Recurrences

Definition

- A ***recurrence*** is an equation or inequality that describes a function in terms of its value on smaller inputs.
- Example from MERGE-SORT

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

Technicalities

- Normally, independent variables only assume integral values
- Example from MERGE-SORT revisited

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

- For simplicity, ignore floors and ceilings – often insignificant

Technicalities

- Boundary conditions (small *n*) are also glossed over

$$T(n) = \quad 2T(n/2) + \Theta(n)$$

❿ Value of T(n) assumed to be small constant for small n

Recursion-Tree Method

- Straightforward technique of coming up with a good guess
- Can help the Substitution Method
- ***Recursion tree***: visual representation of recursive call hierarchy where <u>each node represents the cost of a single subproblem</u>

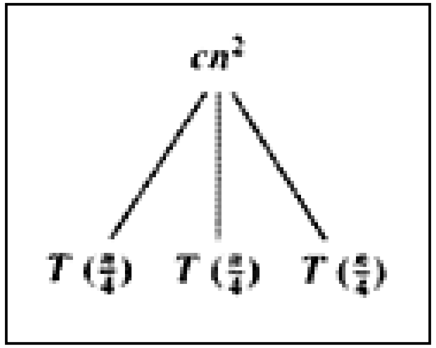
Recursion-Tree Method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

$$\boxed{T(n)}$$

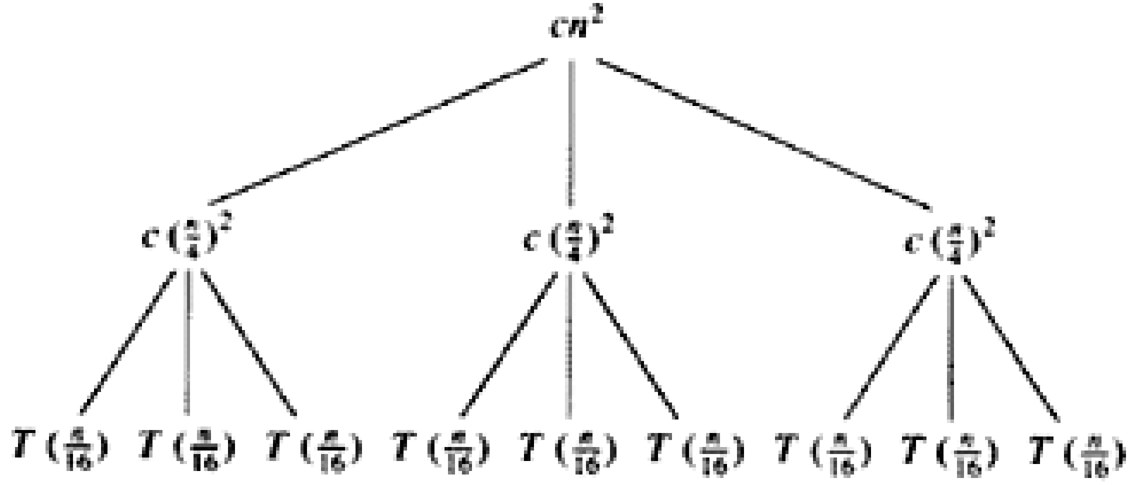
Recursion-Tree Method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$



