Design and Analysis of Algorithms

Recurrence

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: 1684350406@qq.com

Topics

- **Induction**
- **Substitution Method**
- **Recursion-Tree Method**
- **Master Method**

Induction

Induction used to prove that a statement T(n) holds for all integers n:

- Base case: prove T(0)
- Assumption: assume that T(n-1) is true
- Induction step: prove that T(n-1) implies T(n) for all *n>0*

Strong induction: when we assume T(k) is true for ***all*** $\boldsymbol{k \leq n-1}$ and use this in proving T(n)

Integer Multiplication

Let X and Y be n bit integers. $X = \boxed{A|B}$ and $Y = \boxed{C|D}$ where A, B, C, and D are n/2 bit integers.

Simple Method: $XY = (A2^{\frac{n}{2}} + B)(C2^{\frac{n}{2}} + D)$

$$= AC2^{n} + (AD + BC)2^{\frac{n}{2}} + BD$$

Running Time Recurrence: $T(n) = 4T\left(\frac{n}{2}\right) + bn$

How do we solve it?

Induction

The most general strategy:

Guess: the form of the solution.

Verify: by induction.

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

Base case $T(1) = \Theta(1)$.

Guess $O(n^3)$.

Assume that $T(k) \leq ck^3$ for $k < n$.

Prove $T(n) \leq cn^3$ by induction.

Induction

$$T(n) = 4T\left(\frac{n}{2}\right) + bn$$
$$\le 4c\left(\frac{n}{2}\right)^3 + bn$$
$$= \left(\frac{c}{2}\right)n^3 + bn$$
$$= cn^3 - \left(\left(\frac{c}{2}\right)n^3 - bn\right)$$
$$\le cn^3$$

$\boxed{T(k) \le ck^3 \text{ for } k < n}$

For example, if $c \ge 2b$, then $\left(\frac{c}{2}\right)n^3 - bn \ge 0$.

This bound is not tight!

Induction

We also try that $T(n) = O(n^2)$.

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

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + bn \\
&\leq 4c\left(\frac{n}{2}\right)^2 + bn \\
&= cn^2 + bn \\
&\leq cn^2 \text{ X}
\end{aligned}
$$

A Tighter Upper Bound

Strengthen the inductive hypothesis.

Subtract a low-order term.

Inductive hypothesis: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + bn \\
&\le 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\left(\frac{n}{2}\right)\right) + bn \\
&= c_1 n^2 - 2c_2 n + bn \\
&= c_1\, n^2 - c_2 n - (c_2 n - bn) \\
&\le c_1 n^2 - c_2 n
\end{aligned}
$$

$\boxed{T(n) = O(n^2)}$

For example, if $c_2 \ge b$, then $c_2 n - bn \ge 0$.

Example of Substitution

Use algebraic manipulation to make an unknown recurrence similar to what you have seen before.

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

Set $m = \log n$ and we have $T(2^m) = 2T(2^{m/2}) + m$

Set $S(m) = T(2^m)$ and we have $S(m) = 2S(m/2) + m$

$\rightarrow S(m) = O(m \log m)$

As a result, we have $T(n) = O(\log n \log \log n)$

A Useful Recurrence Relation

- $T(n) =$ max number of compares to Merge-Sort a list of size $\leq n$
- $T(n)$ is monotone nondecreasing.

Merge-Sort recurrence

$$T(n) \leq \begin{cases} 0, & if\ n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n, & otherwise \end{cases}$$

Solution. $T(n)\ is\ O(nlogn)$

Assorted proofs. We describe several ways to solve this recurrence. Initially we assume n is a power of 2 and replace "≤" with "=" in the recurrence.

