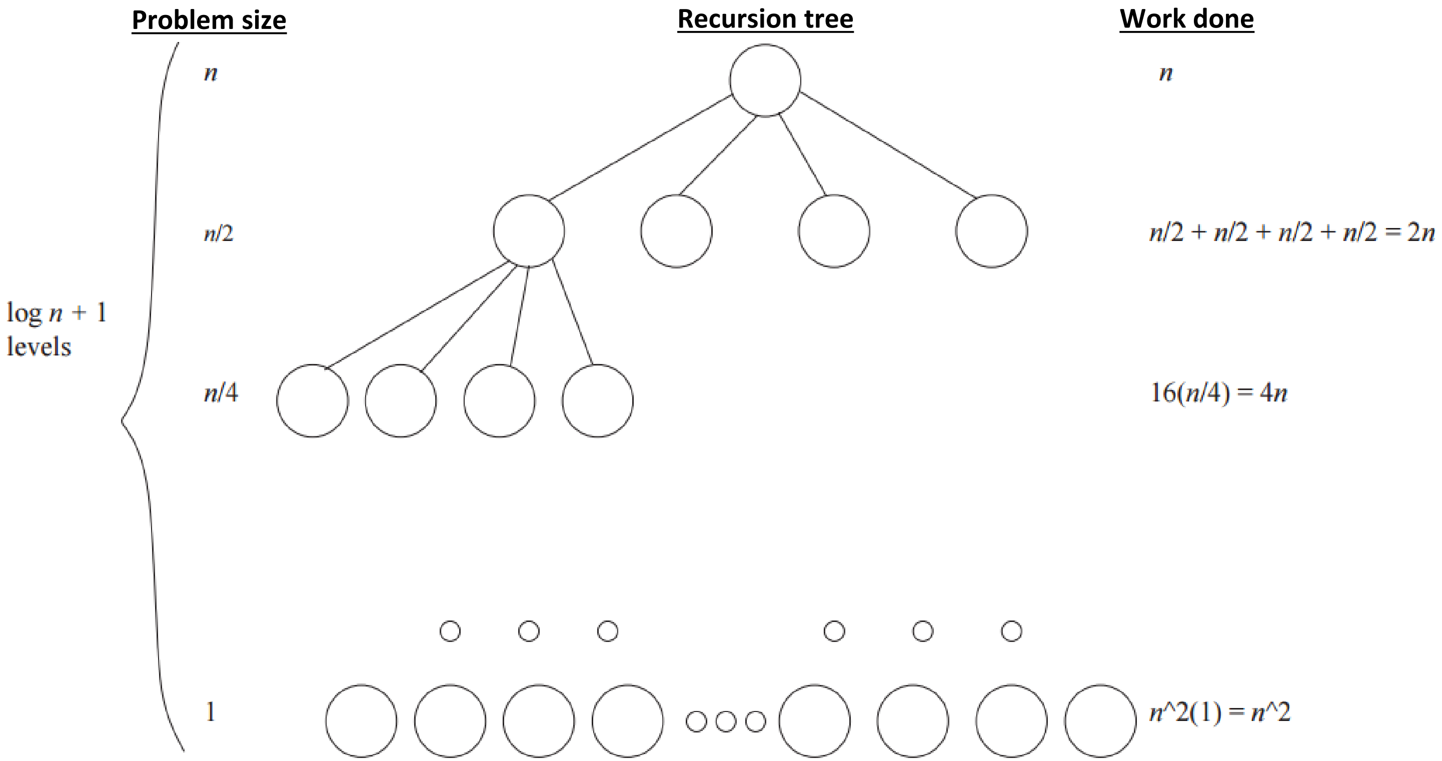
- In the recurrence tree, at each step we divide into 3 subproblems of size n/3. And there are $\log_3$ n + 1 levels. The total work is Θ(n log n) units.



