CSE 2202
Design and Analysis of Algorithms – I

Recurrence Relation

Recurrence Relations (1/2)

- A recurrence relation is an equation which is defined in terms of itself with smaller value.
- Why are recurrences good things?
  - Many natural functions are easily expressed as recurrences
- It is often easy to find a recurrence as the solution of a counting problem

Recurrence Relations (2/2)

- In both, we have general and boundary conditions, with the general condition breaking the problem into smaller and smaller pieces.
- The initial or boundary condition terminate the recursion.

Recurrence Equations

- A recurrence equation defines a function, **say T(n).**
- The function is defined recursively, that is, the function T(.) appear in its definition. (recall recursive function call).
- The recurrence equation should have a base case.

For example:

$$T(n) = \begin{cases} T(n-1)+T(n-2), & \text{if } n>1 \\ \underset{\uparrow\ \text{base case}}{1}, & \text{if } n=1 \text{ or } n=0. \end{cases}$$

for convenient, we sometime write the recurrence equation as:

$$T(n) = T(n-1)+T(n-2)$$

$$T(0) = T(1) = 1.$$

Recurrence Examples

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

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

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

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

More Recurrence equations:

$T(n) = 2 * T(n/2) + 1,$
$T(1) = 1.$

Base case;
initial condition.

$T(n) = T(n-1) + n,$
$T(1) = 1.$

Selection Sort

$T(n) = 2 * T(n/2) + n,$
$T(1) = 1.$

Merge Sort
Quick Sort (best case)

$T(n) = 2*T(n/2) + \log n,$
$T(1) = 1.$

Heap Construction

$T(n) = T(n/2) + 1,$
$T(1) = 0.$

Binary search

Methods for Solving Recurrences

- Master method
- Iteration method
- Substitution method
- Recursion tree method

The Master Method

- Based on the Master theorem.
- "Cookbook" approach for solving recurrences of the form

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

  - $a \geq 1$ is the number of sub-problems.
  - $b > 1$ is a constant, and is the factor by which the problem size is divided.
  - $f(n)$ is asymptotically positive. is the cost of the work done outside the recursive calls, often the cost of dividing the problem or merging the solutions.
  - n/b may not be an integer, but we ignore floors and ceilings.
- Requires memorization of three cases.

The Master Theorem

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

Theorem 4.1 (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

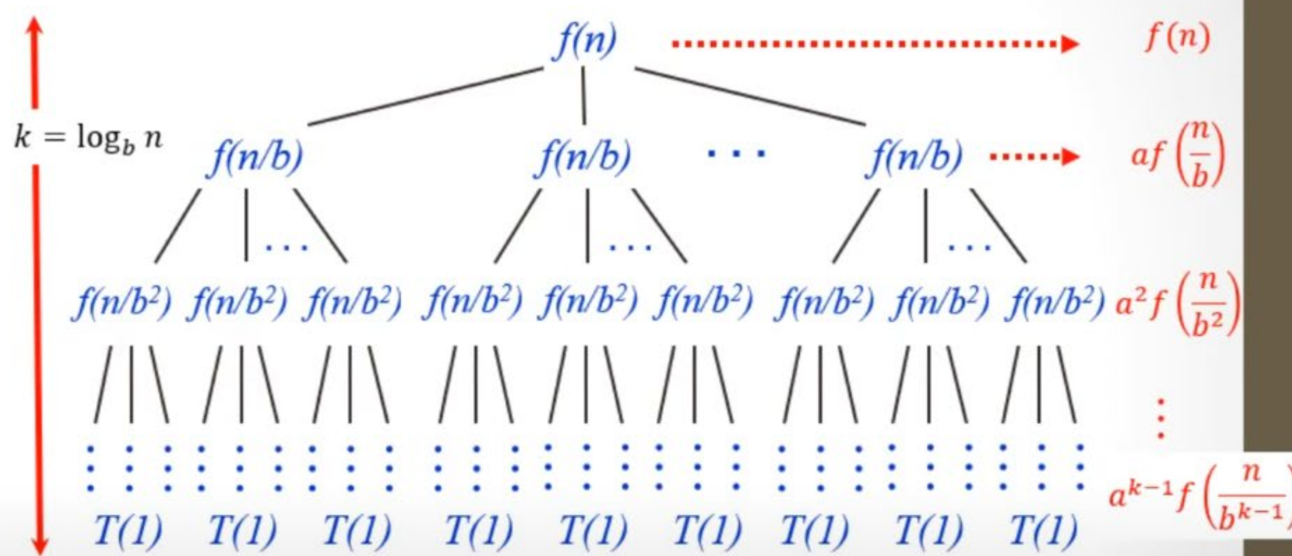
$$T(n) = aT(n/b) + f(n) ,$$

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 $T(n) = \Theta(f(n))$. ■

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

1. $T(n) = \Theta\left(n^{\log_b a}\right)$
2. $T(n) = \Theta\left(n^{\log_b a} \log_b n\right)$
3. $T(n) = \Theta(f(n))$



CASE 1:
Cost increases geometrically from the root to the leaves. $n^{\log_b a}$ is asymptotically larger in growth than $f(n)$ by a polynomial factor n^{ε}.

CASE 2:
Cost is approximately the same on each of the **$\log_b n$** levels.
The growth of $n^{\log_b a}$ is asymptotically equal to $f(n)$.

CASE 3:
Cost decreases geometrically from the root to the leaves. $n^{\log_b a}$ is asymptotically smaller in growth than $f(n)$ by a polynomial factor n^{ε}.

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

- The method compares $f(n)$ with $n^{\log_b a}$.
- The dominant function between the two determines the solution.
  - **Case 1**: If $n^{\log_b a}$ is larger, then $T(n) = \Theta(n^{\log_b a})$.
  - **Case 2**: If both functions are roughly the same size, the solution includes a logarithmic factor: $T(n) = \Theta(n^{\log_b a} \log n)$.
  - **Case 3**: If $f(n)$ is larger, then $T(n) = \Theta(f(n))$.

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

- The method compares $f(n)$ with $n^{\log_b a}$.
- The dominant function between the two determines the solution.
  - **Case 1**: If $n^{\log_b a}$ is larger, then $T(n) = \Theta(n^{\log_b a})$.
  - **Case 2**: If both functions are roughly the same size, the solution includes a logarithmic factor: $T(n) = \Theta(n^{\log_b a} \log n)$.
  - **Case 3**: If $f(n)$ is larger, then $T(n) = \Theta(f(n))$.

For Case 2:

- the algorithm's complexity is influenced not just by the **work at each level** but also by the **depth of the recursive decomposition**.
- The depth of the recursive tree for divide-and-conquer algorithms is logarithmic in nature.
- So, if you're doing logarithmic work at each level, and you have a logarithmic number of levels

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

- **Case 1**: $f(n)$ should be polynomially smaller than $n^{\log_b a}$.
  - Specifically, $f(n)$ must be smaller by a factor of n^{ϵ} for some constant $\epsilon > 0$.
- **Case 3**: $f(n)$ should be polynomially larger than $n^{\log_b a}$ and should satisfy a "regularity" condition.
  - The regularity condition is: $a \times f(n/b) \leq c \times f(n)$ for some constant c.
  - Most polynomially bounded functions we encounter satisfy this condition.

