COT 6405 Introduction to Theory of Algorithms

Topic 4. Recurrences

Recurrences

- What is a recurrence?
  - An equation that describes a function in terms of its value on smaller functions
- The time complexity of divide-and-conquer algorithms can be expressed as recurrences

Recurrence Examples

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

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

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

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

Solving the recurrences

- Substitution method
- Recursion Tree
- Master method

Substitution method

- The substitution method comprises two steps:
  - 1. Guess the form of the solution
  - 2. Use mathematical induction to show the correctness of the guess

Example:

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

1. *Guess:* $T(n) = n \lg n + n$. *[Here, we have a recurrence with an exact function, rather than asymptotic notation, and the solution is also exact rather than asymptotic. We'll have to check boundary conditions and the base case.]*
2. *Induction:*

   Basis: $n = 1 \Rightarrow n \lg n + n = 1 = T(n)$

   Inductive step: Inductive hypothesis is that $T(k) = k \lg k + k$ for all $k < n$. We'll use this inductive hypothesis for $T(n/2)$.

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \\ &= 2\left(\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}\right) + n \qquad \text{(by inductive hypothesis)} \\ &= n\lg\frac{n}{2} + n + n \\ &= n(\lg n - \lg 2) + n + n \\ &= n\lg n - n + n + n \\ &= n\lg n + n . \end{aligned}$$

■

Substitution method (cont’d)

- We generally express the solution by asymptotic notations
- We don’t worry about boundary cases, nor do we show base cases in the substitution proof.
  - because we are ultimately interested in an asymptotic solution to a recurrence, it will always be possible to choose base cases that work.

Example: $T(n) = 2T(n/2) + \Theta(n)$. If we want to show an upper bound of $T(n) = 2T(n/2) + O(n)$, we write $T(n) \le 2T(n/2) + cn$ for some positive constant c.

1. ***Upper bound:***

   Guess: $T(n) \le dn \lg n$ for some positive constant d. We are given c in the recurrence, and we get to choose d as any positive constant. It's OK for d to depend on c.

   Substitution:

Guess T(n) = Θ ($nlgn$)
Prove: T(n) = O($nlgn$) and Ω($nlgn$)

$$
\begin{aligned}
T(n) &\le 2T(n/2) + cn \\
&\le 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\
&= dn\lg\frac{n}{2} + cn \\
&= dn\lg n - dn + cn \\
&\le dn\lg n \qquad \text{if } -dn + cn \le 0\,, \\
& \qquad\qquad\qquad\qquad d \ge c
\end{aligned}
$$

2. ***Lower bound:*** Write $T(n) \geq 2T(n/2) + cn$ for some positive constant c.

Guess: $T(n) \geq dn \lg n$ for some positive constant d.

Substitution:

$$\begin{aligned} T(n) &\geq 2T(n/2) + cn \\ &\geq 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\ &= dn\lg\frac{n}{2} + cn \\ &= dn\lg n - dn + cn \\ &\geq dn\lg n \qquad \text{if } -dn + cn \geq 0, \\ & \qquad\qquad\qquad\qquad d \leq c \end{aligned}$$

Guess $T(n) = \Theta(nlgn)$
Prove: $T(n) = O(nlgn)$ and $\Omega(nlgn)$

Therefore, $T(n) = \Omega(n \lg n)$.

Therefore, $T(n) = \Theta(n \lg n)$. *[For this particular recurrence, we can use $d = c$ for both the upper-bound and lower-bound proofs. That won't always be the case.]* ■

Substitution method

- **For the substitution method:**
  - Show the upper and lower bounds separately.
    - Might need to use different constants for each.
- **Making a good guess**
  - Unfortunately, there is no general way to guess the correct solutions to recurrences.
  - Takes experience and creativity.

Make sure you show the same *exact* form when doing a substitution proof.

Consider the recurrence

$$T(n) = 8T(n/2) + \Theta(n^2) .$$

For an upper bound:

$$T(n) \le 8T(n/2) + cn^2 .$$

Guess: $T(n) \le dn^3$.

$$\begin{aligned} T(n) &\le 8d(n/2)^3 + cn^2 \\ &= 8d(n^3/8) + cn^2 \\ &= dn^3 + cn^2 \\ &\not\le dn^3 \end{aligned}$$

doesn't work!