Proof by Induction

If $T(n)$ satisfies the following recurrence, then $T(n)\ is\ O(nlogn)$.

$$T(n) = \begin{cases} 0, & if\ n = 1 \\ 2T(n/2) + n, & otherwise \end{cases}$$

assuming n is a power of 2

- Base case: when $n = 1, T(1) = 0 = nlogn$.
- Inductive hypothesis: assume $T(n) = nlogn$.
- Goal: show that $T(2n) = 2nlog(2n)$

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2nlogn + 2n \\ &= 2n(\log(2n) - 1) + 2n \\ &= 2nlog(2n) \end{aligned}$$

Analysis of Merg-Sort Recurrence

If $T(n)$ satisfies the following recurrence, then $T(n) \le n\lceil logn \rceil$.

$$T(n) \le \begin{cases} 0, & if\ n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n, & otherwise \end{cases}$$

- Base case: n=1, $T(1) = 0$.
- Define： $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$.
- Induction step: assume true for 1, 2, …, n-1.

$$\boxed{\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \log_2 n \rceil} / 2 \end{aligned}}$$

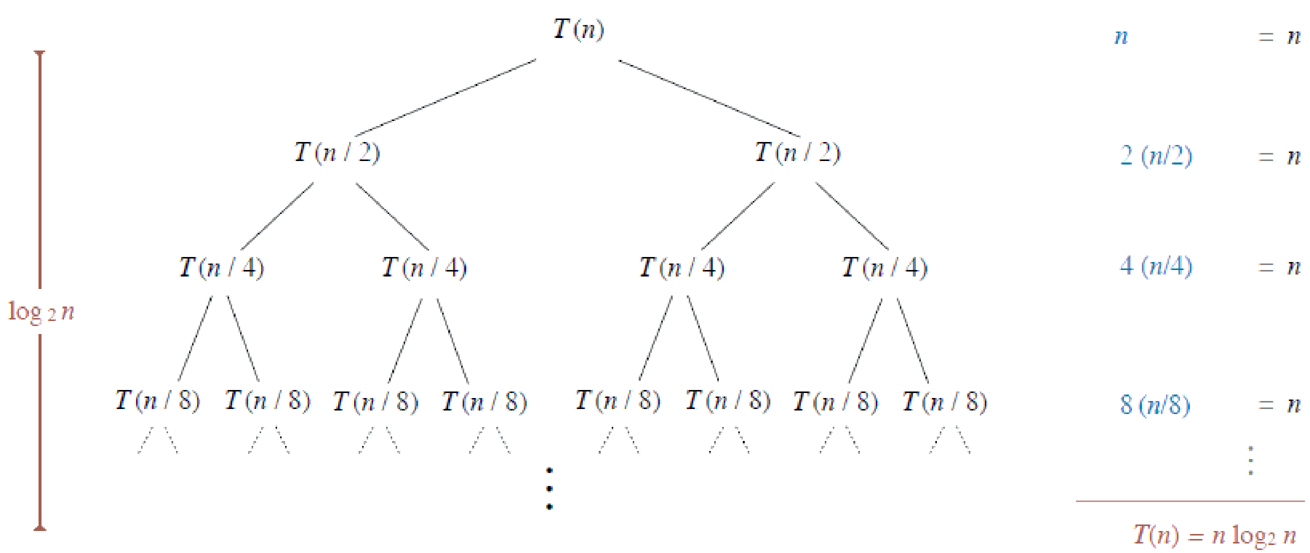
$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\le n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n \lceil \log_2 n_2 \rceil + n \longleftarrow \log_2 n_2 \le \lceil \log_2 n \rceil - 1 \\ &\le n(\lceil \log_2 n \rceil - 1) + n \\ &= n \lceil \log_2 n \rceil \end{aligned}$$

Recursion Tree

If $T(n)$ satisfies the following recurrence, then $T(n)\ is\ O(nlogn)$.