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

Limitations of the Master Method:

- The three cases don't cover all possible scenarios for $f(n)$.
  - Gap between Cases 1 and 2: If $f(n)$ is smaller than $n^{\log_b a}$ but not polynomially smaller.
  - Gap between Cases 2 and 3: If $f(n)$ is larger than $n^{\log_b a}$ but not polynomially larger.
- If $f(n)$ falls into one of these gaps, or if the regularity condition for Case 3 isn't met, the Master Method cannot be applied to the recurrence.

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ∎

$$T(n) = 2\,T\left(\frac{n}{2}\right) + n$$

$a = 2$

$b = 2$

$f(n) = n$

$n^{\log_b a} \Longrightarrow n^{\log_2 2} \Longrightarrow n$

$f(n) = n^{\log_b a}$ so case 2 is applied. $[f(n) = \Theta(n^{\log_b a})]$

$$T(n) = \Theta(n^{\log_b a} \lg n)$$
$$= \Theta(n^{\log_2 2} \lg n)$$
$$= \Theta(n \lg n)$$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$f(n) > n^{\log_b a}$ so case 3 is applied. $[f(n) = \Omega(n^{\log_b a+\varepsilon})]$

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

$$a = 2$$

$$b = 2$$

$$f(n) = n^2$$

$$n^{\log_b a} \Longrightarrow n^{\log_2 2} \Longrightarrow n$$

$$T(n) = \Theta(f(n))$$

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

Verify Regularity Condition:

$$a \cdot f\left(\frac{n}{b}\right) \leq cf(n)$$

$$2 \cdot f\left(\frac{n}{2}\right) \leq cn^2$$

$$2 \cdot \frac{n^2}{4} \leq cn^2$$

$$\frac{1}{2} \leq c$$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 9\,T\left(\frac{n}{3}\right) + n$$

$f(n) < n^{\log_b a}$ so case 1 is applied. $[f(n) = O(n^{\log_b a - \varepsilon})]$

$a = 9$

$b = 3$

$f(n) = n$

$n^{\log_b a} \Longrightarrow n^{\log_3 9} \Longrightarrow n^2$

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

$$= \Theta\left(n^{\log_3 9}\right)$$

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

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 4\,T\left(\frac{n}{2}\right) + n^3$$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

$f(n) > n^{\log_b a}$ so case 3 is applied. $[f(n) = \Omega(n^{\log_b a+\varepsilon})]$

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

$a = 4$

$b = 2$

$f(n) = n^3$

$n^{\log_b a} \Longrightarrow n^{\log_2 4} \Longrightarrow n^2$

Verify Regularity Condition:

$$a \cdot f\left(\frac{n}{b}\right) \leq cf(n)$$

$$4 \cdot f\left(\frac{n}{2}\right) \leq cn^3$$

$$4 \cdot \frac{n^3}{8} \leq cn^2$$

$$\frac{1}{2} \leq c$$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$a = 4$

$b = 2$

$f(n) = n^2$

$n^{\log_b a} \Longrightarrow n^{\log_2 4} \Longrightarrow n^2$

$f(n) = n^{\log_b a}$ so case 2 is applied. $[f(n) = \Theta(n^{\log_b a})]$

$$\begin{aligned} T(n) &= \Theta\left(n^{\log_b a} \lg n\right) \\ &= \Theta\left(n^{\log_2 4} \lg n\right) \\ &= \Theta(n^2 \lg n) \end{aligned}$$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 2\,T\left(\frac{n}{2}\right) + \sqrt{n}$$

$n^{1/2}$

$f(n) < n^{\log_b a}$ so case 1 is applied. $[f(n) = O(n^{\log_b a - \varepsilon})]$

$a = 2$

$b = 2$

$f(n) = n^{1/2}$

$n^{\log_b a} \Longrightarrow n^{\log_2 2} \Longrightarrow \boxed{n}$

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

$= \Theta(n^{\log_2 2})$

$\frac{f(n)}{n}$

$n^{-1/2}$

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\lg n}$$

$a = 4$

$b = 2$

$f(n) = n^2/\lg n$

$n^{\log_b a} \Longrightarrow n^{\log_2 4} \Longrightarrow n^2$

Non-polynomial difference between $\boxed{f(n) \text{ and } n^{\log_b a}}$. Master method does not apply.

The difference must be polynomially larger by a factor of n^{ε} where $\varepsilon > 0$.

In this case the difference is only larger by a factor of $1/\lg n$.

Theorem 4.1 (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 $T(n) = \Theta(f(n))$. ■

$$T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$$

$a = 2$

$b = 2$

$f(n) = n \lg n$

$n^{\log_b a} \Rightarrow n^{\log_2 2} \Rightarrow n$

Master method does not apply. Non-polynomial difference between $f(n)$ and $n^{\log_b a}$.

The difference must be polynomially larger by a factor of n^{ε} where $\varepsilon > 0$.

In this case the difference is only larger by a factor of $\lg n$.

Seems like case 3 should apply.

Master Method – Examples

- $T(n) = 16T(n/4)+n$
  - $a = 16$, $b = 4$, $n^{\log_b a} = n^{\log_4 16} = n^2$.
  - $f(n) = n = O(n^{\log_b a-\varepsilon}) = O(n^{2-\varepsilon})$, where $\varepsilon = 1 \Rightarrow$ **Case 1.**
  - Hence, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

- $T(n) = T(3n/7) + 1$
  - $a = 1$, $b=7/3$, and $n^{\log_b a} = n^{\log_{7/3} 1} = n^0 = 1$
  - $f(n) = 1 = \Theta(n^{\log_b a}) \Rightarrow$ **Case 2**.
  - Therefore, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

Master Method – Examples

- $T(n) = 3T(n/4) + n \lg n$
  - $a = 3$, $b=4$, thus $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
  - $f(n) = n \lg n = \Omega(n^{\log_4 3 + \varepsilon})$ where $\varepsilon \approx 0.2 \Rightarrow$ **Case 3**.
  - Therefore, $T(n) = \Theta(f(n)) = \Theta(n \lg n)$.

- $T(n) = 2T(n/2) + n \lg n$
  - $a = 2$, $b=2$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_2 2} = n$
  - ***f*(*n*)** is asymptotically larger than $n^{\log_b a}$, but not polynomially larger. The ratio $\lg n$ is asymptotically less than n^{ε} for any positive ε. Thus, the Master Theorem ***doesn't*** apply here.