Guess T(n) = $\Theta(n^3)$
Prove: T(n) = O(n^3) and $\Omega(n^3)$

How to fix this?

Remedy: *Subtract off* a lower-order term.

Guess: $T(n) \le dn^3 - d'n^2$.

$$\begin{aligned} T(n) &\le 8(d(n/2)^3 - d'(n/2)^2) + cn^2 \\ &= 8d(n^3/8) - 8d'(n^2/4) + cn^2 \\ &= dn^3 - 2d'n^2 + cn^2 \\ &= dn^3 - d'n^2 - d'n^2 + cn^2 \\ &\le dn^3 - d'n^2 \qquad \text{if } -d'n^2 + cn^2 \le 0\,, \\ & \qquad\qquad\qquad\qquad d' \ge c \end{aligned}$$

Avoiding Pitfalls

- It is easy to err in the use of asymptotic notation
- Solve $T(n) = 2T(n/2) + \Theta(n)$
- Guess: $T(n) = O(n)$ and $T(n) \leq dn$ for some positive constant number d
- Induction: $T(n) \leq 2T(n/2) + cn$

$$\leq 2(d(n/2)) + cn$$

$$\leq dn + cn = (d+c)n = O(n)$$

Why wrong?

Changing variables

- Sometimes, a little algebraic manipulations can make an unknown recurrence similar to one you have seen before.
- Solve the recurrence $T(n) = 2T(\sqrt{n}) + lgn$
  - Renaming $m = lgn$ yields $T(2^m) = 2T(2^{m/2}) + m$
  - We can now rename $S(m) = T(2^m)$ to produce the new recurrence $S(m) = 2S(m/2) + m$
  - $S(m) = \Theta(mlgm)$
  - $T(n) = T(2^m) = S(m) = \Theta(mlgm) = \Theta(lgnlglgn)$

Recursion tree method

- How to solve the recurrence of merge sort?
- By using substitution method, we can have
  - $T(n) = 2T(n/2) + n$

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

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

= …….

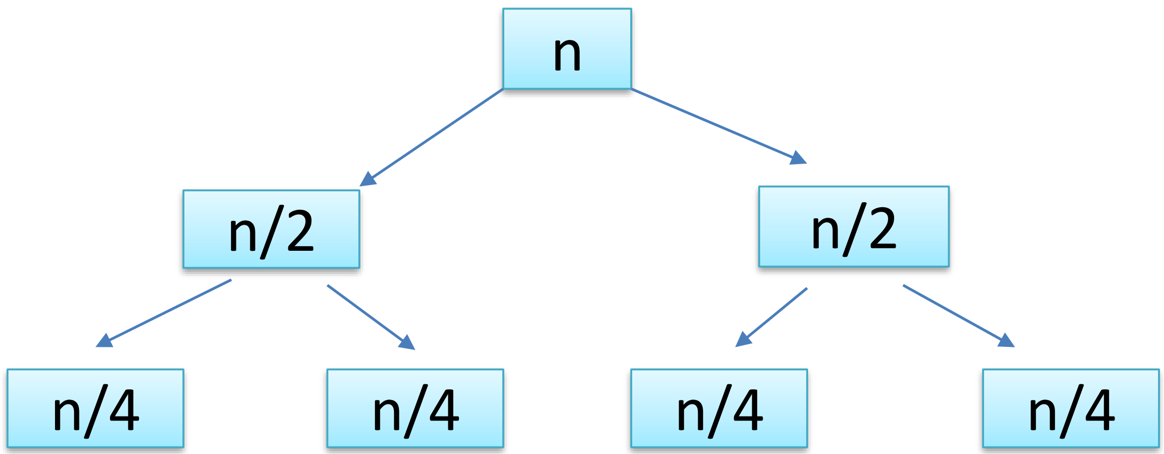
Recursion tree method (cont'd)

- An alternative approach: draw a tree to diagram all the recursive calls that take place