$$T(n) = \begin{cases} 0, & if\ n = 1 \\ 2T(n/2) + n, & otherwise \end{cases}$$

assuming n is a power of 2



Example of Recursion Tree

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

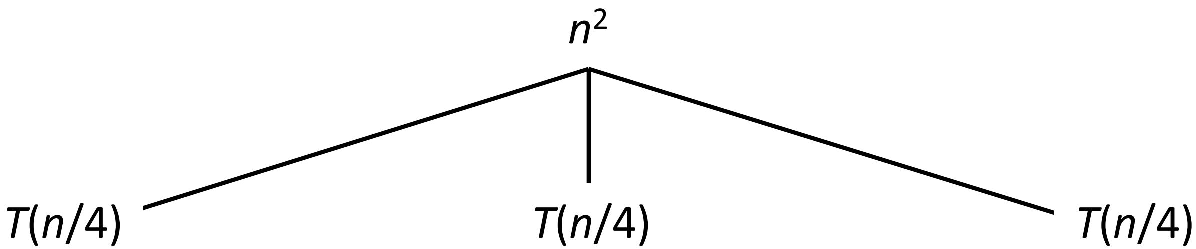
Example of Recursion Tree

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

$T(n)$

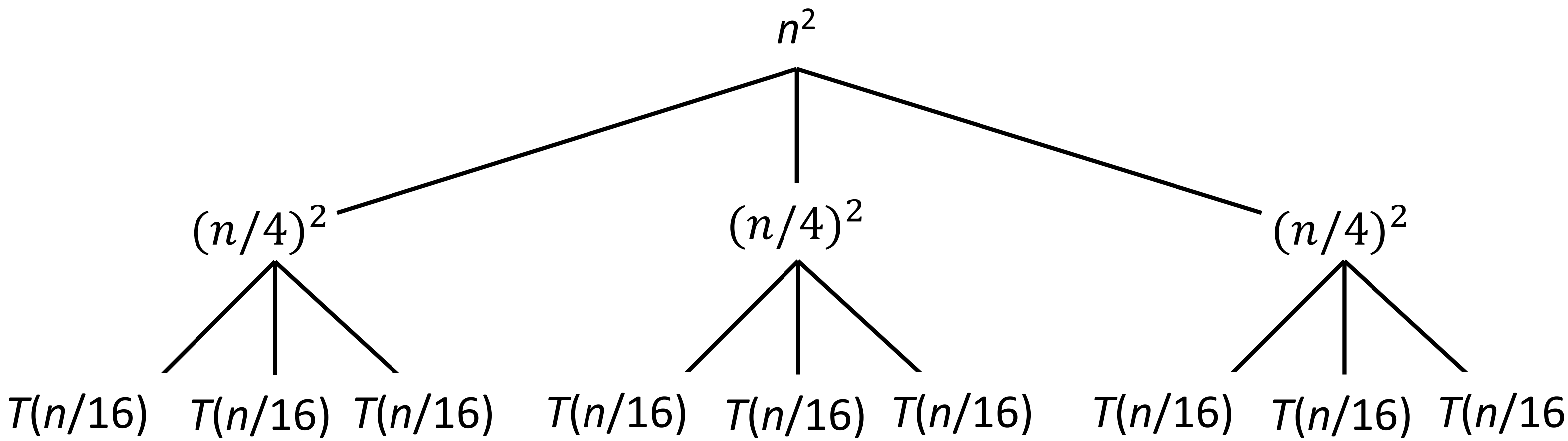
Example of Recursion Tree

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



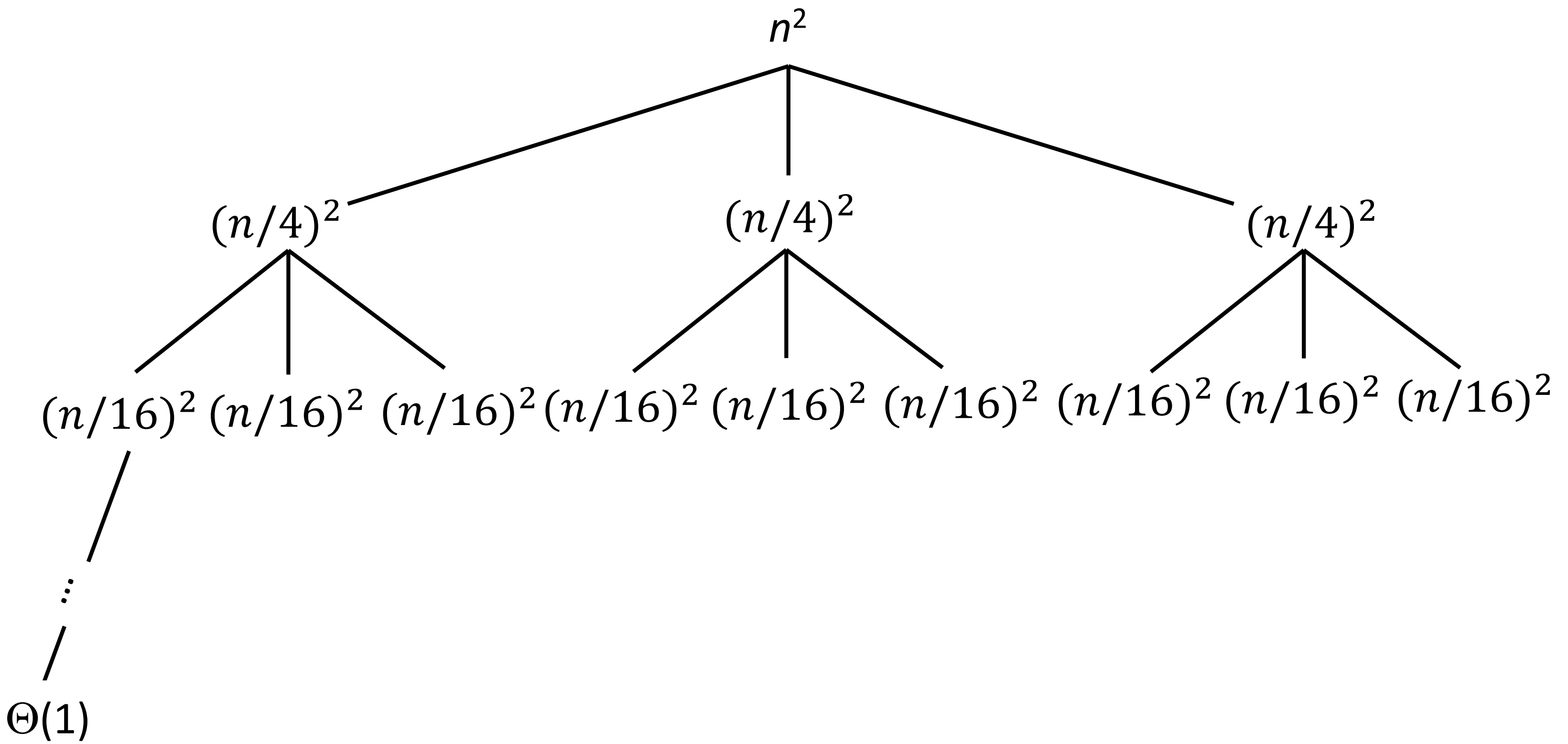
Example of Recursion Tree