Simplications:

- There are two simplifications we apply that won't affect asymptotic analysis
  - ignore floors and ceilings (justification in text)
  - assume base cases are constant, i.e., $T(n) = \Theta(1)$ for n small enough

Solving Recurrences: Iteration (convert to summation)

- Expand the recurrence
- Work some algebra to express as a summation
- Evaluate the summation

The Iteration Method

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

$$\begin{aligned} T(n) &= c + T(n/2) \\ &= c + c + T(n/4) \\ &= c + c + c + T(n/8) \end{aligned}$$

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

$$T(n/4) = c + T(n/8)$$

Assume $n = 2^k$

$$\begin{aligned} T(n) &= \underbrace{c + c + \ldots + c}_{\text{k times}} + T(1) \\ &= c\lg n + T(1) \\ &= \Theta(\lg n) \end{aligned}$$

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

- s(n) =

  c + s(n-1)

  c + c + s(n-2)

  2c + s(n-2)

  2c + c + s(n-3)

  3c + s(n-3)

  …

  kc + s(n-k) = ck + s(n-k)
- What if k = n?
  - s(n) = cn + s(0) = cn

Solving Recurrences: Iteration (convert to summation)

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

T(n) = 4T(n/2) + n /**expand**/

= 4(n/2 + 4T(n/4)) + n /**simplify**/

= 16T(n/4) + 2n + n /**expand**/

= 16(n/4 + 4T(n/8)) + 2n + n /**simplify**/

= $4^{\log n}$ T(1)+ … + 4n + 2n + n /** #levels = log n **/

$$= c4^{\log n} + n\sum_{k=0}^{\log n-1} 2^k$$ /** convert to summation**/

$$= cn^{\log 4} + n\left(\frac{2^{\log n} - 1}{2-1}\right)$$ /** $a^{\log b} = b^{\log a}$ **/

Solving Recurrences: Iteration (convert to summation) (cont.)

$= cn^2 + n(n^{\log 2} - 1)$ /** $2^{\log n} = n^{\log 2}$ **/

$= cn^2 + n(n - 1)$

$= cn^2 + n^2 - n$

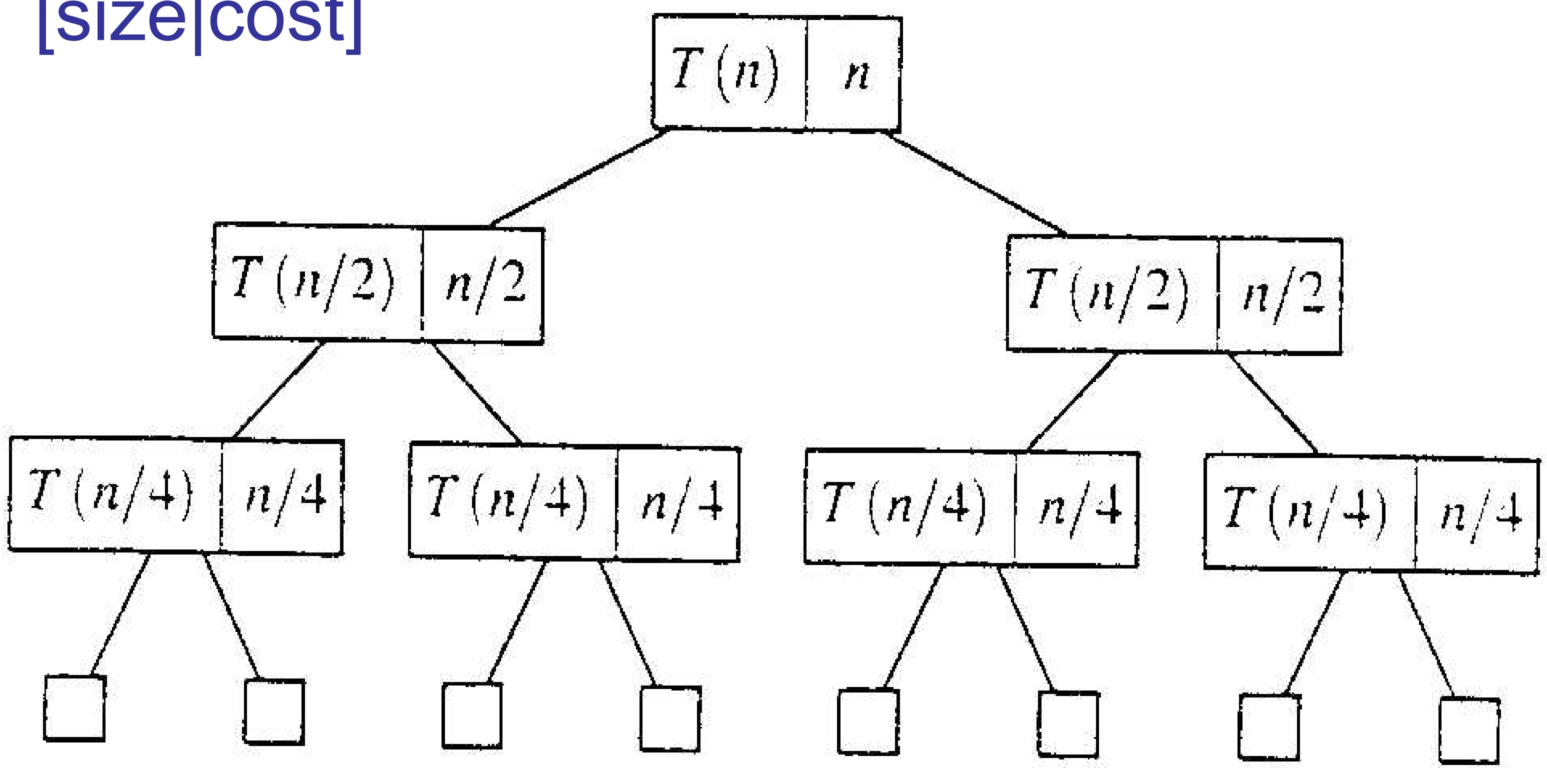
$= \Theta(n^2)$

Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion tree method is good for generating guesses for the substitution method.
- The recursion-tree method can be unreliable.
- The recursion-tree method promotes intuition, however.

Evaluate recursive equation using Recursion Tree

- Evaluate: T(n) = T(n/2) + T(n/2) + n
  - Work copy: T(k) = T(k/2) + T(k/2) + k
  - For k=n/2, T(n/2) = T(n/4) + T(n/4) + (n/2)
- [size|cost]



Recursion Tree e.g.

- To evaluate the total cost of the recursion tree
  - sum all the non-recursive costs of all nodes
  - = Sum (rowSum(cost of all nodes at the same depth))
- Determine the maximum depth of the recursion tree:
  - For our example, at tree depth d
    the size parameter is $n/(2^d)$
  - the size parameter converging to base case, i.e. case 1
  - such that, $n/(2^d) = 1$,
  - $d = \lg(n)$
  - The rowSum for each row is n
- Therefore, the total cost, $T(n) = n \lg(n)$