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

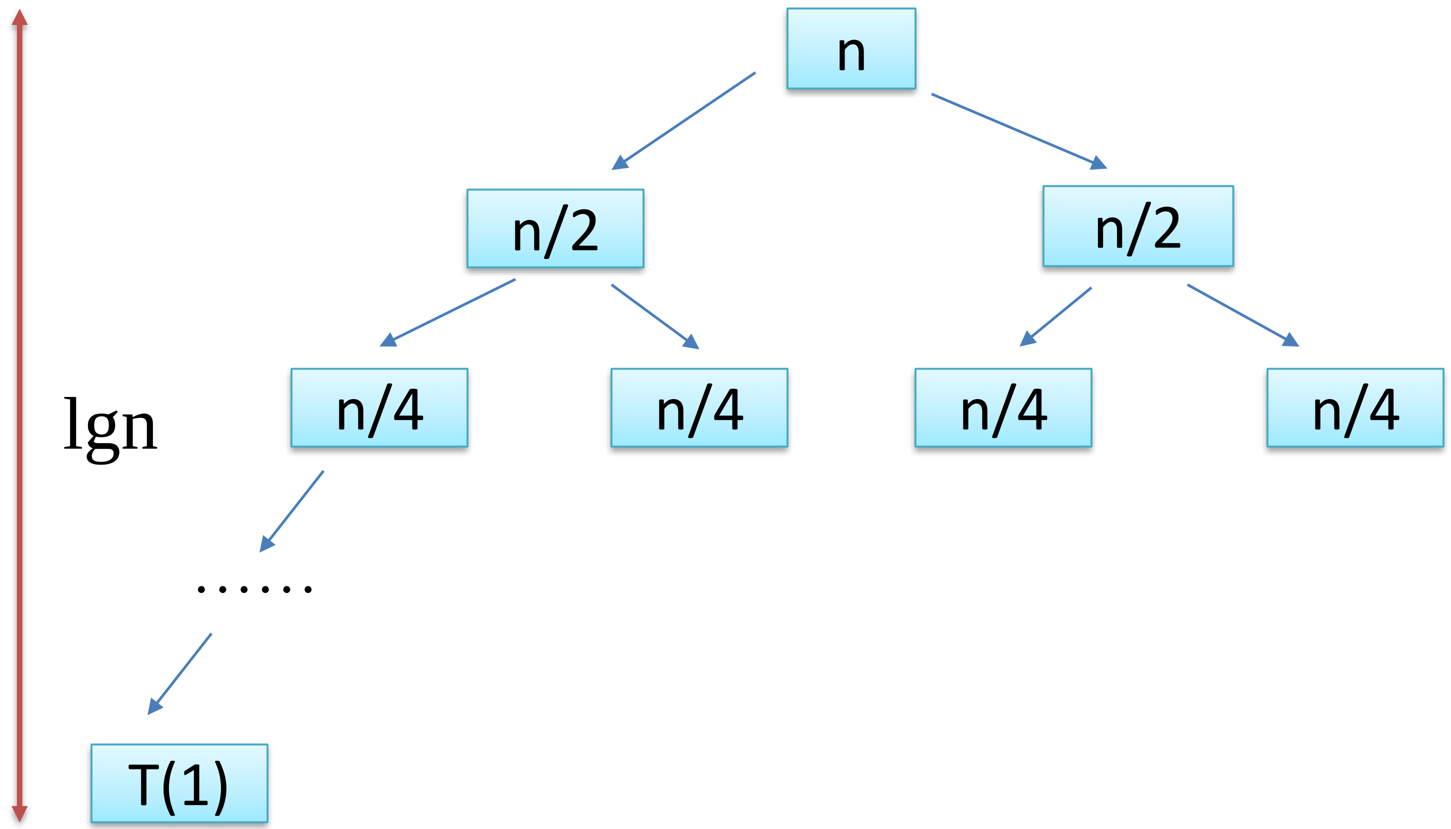
- For the original problem, we have a cost of n, plus the two subproblems, each costing n/2