Recursion Tree – Example 4

- Solve the following recurrence using a recursion tree.

$$T(n) = \begin{cases} 4T(n/2) + n & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$



Recursion Tree – Example 4 (cont'd)

- There are $\log_2 n + 1$ levels in the recursion tree and we have 4 subproblems of size n/2 at each level. Thus level 0 has 1 node, level 1 has 4 nodes, level 2 has 16 nodes, … level i has 4^i nodes. At level i, each node corresponds to a subproblem of size $n/2^i$ and hence requires $n/2^i$ units of additional work. Thus the total work on level i is 4^i x $(n/2^i)=2^i n$ units. Summing up the work at all levels we get:

$$\sum_{i=0}^{\log_2 n} 2^i n = n \sum_{i=0}^{\log_2 n} 2^i.$$

- Then the solution will be:

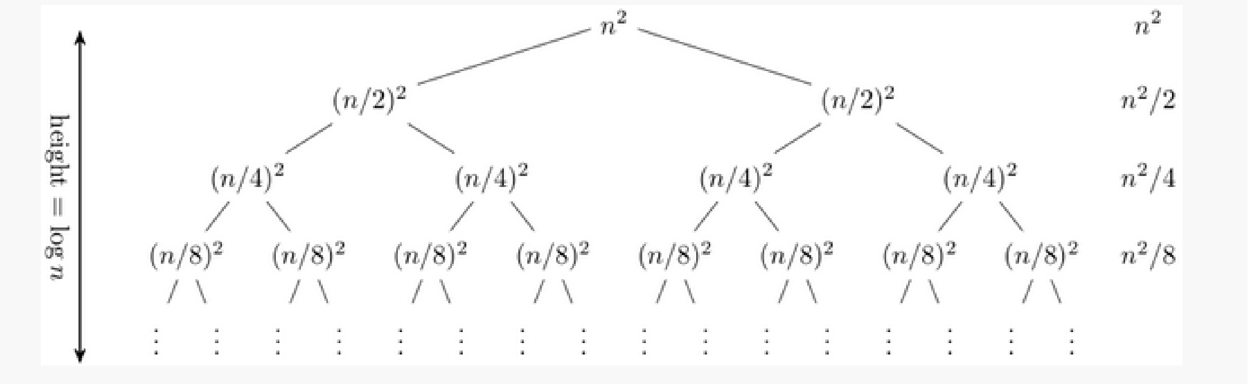
$$\begin{aligned} T(n) &= n \sum_{i=0}^{\log_2 n} 2^i \\ &= n \frac{1 - 2^{(\log_2 n)+1}}{1-2} \\ &= n \frac{1-2n}{-1} \\ &= 2n^2 - n \\ &= \Theta(n^2)\ . \end{aligned}$$

$(a^{\log_a x} = x)$

Recursion Tree – Example 5

- Solve the recurrence $T(n) = 2T(n/2) + n^2$ using a recursion tree.