Example of recursion tree

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

Example of recursion tree

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

$$T(n)$$

Example of recursion tree

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

$$n^2$$

$$T(n/4) \qquad\qquad T(n/2)$$

Example of recursion tree

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

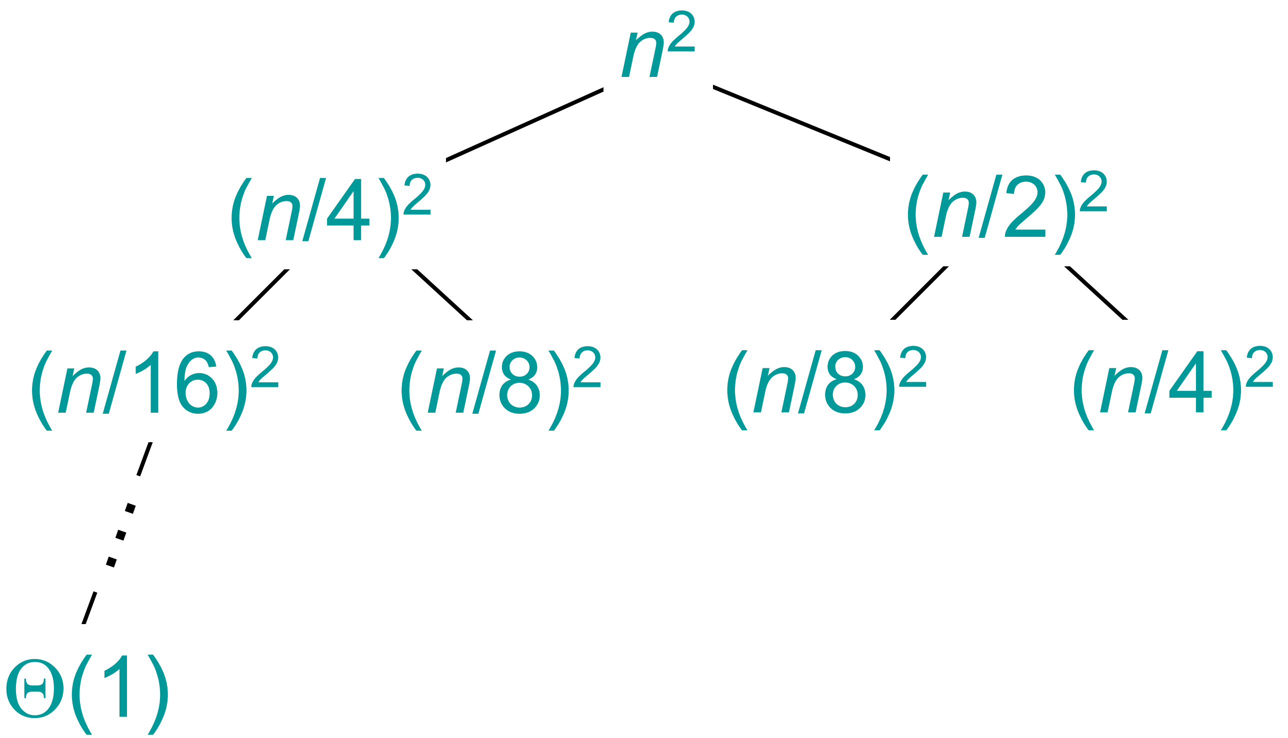
$$n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

$$T(n/16) \quad T(n/8) \quad T(n/8) \quad T(n/4)$$

Example of recursion tree

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



Example of recursion tree

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

$$n^2 \quad \cdots\cdots\cdots \quad n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

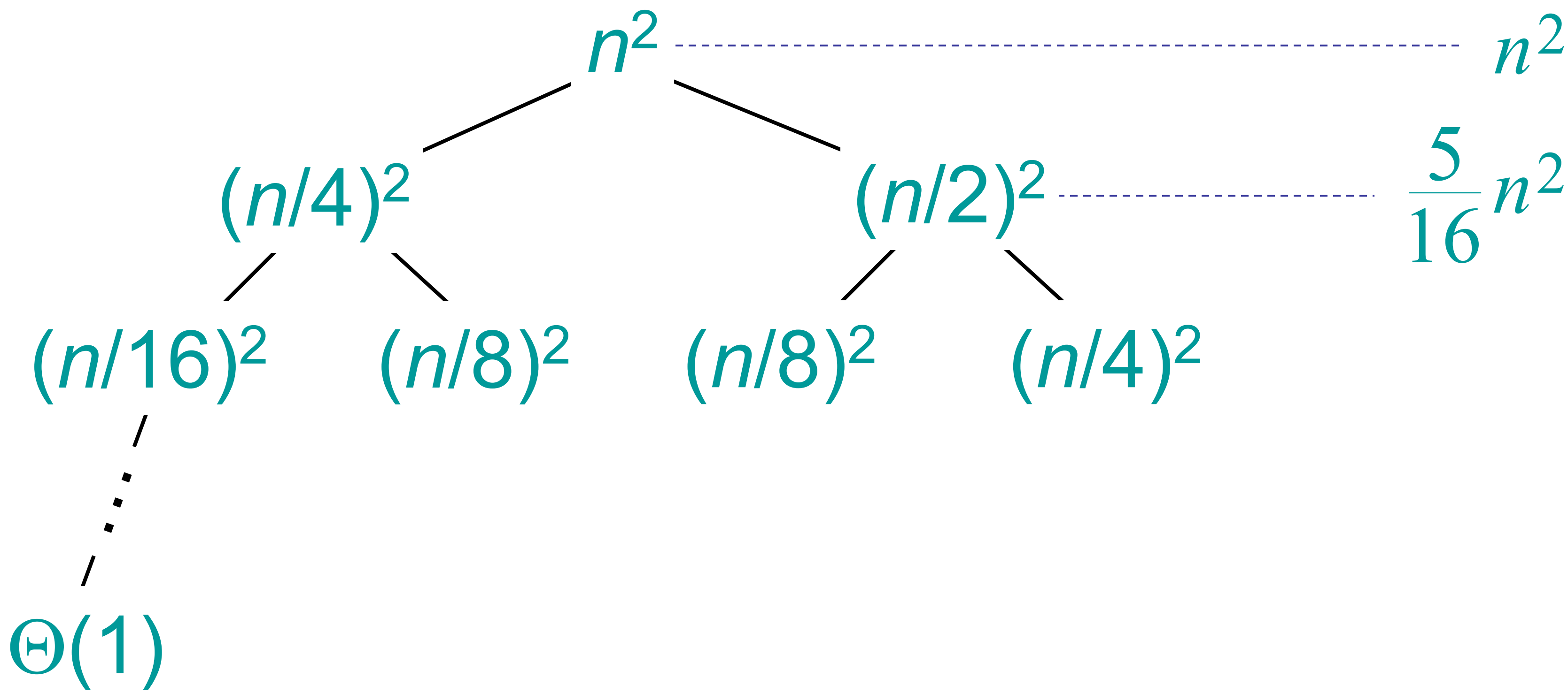
$$(n/16)^2 \quad (n/8)^2 \quad (n/8)^2 \quad (n/4)^2$$