Constructing the tree

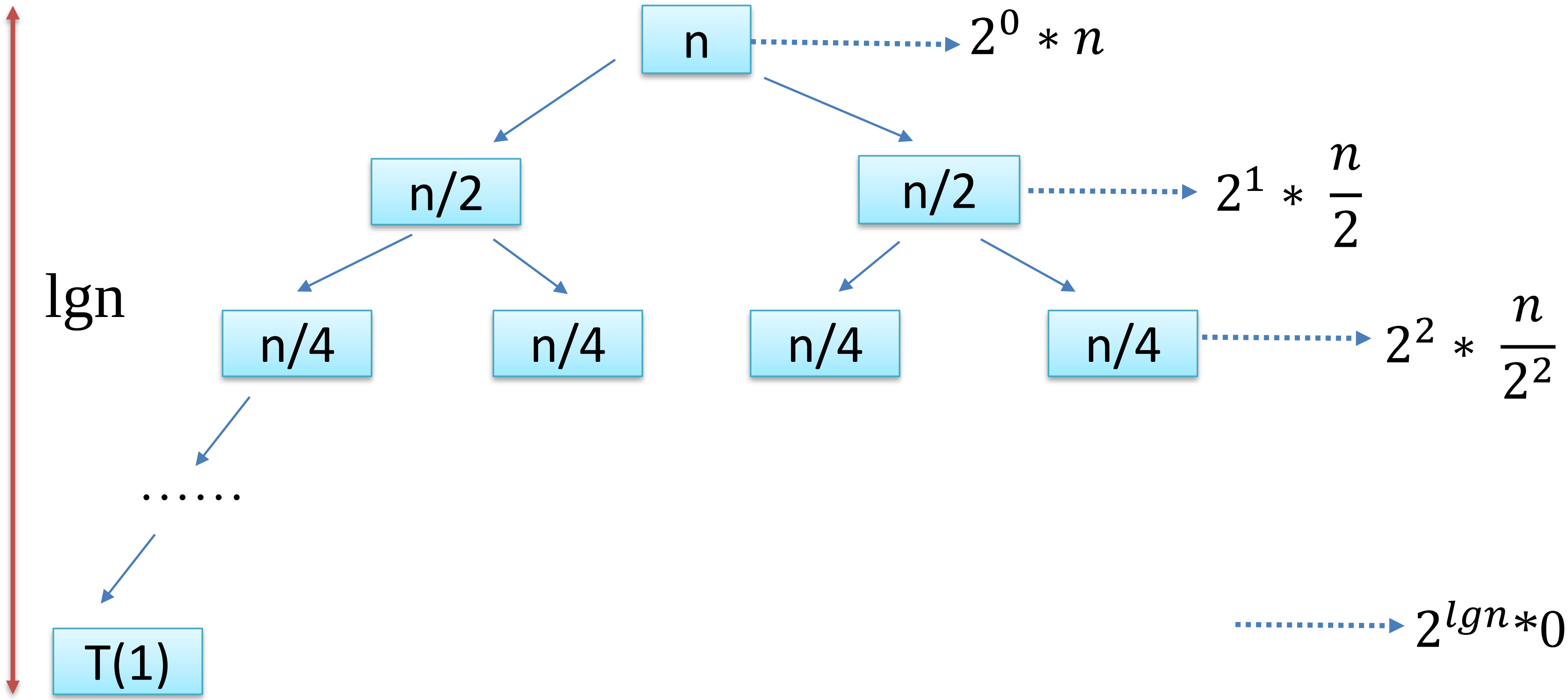


For each of the size-n/2 subproblems, we have a cost of n/2, plus two subproblems, each costing n/4

Constructing the tree (cont'd)