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



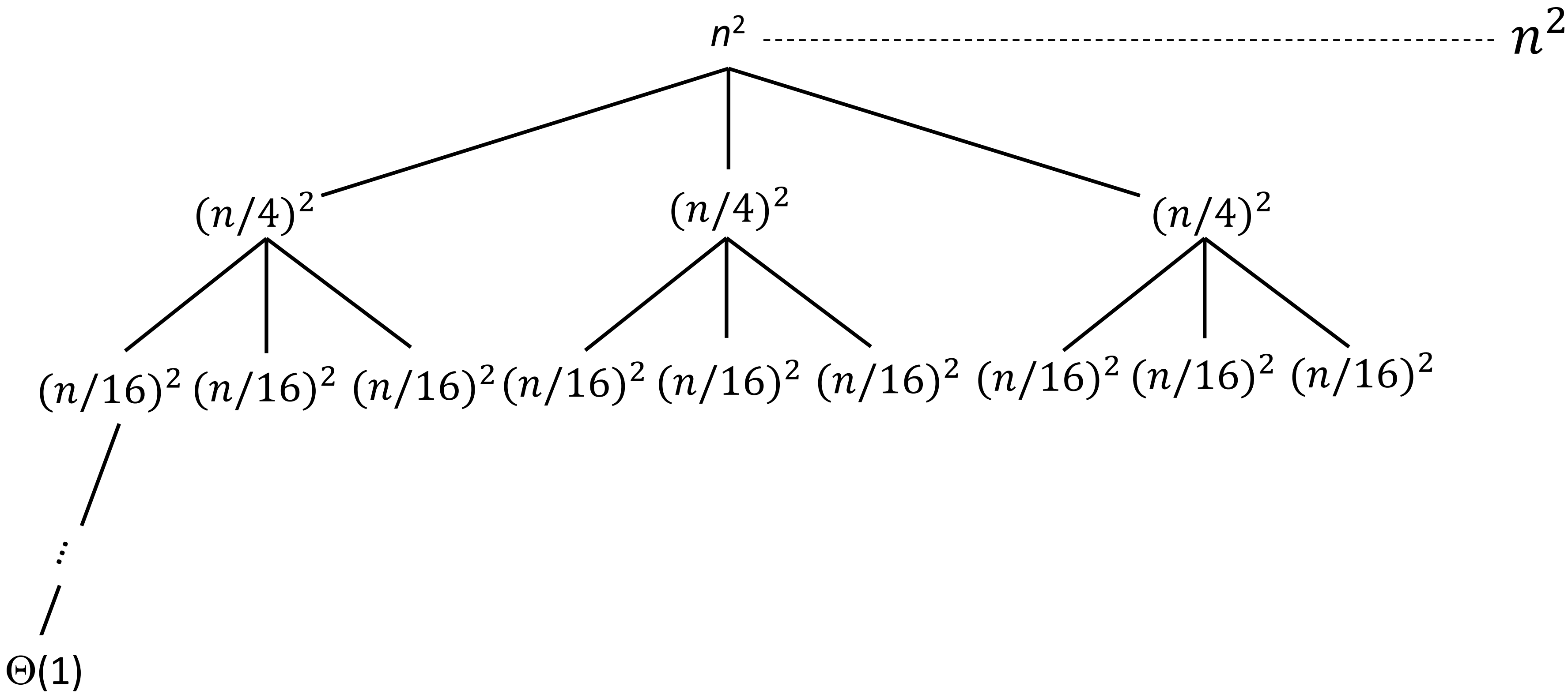
Example of Recursion Tree

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



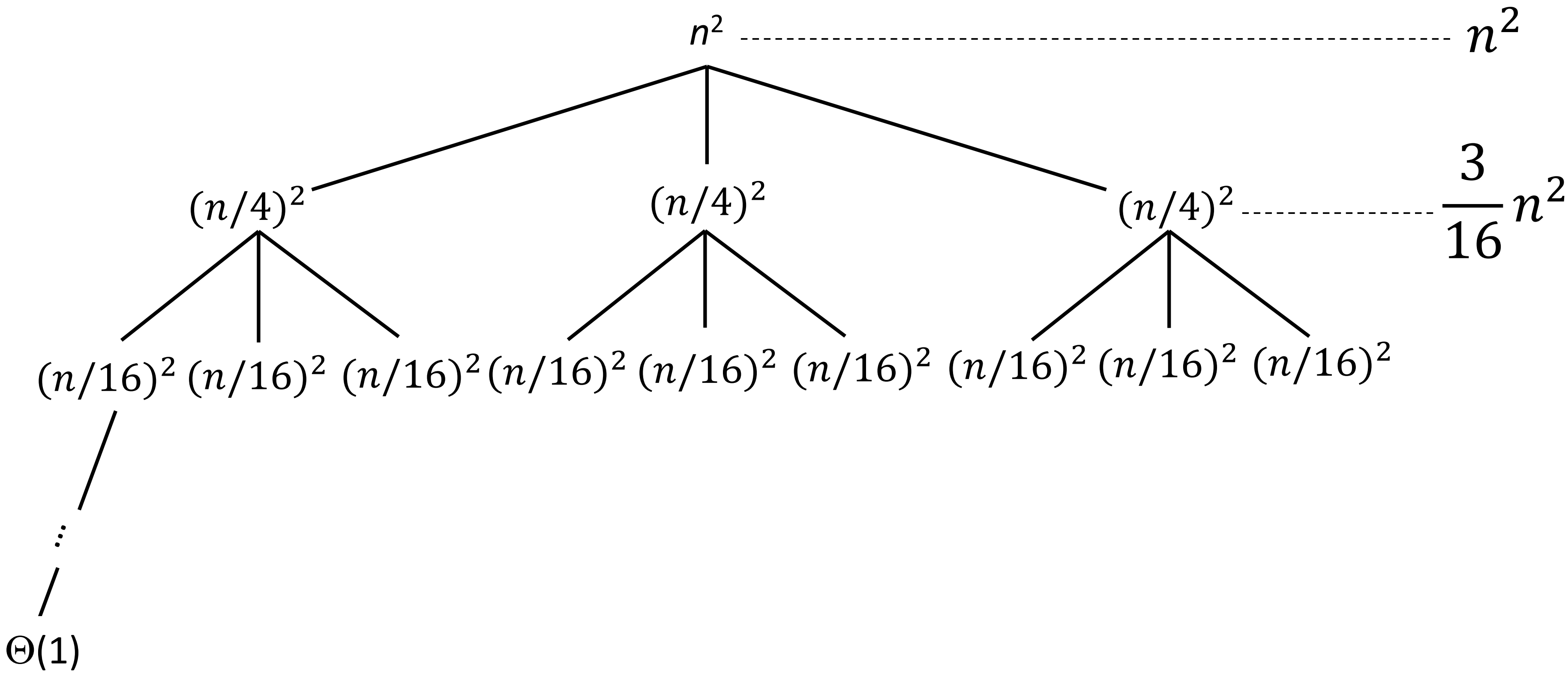
Example of Recursion Tree