$$\vdots$$

$$\Theta(1)$$

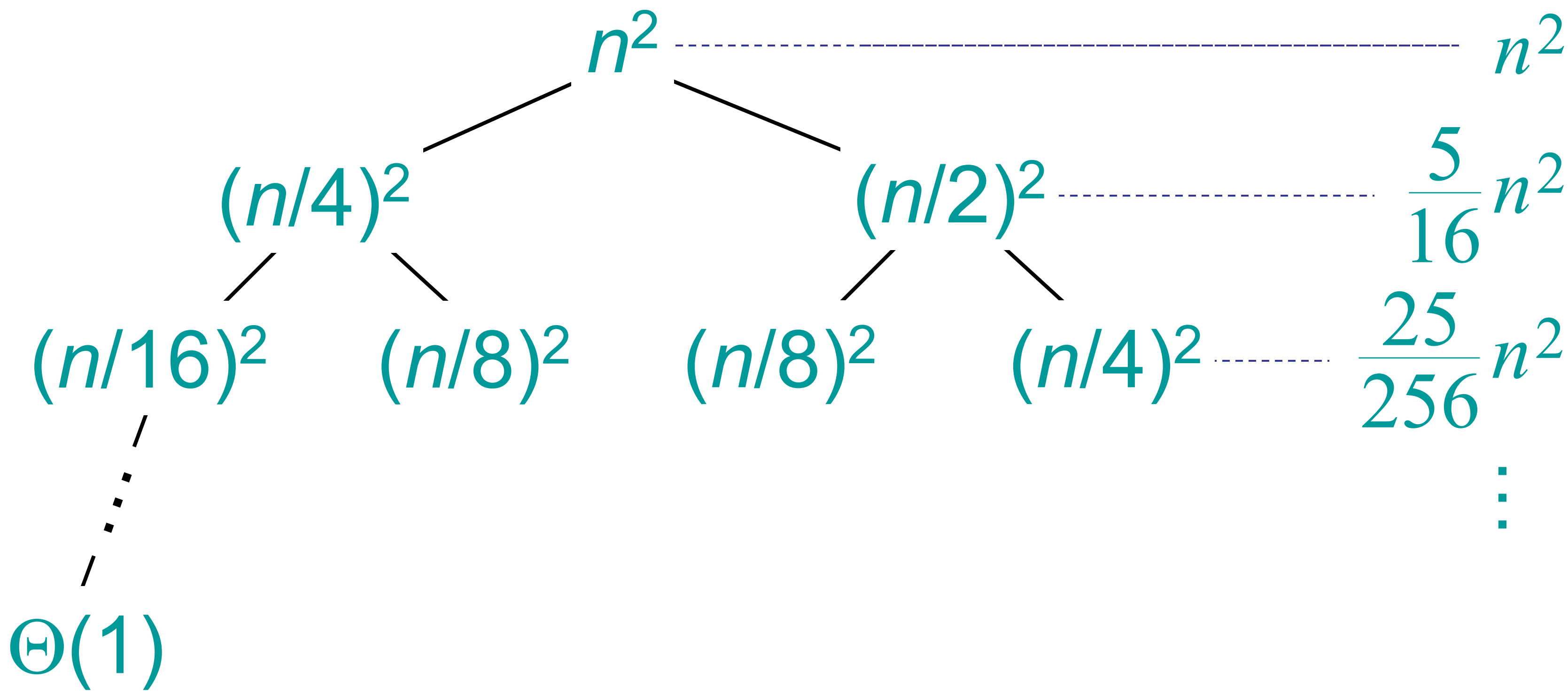
Example of recursion tree

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



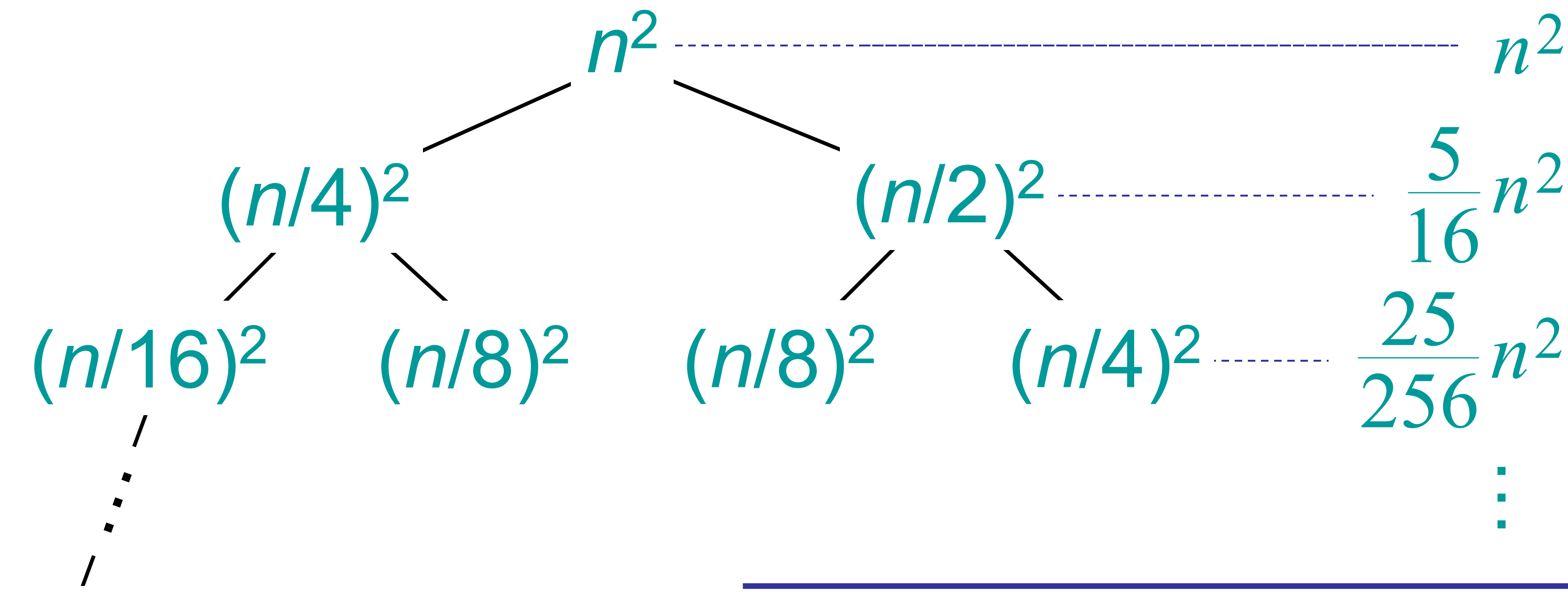
Example of recursion tree

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



Example of recursion tree

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

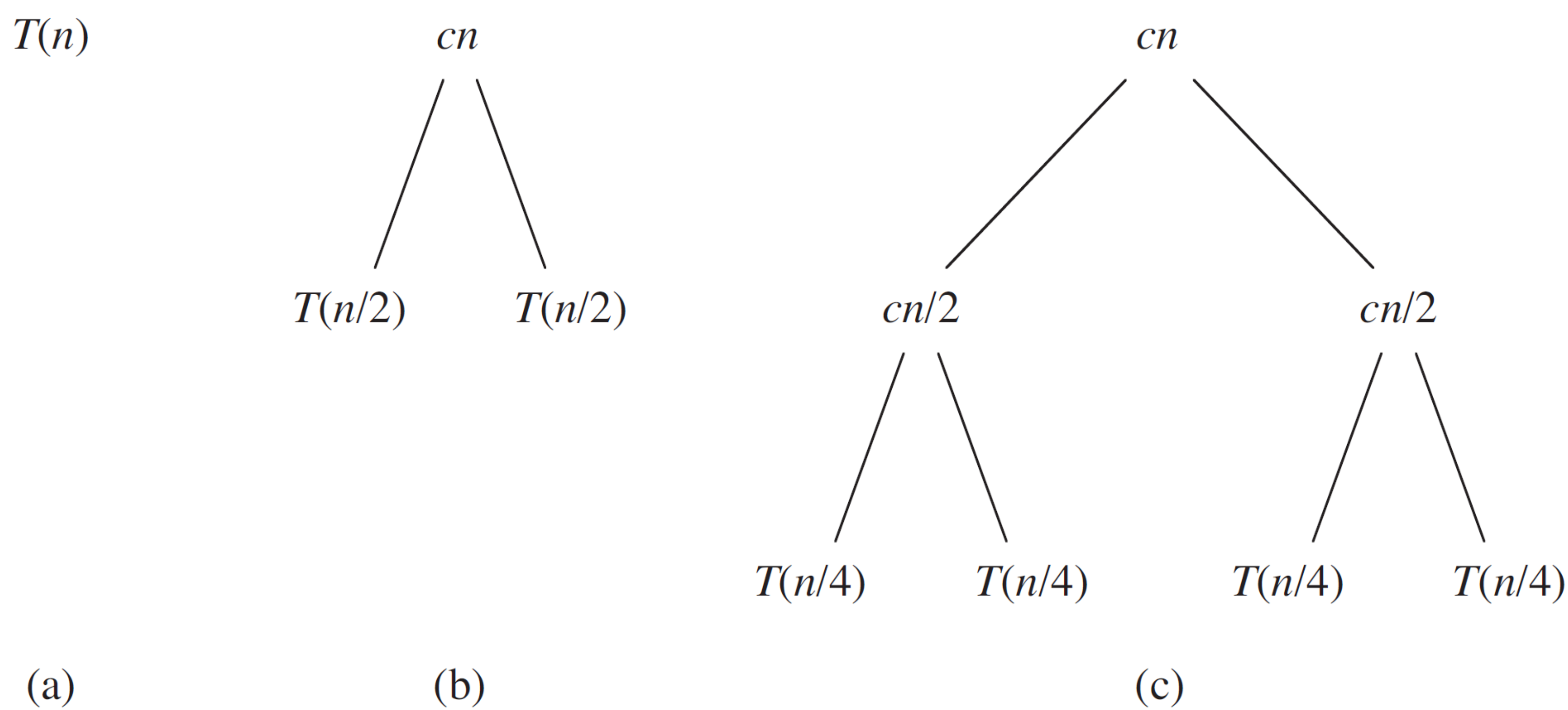


Total $= n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$

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