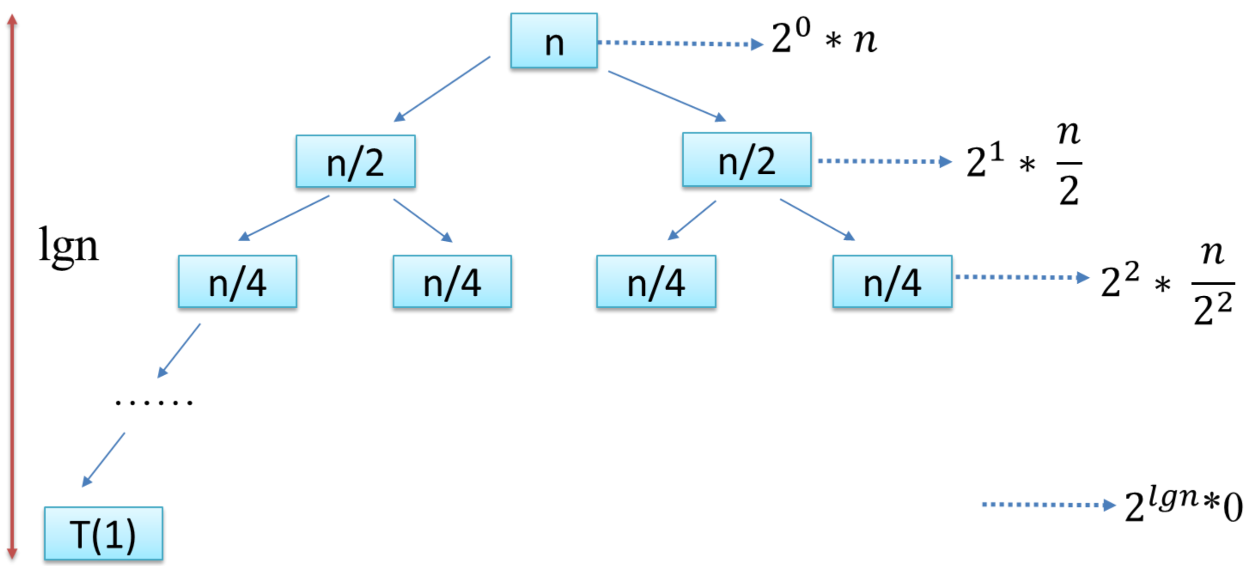
Constructing the tree (cont’d)



Computing the cost

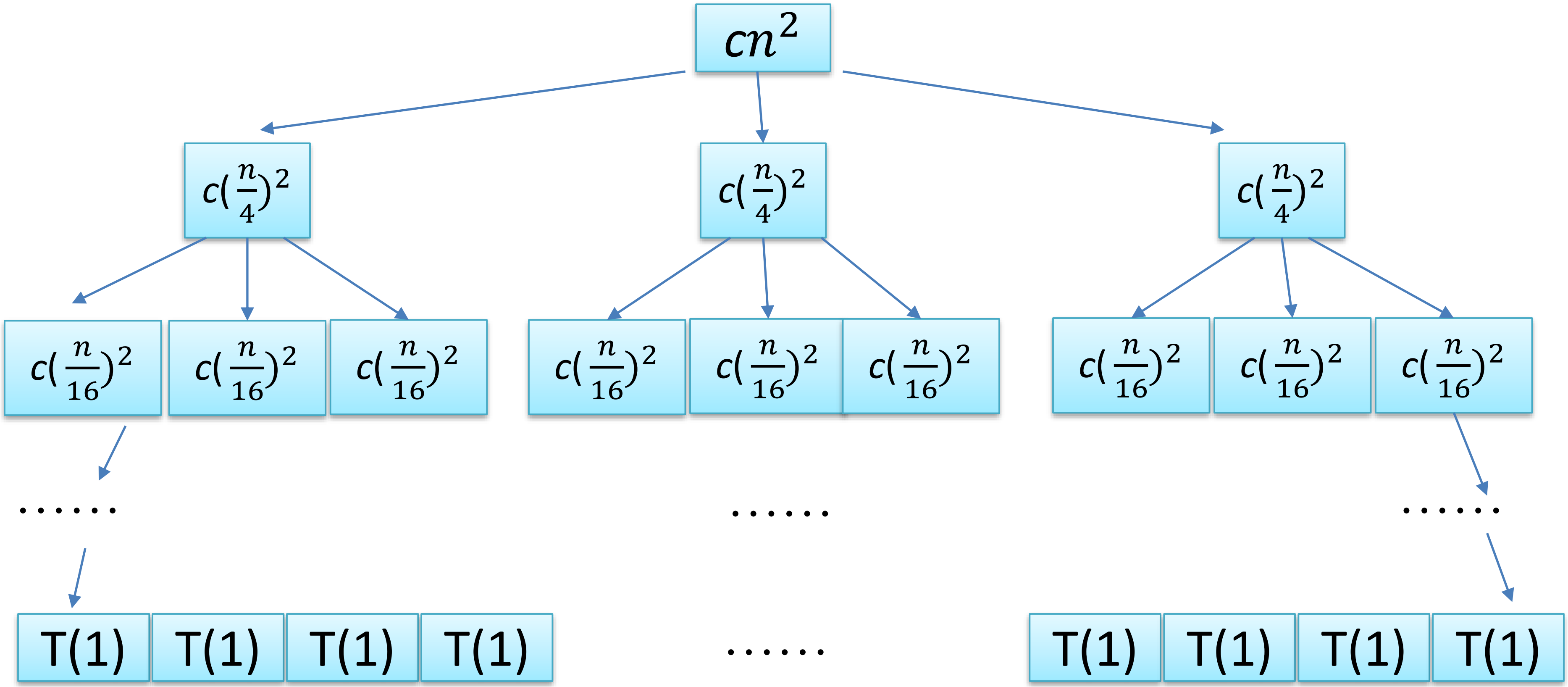
- We add up the costs over all levels to determine the cost for the entire tree
- $T(n) = \ 2^0 * n \ + 2^1 * \frac{n}{2} \ + 2^2 \ * \frac{n}{2^2} + \ldots\ldots + \ 2^{lgn} * \ 0$