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



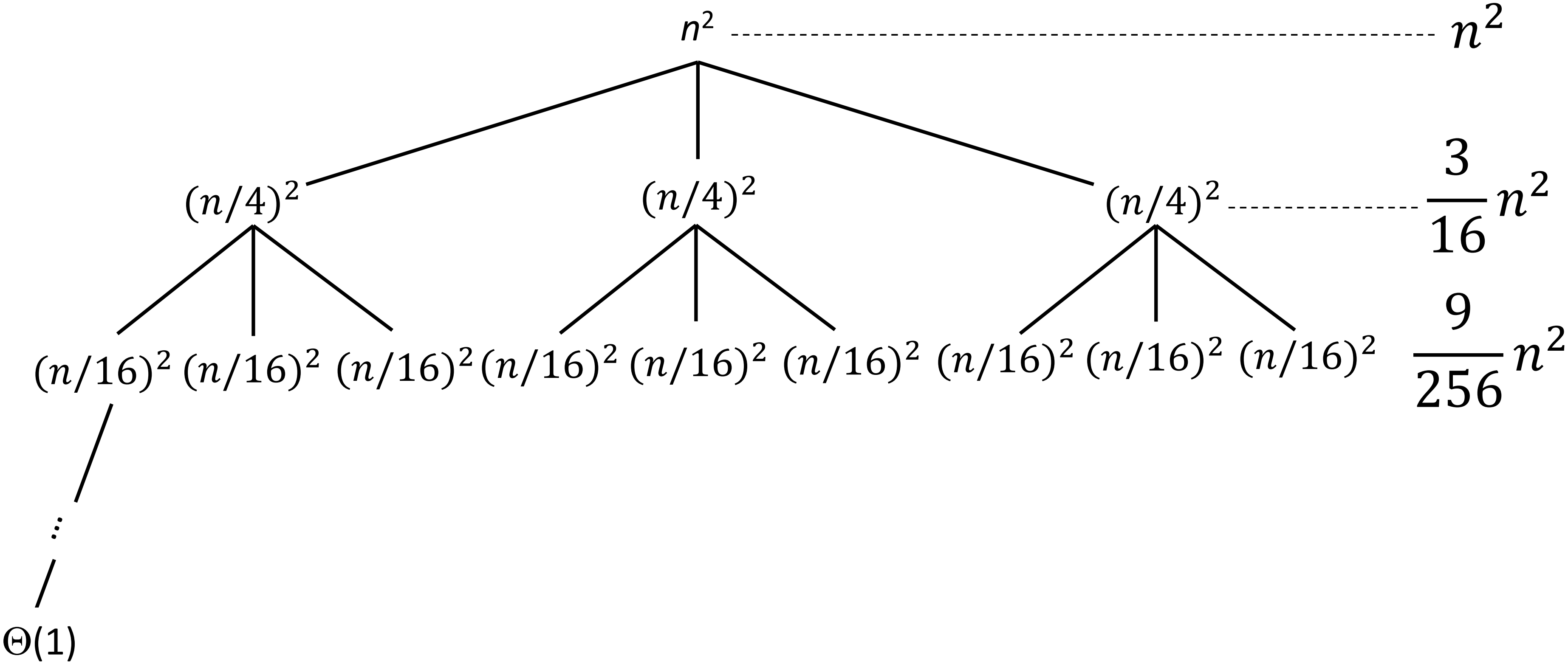
Example of Recursion Tree

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



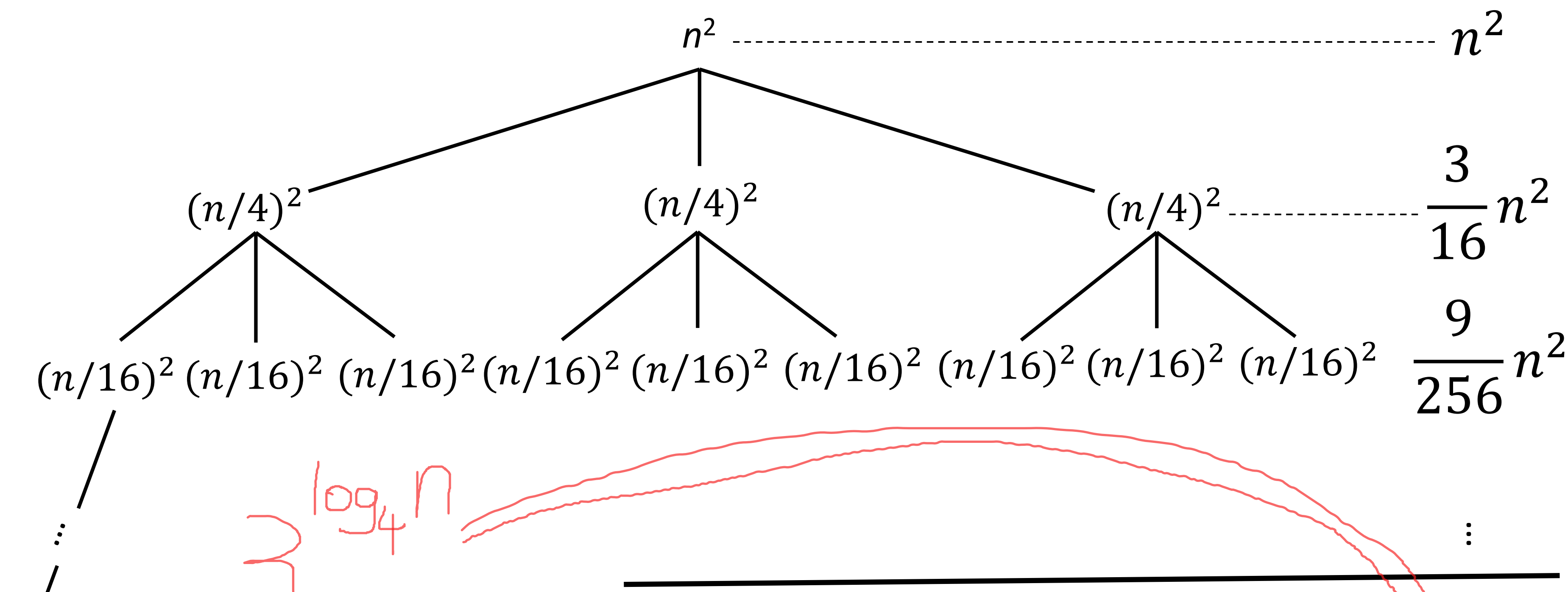
Example of Recursion Tree

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



Example of Recursion Tree

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



$$\text{Total} = n^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \cdots\right) + n^{\log_4 3}$$

