Recurrences

- The substitution method
- The recursion tree method
- The master method

The substitution method

- Guessing the form of the solution
- Using the mathematical induction to show that the solution works

The substitution method: an example

We'd like to solve $T(n) = 3T(\lfloor n/4 \rfloor) + n$.

We guess $T(n) \in O(n)$.

We prove by induction that there exists a constant c such that $T(n) \leq cn$ for sufficiently large n.

$$
\begin{aligned}
T(n) &= n + 3T(\lfloor n/4 \rfloor) \\
&\leq n + 3 * c * \lfloor n/4 \rfloor \\
&\leq (1 + 3c/4)n \\
&\leq cn,\ when\ c \geq 4
\end{aligned}
$$

The substitution method: subtleties

We'd like to solve $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.

We guess $T(n) \in O(n)$.

We try to prove by induction that there exists a constant c such that $T(n) \leq cn$ for sufficiently large n.

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\ &\leq c * \lfloor n/2 \rfloor + c * \lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

We really need to show that $T(n) \leq cn$.

The trick

We'd like to solve $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.

We guess $T(n) \in O(n)$.

We prove by induction that there exists a constant c, b such that $T(n) \leq cn - b$ for sufficiently large n.

$$
\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\
&\leq c * \lfloor n/2 \rfloor - b + c * \lceil n/2 \rceil - b + 1 \\
&= cn - b - (b - 1) \\
&< cn - b \; when \; b > 1
\end{aligned}
$$

The recursion-tree method

- The method
  - Draw a recursion tree where each node represents the cost of a single subproblem
  - Sum the cost of each level to get per-level cost
  - Sum all per-level costs to get the total cost
- Applications
  - Can be used to find a good guess. Complete by using the substitution method. Can be a bit sloppy when constructing the tree.
  - Can serve as a direct proof. Need to be strict when draw the tree.

The recursion-tree method: an example

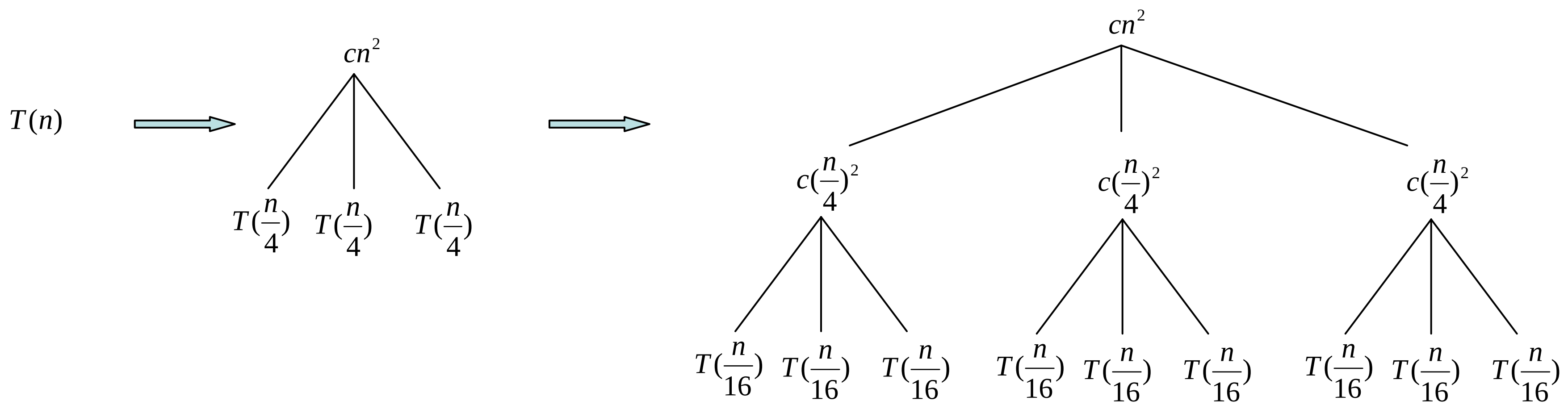
We'd like to solve $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.

We instead draw a tree for $T(n) = 3T(n/4) + cn^2$.