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



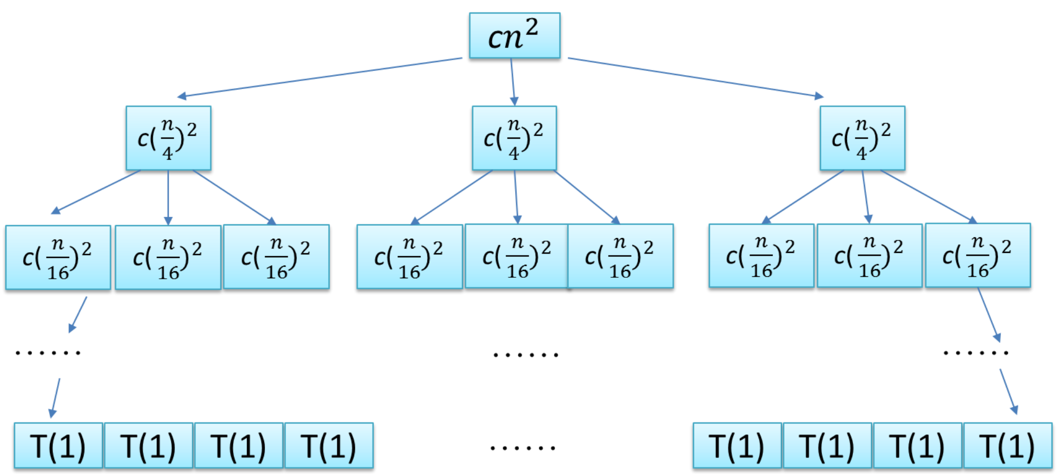
Example

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



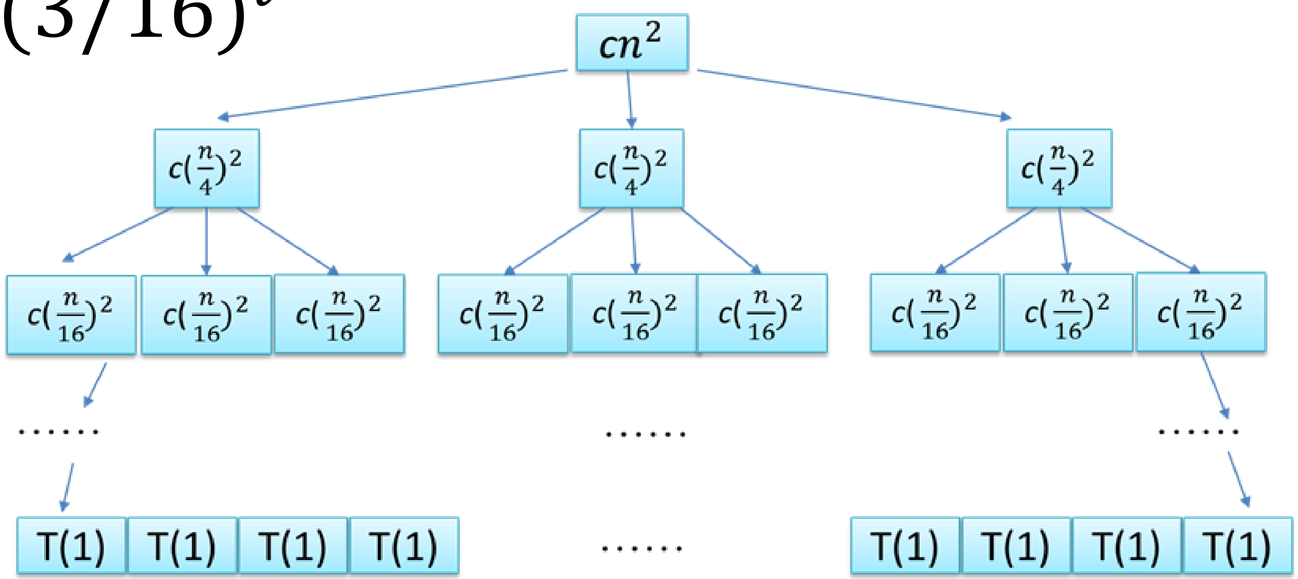
Example(cont'd)

- The subproblem size for a node at depth *i* is $n/4^i$
- The subproblem size hits T(1), when $n/4^i$ = 1, or $i = \log_4 n$
- Thus, tree has 1+$\log_4 n$ levels (*i* = 0,1,…$\log_4 n$)



Example(cont'd)

- Each node at level *i* has a cost of $c(n/4^i)^2$
- Each level has 3^i nodes
- Thus, the total cost of level *i* is $3^i c(n/4^i)^2$ = $cn^2(3/16)^i$



Example(cont'd)

- The bottom level has $3^{\log_4 n}$ = $n^{\log_4 3}$nodes, each costing T(1)
- Assume T(1) is a constant. The total cost of the bottom level will be

  T(1) $n^{\log_4 3}$ = $\Theta(n^{\log_4 3})$

Total cost

- The total cost of level i is $cn^2(3/16)^i$
- The total cost of the bottom level $\Theta(n^{\log_4 3})$
- We add up the costs over all levels to determine the total cost for the entire tree:

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

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

$$= \frac{\left(\frac{3}{16}\right)^{\log_4 n - 1} - 1}{\frac{3}{16} - 1} cn^2 + \Theta(n^{\log_4 3})$$

How to simplify the answer

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

$$\leq \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) = \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

How to simplify the answer (cont'd)