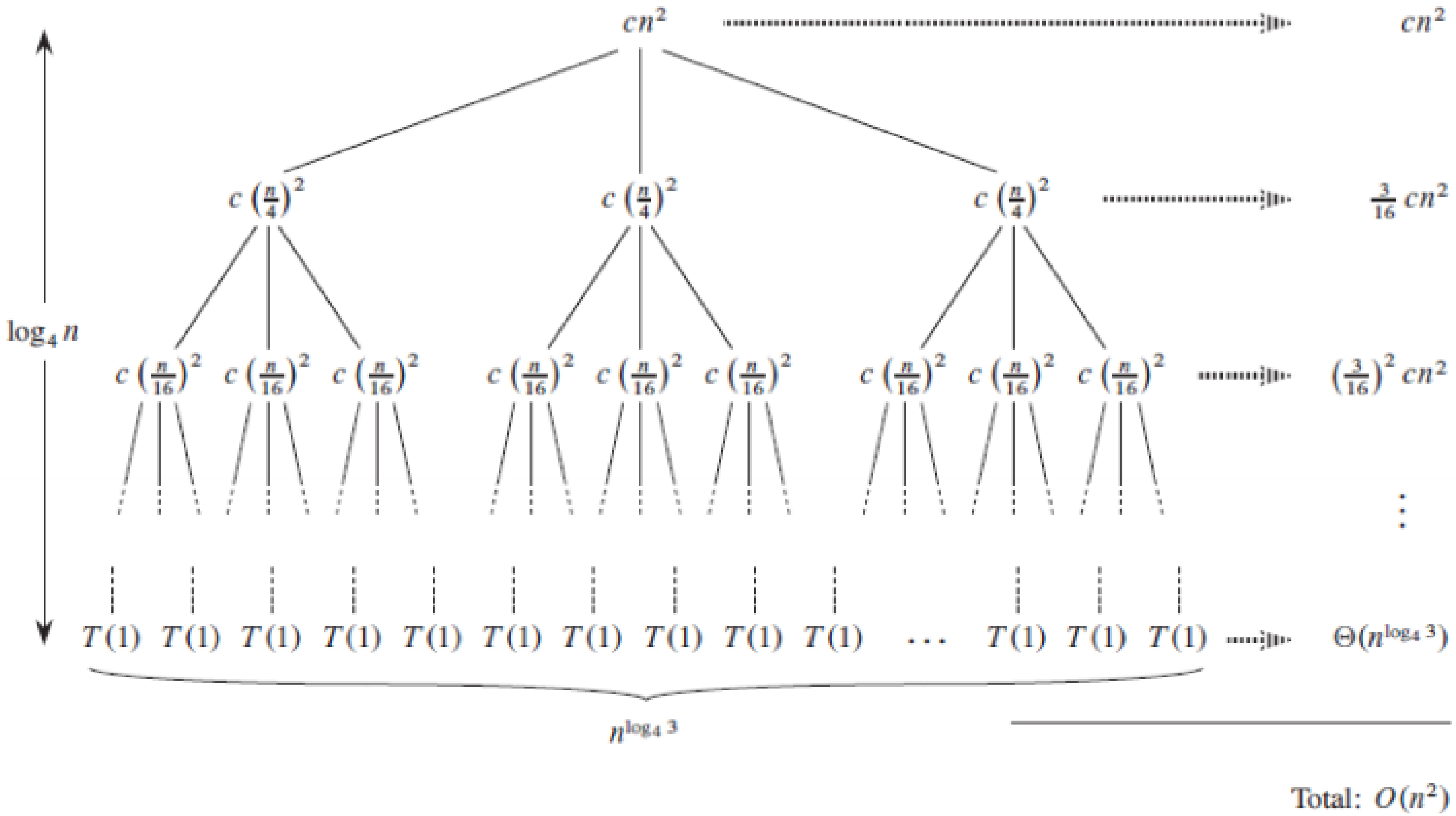
Recursion tree **Work done**



- This is a geometric series, thus in the limit the sum is $O(n^2)$.

Recursion Tree - Example 6

$T(n) = 3T(n/4) + cn^2$



Recursion Tree - Example 6 (cont'd)

$T(n) = 3T(n/4) + cn^2$

- Subproblem size at level i is: $n/4^i$
- Subproblem size hits 1 when $1 = n/4^i \Rightarrow i = \log_4 n$
- Cost of a node at level i = $c(n/4^i)^2$
- Number of nodes at level i = $3^i \Rightarrow$ last level has $3^{\log_4 n} = n^{\log_4 3}$ nodes
- Total cost:

$$T(n) = \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \le \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) = \frac{1}{1-\frac{3}{16}} cn^2 + \Theta\left(n^{\log_4 3}\right) = O(n^2)$$

$\Rightarrow T(n) = O(n^2)$

Master Method

- The master method depends on the Master Theorem:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

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 $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Master Method – Example 1

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

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n$

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$

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

Master Method – Example 2

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

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2$

Case 2: $f(n) = \Theta(n^2)$

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

Master Method – Example 3

$T(n) = 4T(n/2) + n^3$

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^3$

Case 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$ ***and*** $4(n/2)^3 \leq cn^3$ for $c = 1/2$

$\therefore T(n) = \Theta(n^3)$

Master Method – Example 4

$T(n) = 4T(n/2) + n^2/\log n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2/\log n$

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^{\varepsilon} = \omega(\log n)$.

Master Method – Example 5

$T(n) = 4T(n/2) + n^{2.5}$

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^{2.5}$

Case 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 0.5$ ***and***
$4(n/2)^{2.5} \le cn^{2.5}$ (reg. cond.) for $c = 0.75$

$\therefore\ T(n) = \Theta(n^{2.5})$

Master Method – Example 6

$T(n) = 4T(n/2) + n^2 \log n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2 \log n.$

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\log n)$.

More Exercises for Master Method

a. $T(n) = 4T(n/2) + n;$

b. $T(n) = 9T(n/3) + n^2;$

c. $T(n) = 6T(n/4) + n;$

d. $T(n) = 2T(n/4) + n;$

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

f. $T(n) = 4T(n/4) + n \log n.$