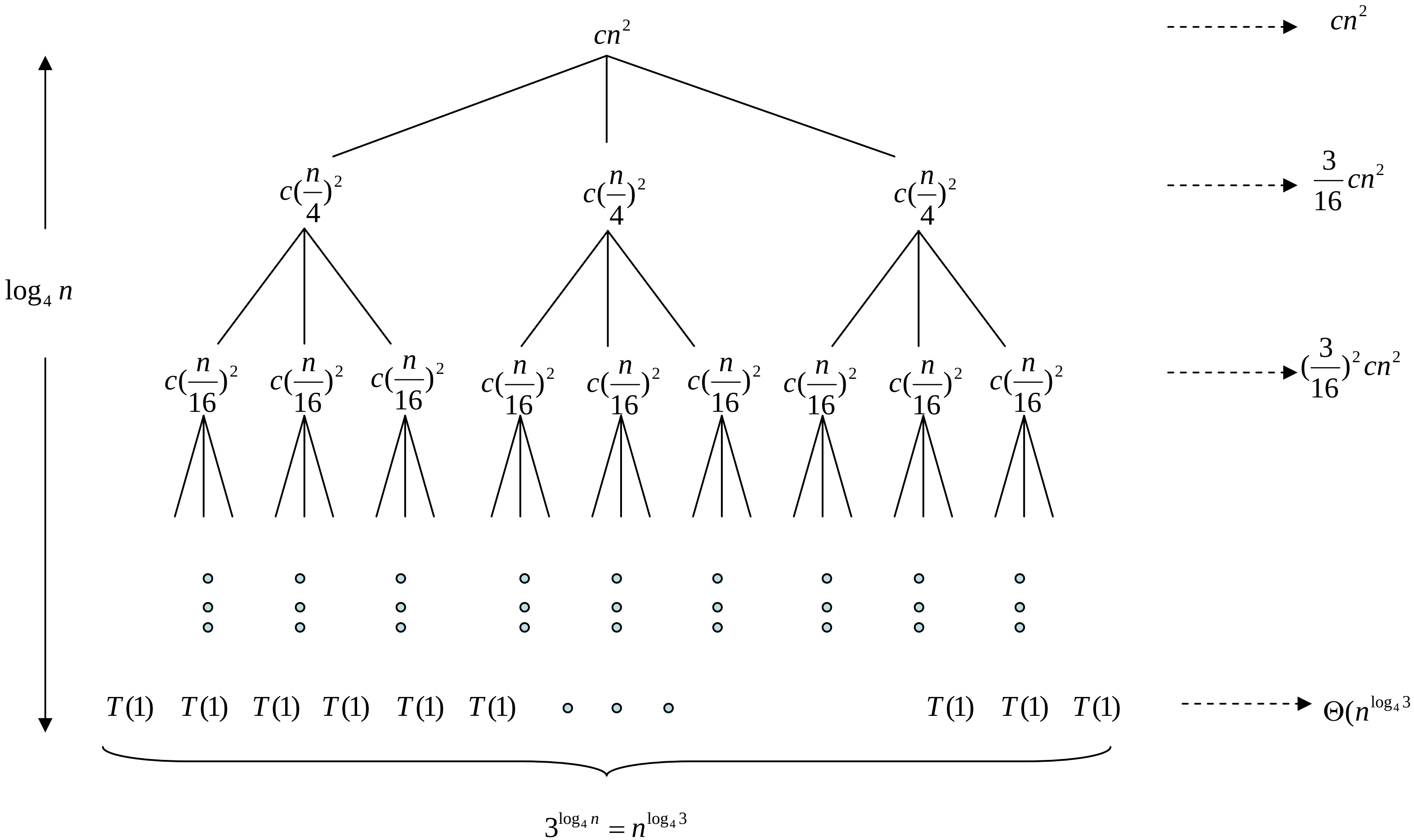
Some sloppiness we use here

- assume n is an exact power of 4 to remove the floor function
- replace $\Theta(n^2)$ by cn^2