- On the other hand,

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

Thus, $T(n) = \Omega(n^2)$ and we conclude that

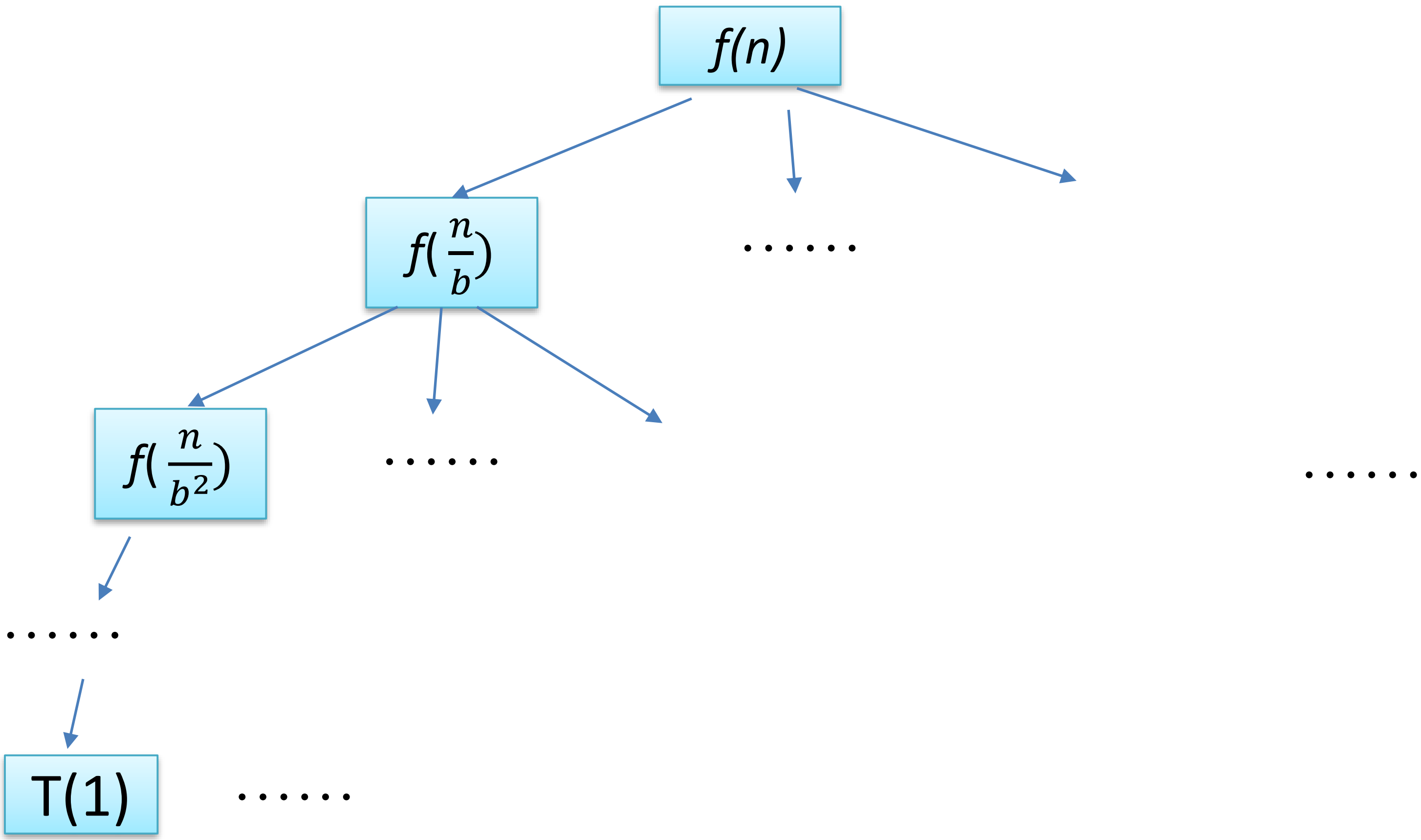
$$T(n) = \Theta(n^2)$$

How to use substitution method to verify?

Exercise

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

Exercise (cont'd)



Exercise (cont'd)

- The subproblem size for a node at depth *i* is n/b^i
- The subproblem size hits T(1), when n/b^i = 1, or $i = \log_b n$
- Thus, tree has 1+$\log_b n$ levels (*i* = 0,1,...$\log_b n$)

Exercise (cont'd)

- Each node at level *i* has a cost of f(n/b^i)
- Each level has a^i nodes
  - Level 0: 1, level 1: a, level 2: a^2, level 3: a^3….
- Thus, the total cost of level *i* is a^if(n/b^i)

Exercise (cont'd)

- The bottom level has $a^{\log_b n}$ = $n^{\log_b a}$nodes, each costing T(1)
- Assume T(1) is a constant. The total cost of the bottom level will be

  T(1)$n^{\log_b a}$= $\Theta(n^{\log_b a})$

Exercise (cont'd)

- We add up the costs over all levels to determine the total cost for the entire tree:

$$T(n) = f(n) + af(n/b) + a^2 f(n/b^2) + \cdots + a^{\log_b n - 1} f(n/b^{\log_b n - 1}) + \Theta(n^{\log_b a})$$

$$= \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) \;\; + \Theta(n^{\log_b a})$$