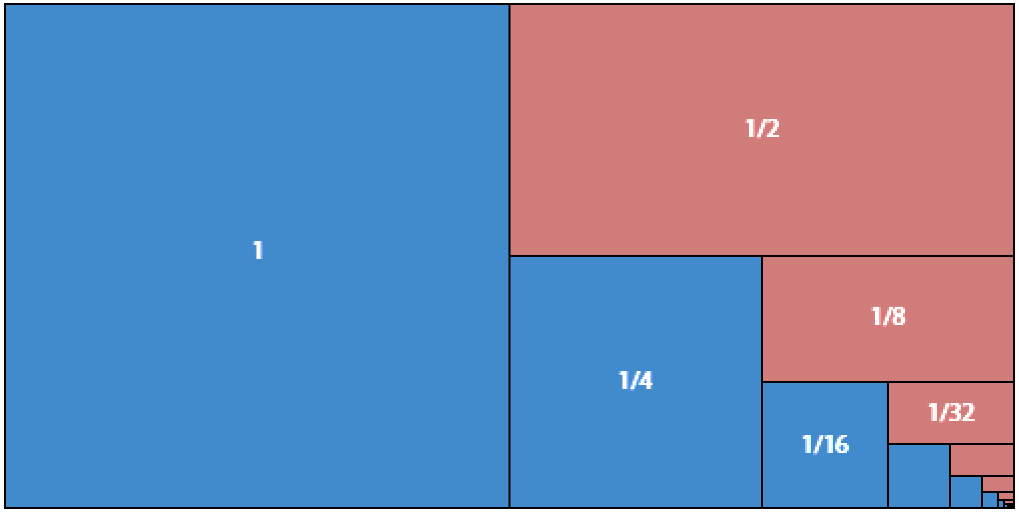
$$= \Theta(n^2)$$ *geometric series*

Geometric Series

Fact 1. For $r \neq 1$, $1 + r + r^2 + r^3 + \ldots + r^{k-1} = \dfrac{1 - r^k}{1 - r}$

Fact 2. For $r = 1$, $1 + r + r^2 + r^3 + \ldots + r^{k-1} = k$

Fact 3. For $r < 1$, $1 + r + r^2 + r^3 + \ldots = \dfrac{1}{1 - r}$



1 + 1/2 +1/4 + 1/8 + ... = 2

Master Method

Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

With $T(0) = 0\ and\ T(1) = \Theta(1)$.

Terms.

- $a \geq 1$ is the (integer) number of subproblems.
- $b > 1$ is the (integer) factor by which the subproblem size decreases.
- $f(n) =$ work to divide and combine subproblems.

Recursion tree.

- Number of levels:
- Number of subproblems at level i:
- Size of subproblem at level i:
- Number of leaves:

Master Method

Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

With $T(0) = 0 \ and \ T(1) = \Theta(1)$.

Terms.

- $a \geq 1$ is the (integer) number of subproblems.
- $b > 1$ is the (integer) factor by which the subproblem size decreases.
- $f(n) =$ work to divide and combine subproblems.

Recursion tree.

- Number of levels: $k = \log_b n$.
- Number of subproblems at level i: a^i.
- Size of subproblem at level i: n/b^i.
- Number of leaves: $n^{\log_b a}$.

Master Theorem

Master Theorem. Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

with $T(0) = 0 \; and \; T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $f(n) = O(n^k)$ for some constant $k < \log_b a$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

Ex. $T(n) = 3T(n/2) + 5n$
$a = 3, b = 2, f(n) = 5n, k = 1, \log_b a = 1.58$
$T(n) = \Theta\left(n^{\log_2 3}\right)$

Master Theorem

Master Theorem. Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

with $T(0) = 0\ and\ T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 2. If $f(n) = \Theta(n^k log^p n)$ for $p \geq 0$ and $k = \log_b a$, then $T(n) = \Theta\left(n^k log^{p+1} n\right)$.

多项式次数同⇒加logn

Ex. $T(n) = 2T(n/2) + 17n\log n$

$a = 2, b = 2, f(n) = 17n\log n\,, k = 1, p = 1, \log_b a = 1$

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

Master Theorem

Master Theorem. Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

with $T(0) = 0 \; and \; T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 3. If $f(n) = \Omega(n^k)$ for some constant $k > \log_b a$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\big(f(n)\big)$.

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

$a = 3, b = 2, f(n) = n^2, k = 2, \log_b a = 1.58$

Regularity condition: $3(n/2)^2 \leq cn^2$ for $c = 3/4$

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

Master Theorem

Master Theorem. Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

with $T(0) = 0$ *and* $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Case 1. If $f(n) = O(n^k)$ for some constant $k < \log_b a$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
Case 2. If $f(n) = \Theta(n^k log^p n)$ for $p \geq 0$ and $k = \log_b a$, then $T(n) = \Theta\left(n^k log^{p+1} n\right)$.
Case 3. If $f(n) = \Omega(n^k)$ for some constant $k > \log_b a$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

Master Theorem Need Not Apply

Gaps in master theorem

- Number of subproblems must be a constant.

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

- Number of subproblems must be ≥ 1.

$$T(n) = \frac{1}{2}T(n/2) + n^2$$

- Non-polynomial separation between $f(n)$ and $\log n$.

$$T(n) = 2T(n/2) + \frac{n}{logn}$$

- $f(n)$ is not positive.

$$T(n) = 2T(n/2) - n^2$$

- Regularity condition does not hold.

$$T(n) = T(n/2) + \mathrm{n}(2 - \cos n)$$