Constructing the recursion tree



Constructing the recursion tree

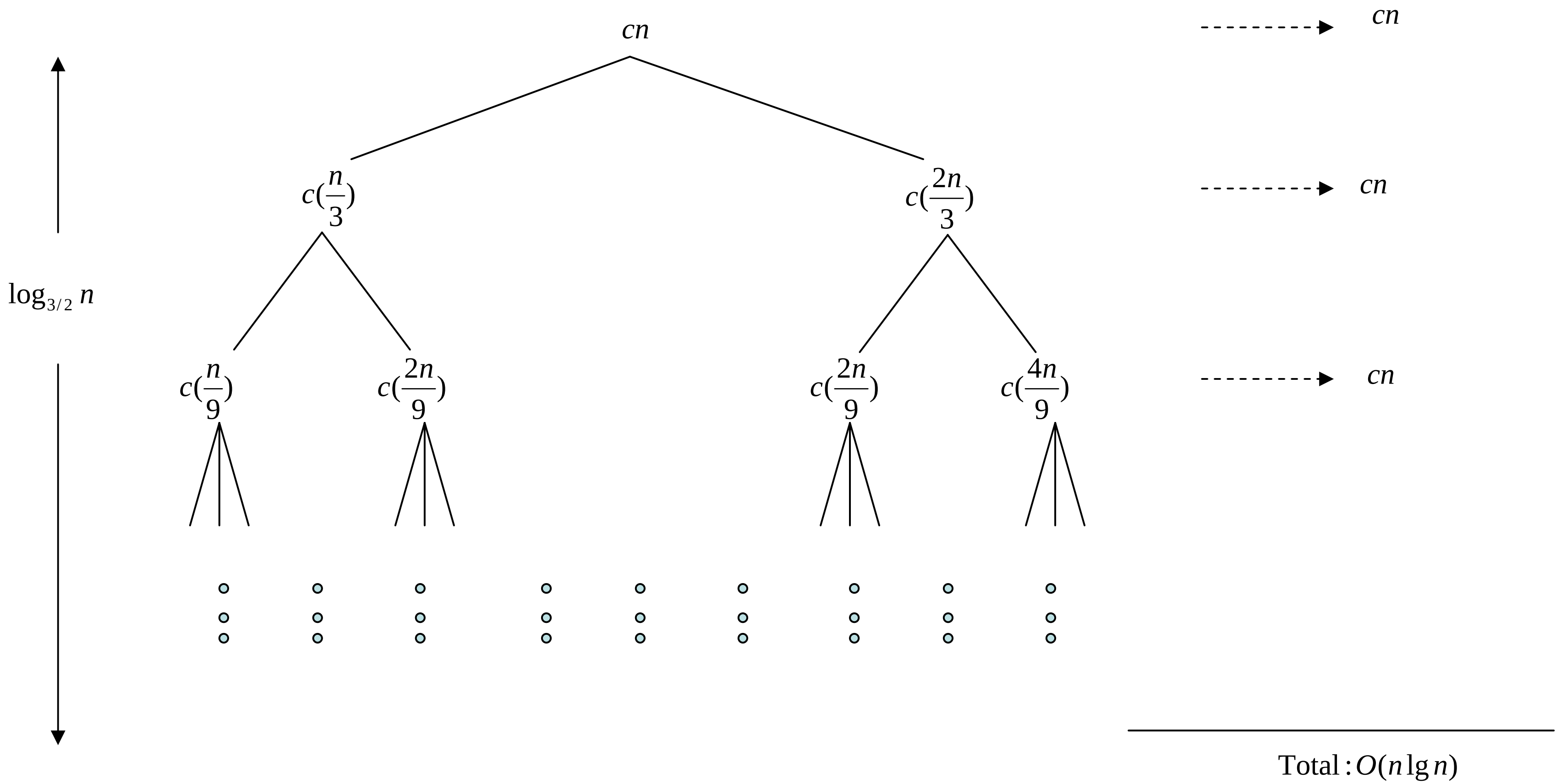


Total : $O(n^2)$

The sum of per-level costs results below:

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + ... + (\frac{3}{16})^{log_4 n - 1} cn^2 + 3^{log_4 n}\Theta(1) \\
&= \sum_{i=0}^{log_4 n - 1} (\frac{3}{16})^i cn^2 + \Theta(n^{log_4 3}) \\
&< \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{log_4 3}) \\
&= \frac{1}{1 - 3/16} cn^2 + \Theta(n^{log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

Recursion tree for T(n) = T(n/3)+T(2n/3)+cn



This is not a complete binary tree

There are fewer than $2^{\log_{3/2} n} = n^{\log_{3/2} 2} \in w(n \log n)$ leaves

Master Theorem: a simple version

Let $T : N \to R^+$ be an eventually non-decreasing function such that

$$T(n) = lT(n/b) + cn^k, n > n_0$$

when n/n_0 is an exact power of b. The constants $n_0, l \geq 1, b \geq 2$, and $k \geq 0$ are all integers. c is a positive real number.

We have

$$T(n) \in \begin{cases} \Theta(n^k) & if \quad k > log_b l \\ \Theta(n^k logn) & if \quad k = log_b l \\ \Theta(n^{log_b l}) & if \quad k < log_b l \end{cases}$$

Asymptotic recurrences

Consider a function $T : N \rightarrow R^+$ such that

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

for all sufficiently large n, where $l \geq 1$ and $b \geq 2$ are constants, and $f(n) \in \Theta(n^k)$ for some $k \geq 0$. We conclude that

$$T(n) \in \begin{cases} \Theta(n^k) & if \quad k > log_b l \\ \Theta(n^k \log n) & if \quad k = log_b l \\ \Theta(n^{\log_b l}) & if \quad k < log_b l \end{cases}$$

Master Theorem

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

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

where we interpret n/b be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. The $T(n)$ can be bound asymptotically as follows.

1. If $f(n) = O(n^{log_b l - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{log_b l})$.
2. If $f(n) = \Theta(n^{log_b l})$, then $T(n) = \Theta(n^{log_b l} lgn)$.
3. If $f(n) = \Omega(n^{log_b l + \epsilon})$ for some constant $\epsilon > 0$, and if $lf(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Examples

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

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

$$T(n) = 9T(n/3) + n.$$

$$T(n) = 3T(n/4) + nlgn.$$

Change variables

Consider the recurrence $T(n) = 2T(\sqrt{n}) + lgn$.

Let $n = 2^m$, then $T(2^m) = 2T(2^{m/2}) + m$

Let $S(m)$ denote $T(2^m)$, then $S(m) = 2S(m/2) + m$

Using the master method, $S(m) \in \Theta(mlgm)$ So $T(n) \in \Theta(lgnlglgn)$.

Range transformations

Consider the following recurrence where n is a power of 2.

$$T(n) = \begin{cases} 1/3, & n = 1 \\ nT^2(n/2) & otherwise \end{cases}$$

Let t_i denote $T(2^i)$.

$$t_i = T(2^i) = 2^i T^2(2^{i-1}) = 2^i t_{i-1}^2$$

Let u_i denote lgt_i.

$$u_i = lgt_i = lg(2^i t_{i-1}^2) = i + 2lgt_{i-1} = i + 2u_{i-1}$$

We solve this equation using recursion tree.