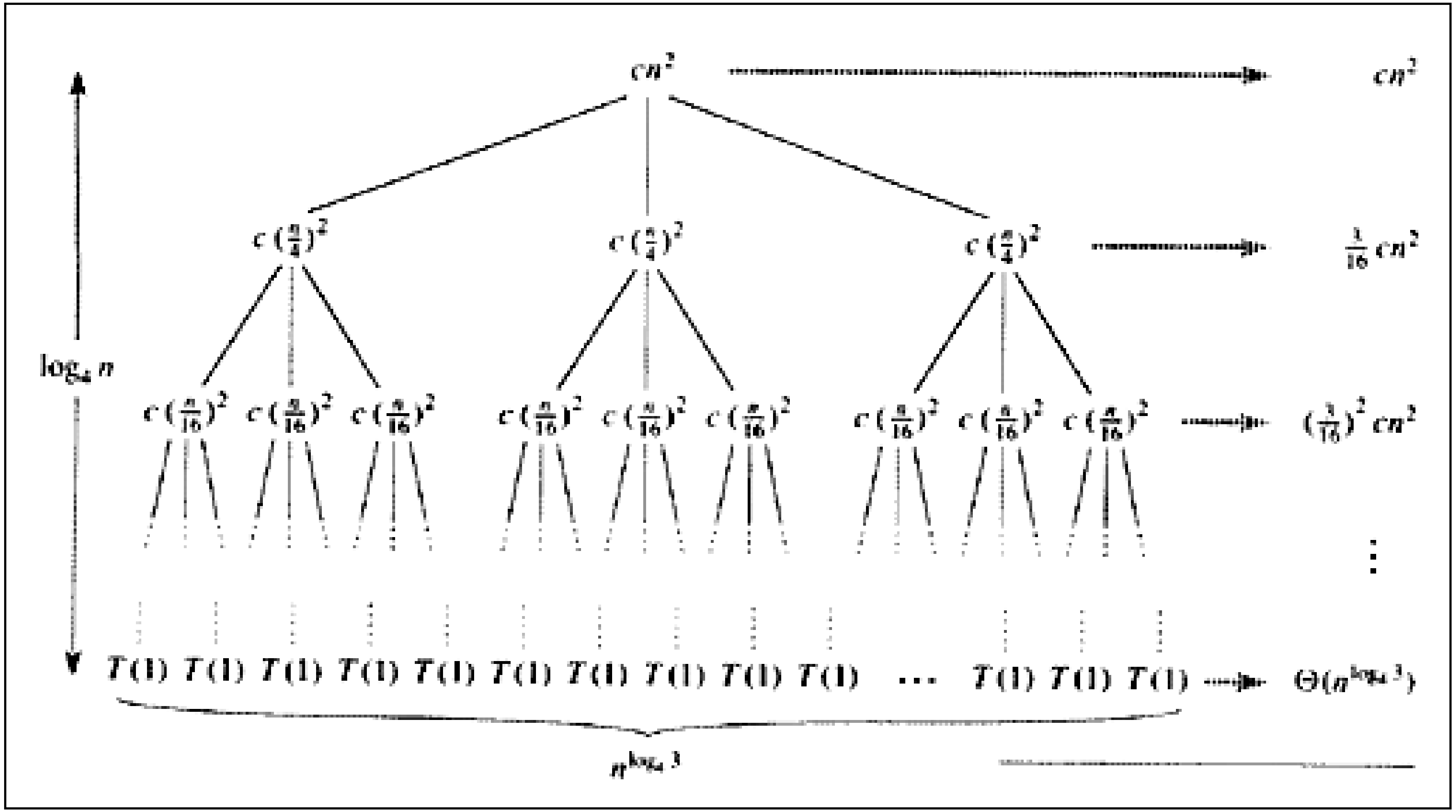
Recursion-Tree Method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$



(c)

Recursion-Tree Method



Recursion-Tree Method

- Gathering all the costs together:

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

$$T(n) \le \sum_{i=0}^{\infty} (3/16)^i cn^2 + o(n)$$

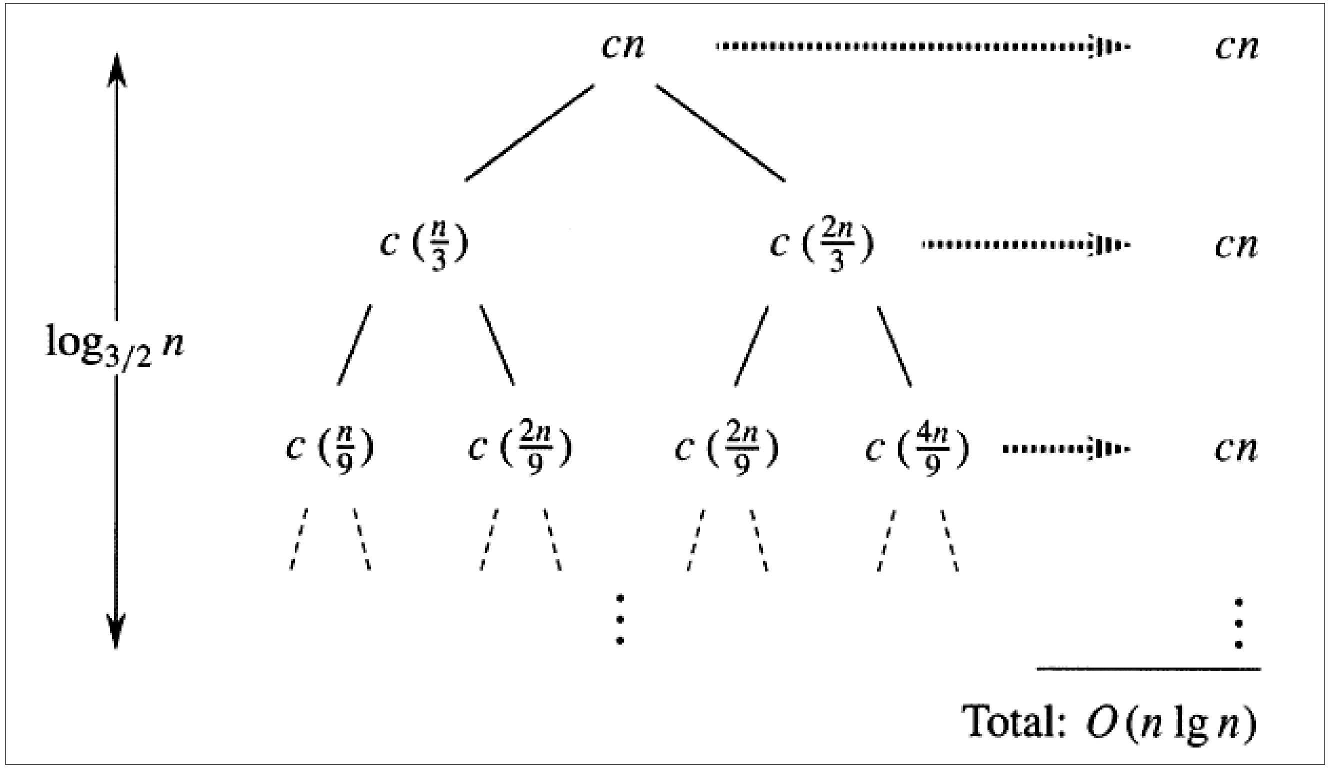
$$T(n) \le (1/(1-3/16))cn^2 + o(n)$$

$$T(n) \le (16/13)cn^2 + o(n)$$

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

Recursion-Tree Method

$$T(n) = T(n/3) + T(2n/3) + O(n)$$



Recursion-Tree Method

- An overestimate of the total cost:

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

- Counter-indications:

$$T(n) = O(n \lg n) + \omega(n \lg n)$$

- Notwithstanding this, use as "guess":

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

Substitution method

The most general method:

1. ***Guess*** the form of the solution.
2. ***Verify*** by induction.
3. ***Solve*** for constants.

Example: $T(n) = 4T(n/2) + 100n$

- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \le ck^3$ for $k < n$.
- Prove $T(n) \le cn^3$ by induction.

Example of substitution

$$\begin{aligned}
T(n) &= 4T(n/2) + 100n \\
&\leq 4c(n/2)^3 + 100n \\
&= (c/2)n^3 + 100n \\
&= cn^3 - ((c/2)n^3 - 100n) \quad \leftarrow \textit{desired} - \textit{residual} \\
&\leq cn^3 \quad \leftarrow \textit{desired}
\end{aligned}$$

whenever $(c/2)n^3 - 100n \geq 0$, for example, if $c \geq 200$ and $n \geq 1$.

residual

Example (continued)

- We must also handle the initial conditions, that is, ground the induction with base cases.
- ***Base:*** $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
- For $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick c big enough.

This bound is not tight!

A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$T(n) = 4T(n/2) + 100n$$
$$\le \; cn^2 + 100n$$
$$\le cn^2$$

for ***no*** choice of $c > 0$. Lose!

A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

Inductive hypothesis: $T(k) \le c_1k^2 - c_2k$ for $k < n$.

$$\begin{aligned} T(n) &= 4T(n/2) + 100n \\ &\le 4(c_1(n/2)^2 - c_2(n/2)) + 100n \\ &= c_1n^2 - 2c_2n + 100n \\ &= c_1n^2 - c_2n - (c_2n - 100n) \\ &\le c_1n^2 - c_2n \quad \text{if } c_2 > 100. \end{aligned}$$

Pick c_1 big enough to handle the initial conditions.

The master method

The master method applies to recurrences of the form

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

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

Three common cases

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

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

Solution: $T(n) = \Theta(n^{\log_b a})$.

Examples

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

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

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

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

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

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

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

2. $f(n) = \Theta(n^{\log_b a})$
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

Solution: $T(n) = \Theta(n^{\log_b a} \lg n)$.

Examples

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

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

CASE 2: $f(n) = \Theta(n^2).$

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

Ex. $T(n) = T(n/2) + \mathrm{c}$ (Binary search)

Ex. $T(n) = 2T(n/2) + \mathrm{n}$

Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that

$$a f(n/b) \le c f(n) \text{ for some constant } c < 1.$$

Solution: $T(n) = \Theta(f(n))$.

Examples

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

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

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(n/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

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

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

Examples

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

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

Master method does not apply.

In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\lg n)$.

Reference:

https://courses.csail.mit.edu/6.046/spring04/lectures/l2.ppt