Geometric	
Sequence formula of n^th term	$a_n = ar^{(n-1)}$
Series formula for the sum of n terms	$S_n = \frac{a(1-r^n)}{(1-r)}$
Series formula for sum of infinite terms	$S_n = \frac{a}{(1-r)}$ when $\|r\| < 1$

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



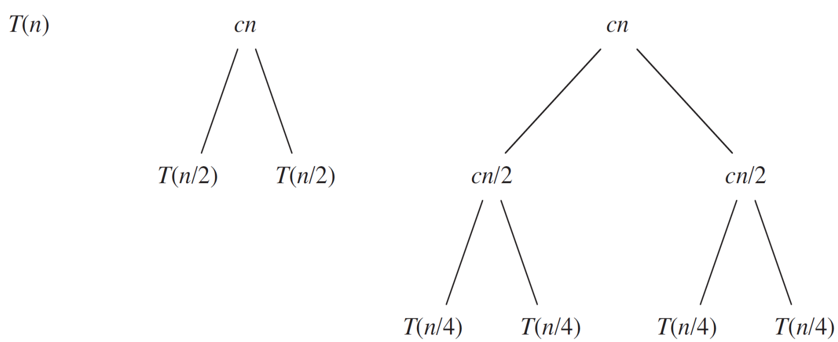
Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes in the last level
5. Cost of the last level

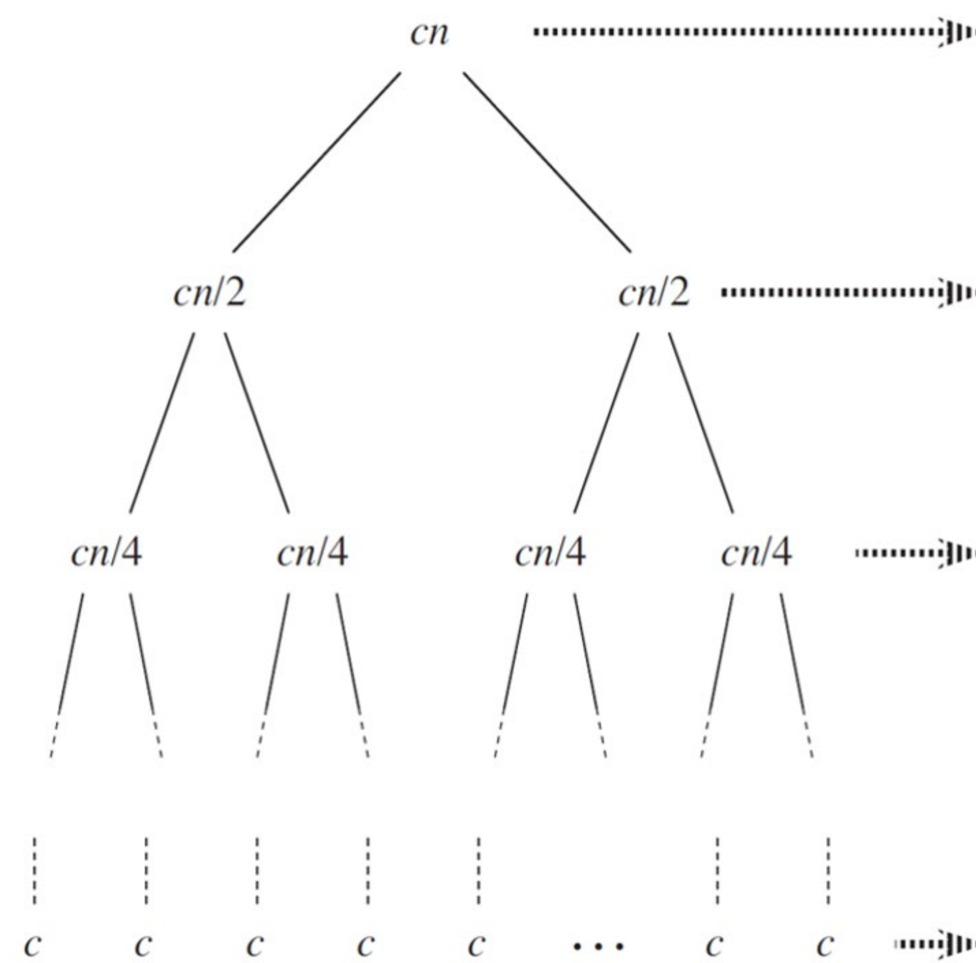
$$T(n) = 2T(n/2) + cn.$$

Steps:

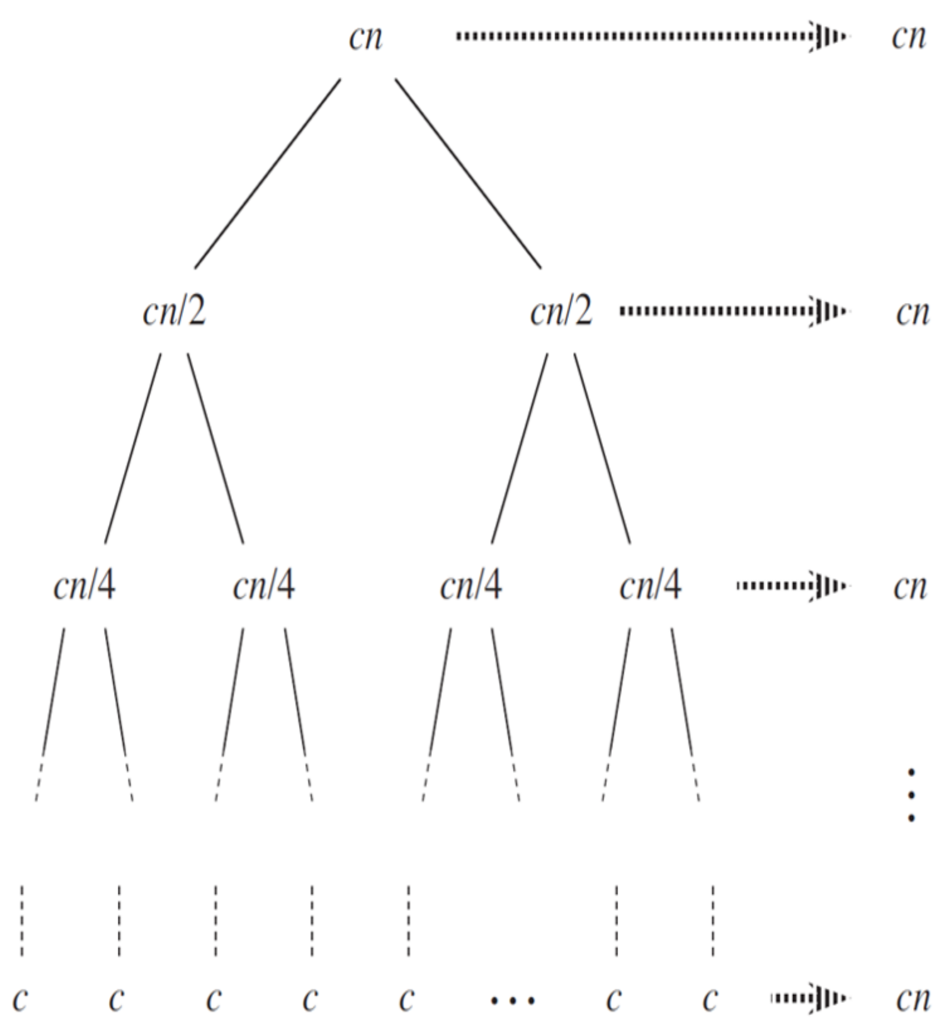
1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes in the last level
5. Cost of the last level



(a) (b) (c)



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



Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes in the last level
5. Cost of the last level

\# cost of level 0 $= cn$

\# cost of level 1 $= \frac{cn}{2} + \frac{cn}{2}$

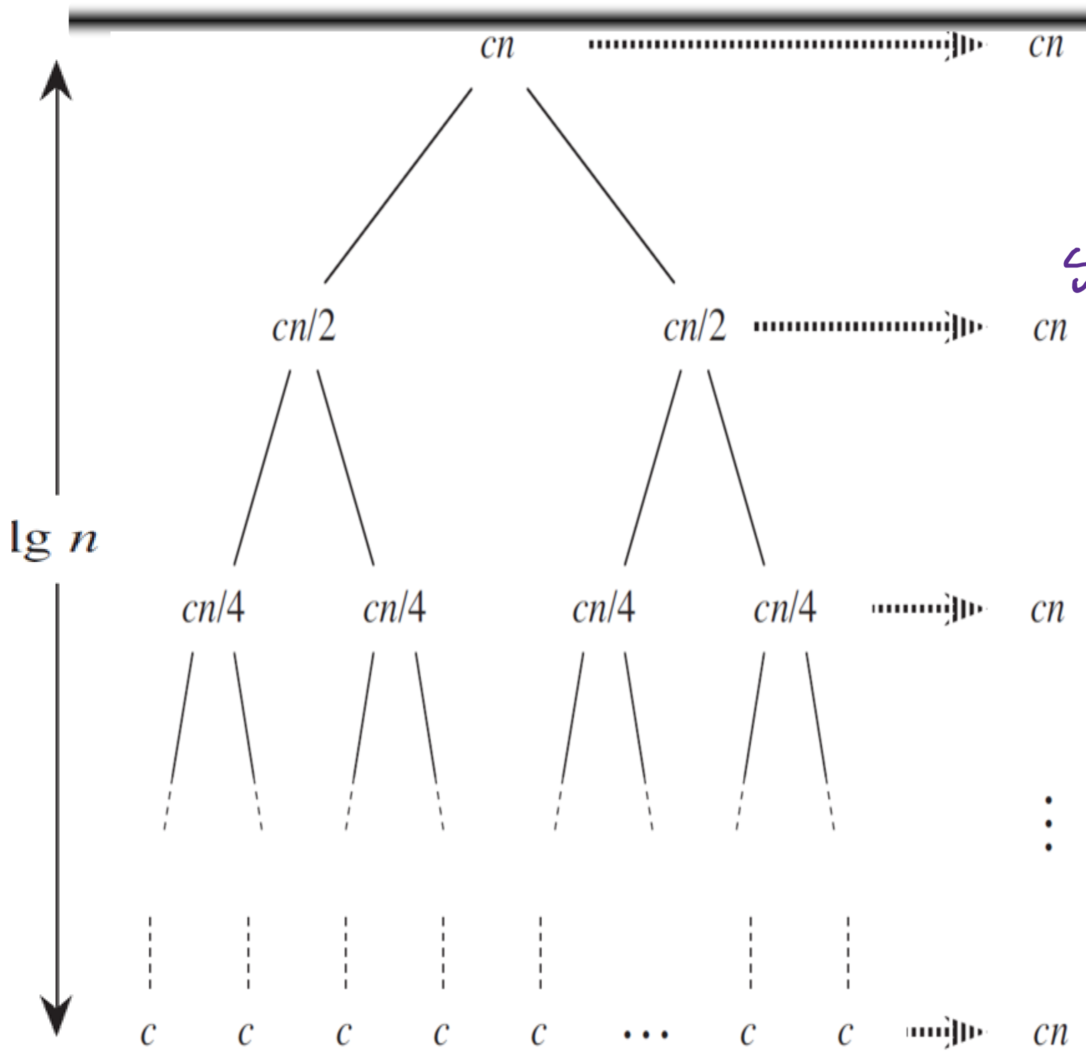
$= cn$

\# cost of level 2 $=$

$\frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4}$

$= cn$

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



Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes: last level
5. Cost of the last level

size of a subproblem:

at level 0 = $cn/2^0$

at level 1 = $cn/2^1$

at level 2 = $cn/2^2$

⋮

at level i = $cn/2^i$

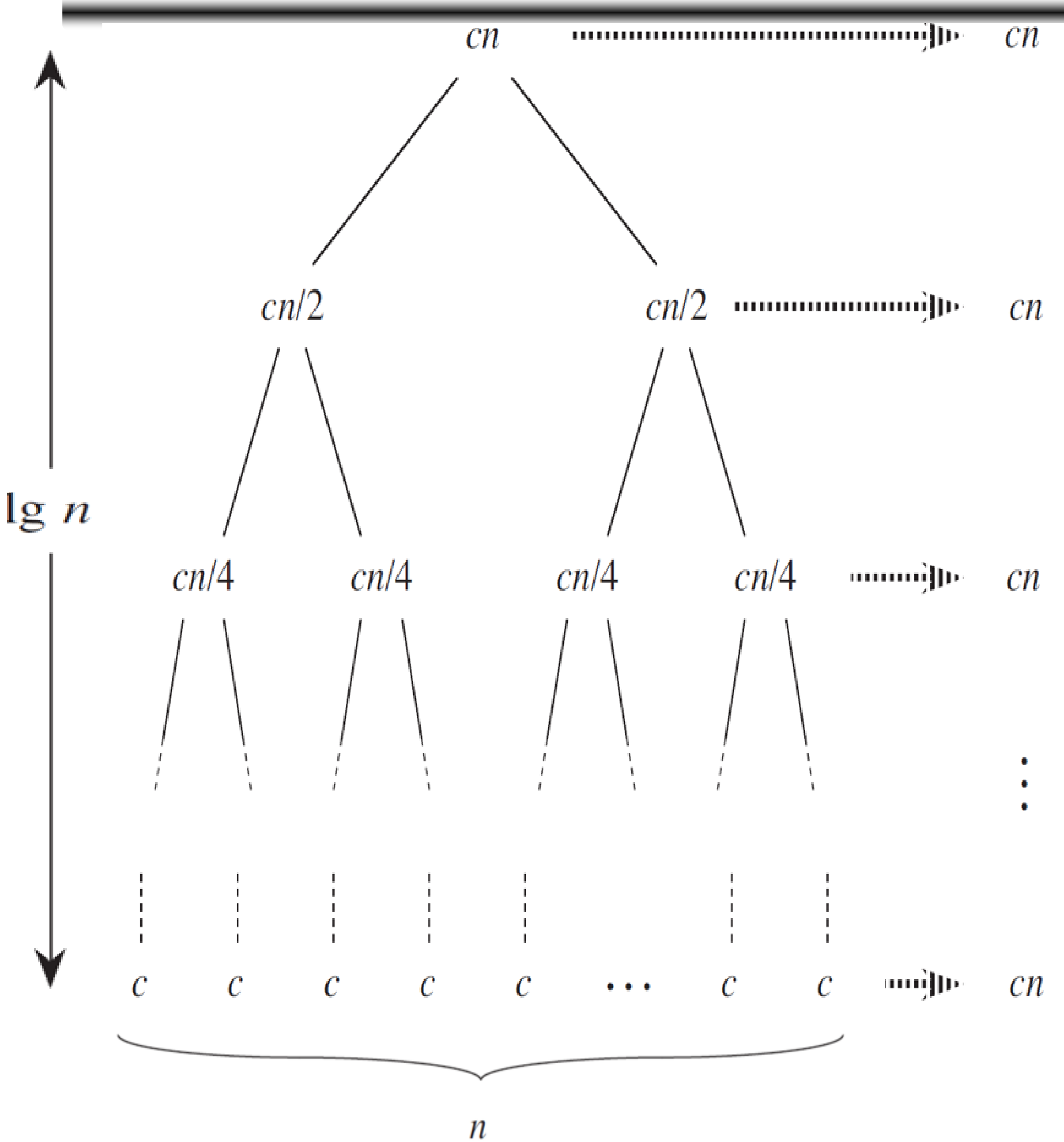
now, $\frac{cn}{2^i} = c$

$\Rightarrow n = 2^i$

$\Rightarrow \log_2 n = \log_2 2^i$

$\Rightarrow i = \log_2 n$

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



Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes: last level
5. Cost of the last level

Level 0 has 2^0 nodes (1)

" 1 " 2^1 " (2)

" 2 " 2^2 " (4)

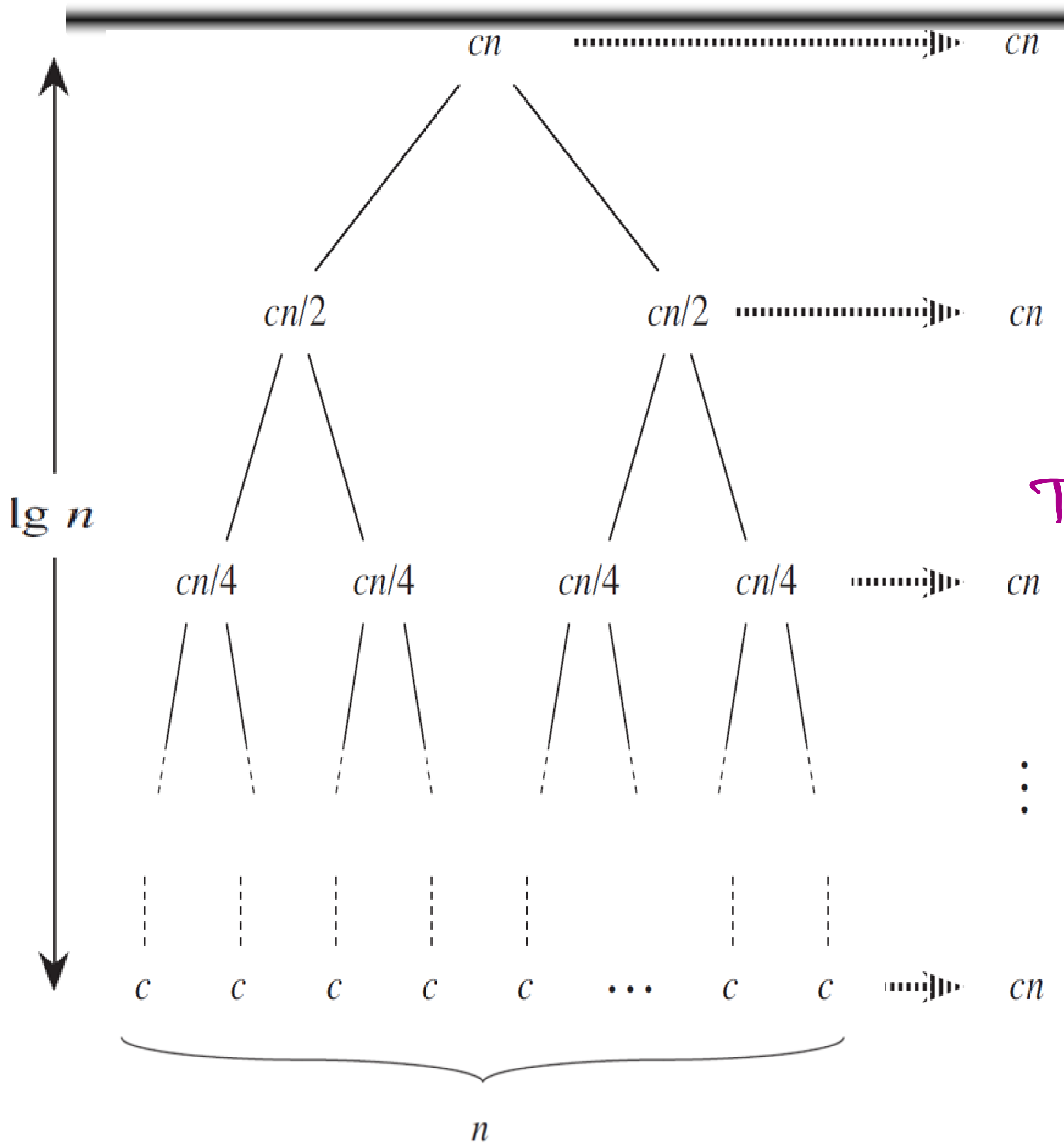
⋮

Level $\log_2 n$ " $2^{\log_2 n}$ nodes

Hence, $n^{\log_2 2}$

$= n$ nodes

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



Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes: last level
5. Cost of the last level

$cn \approx O(n)$

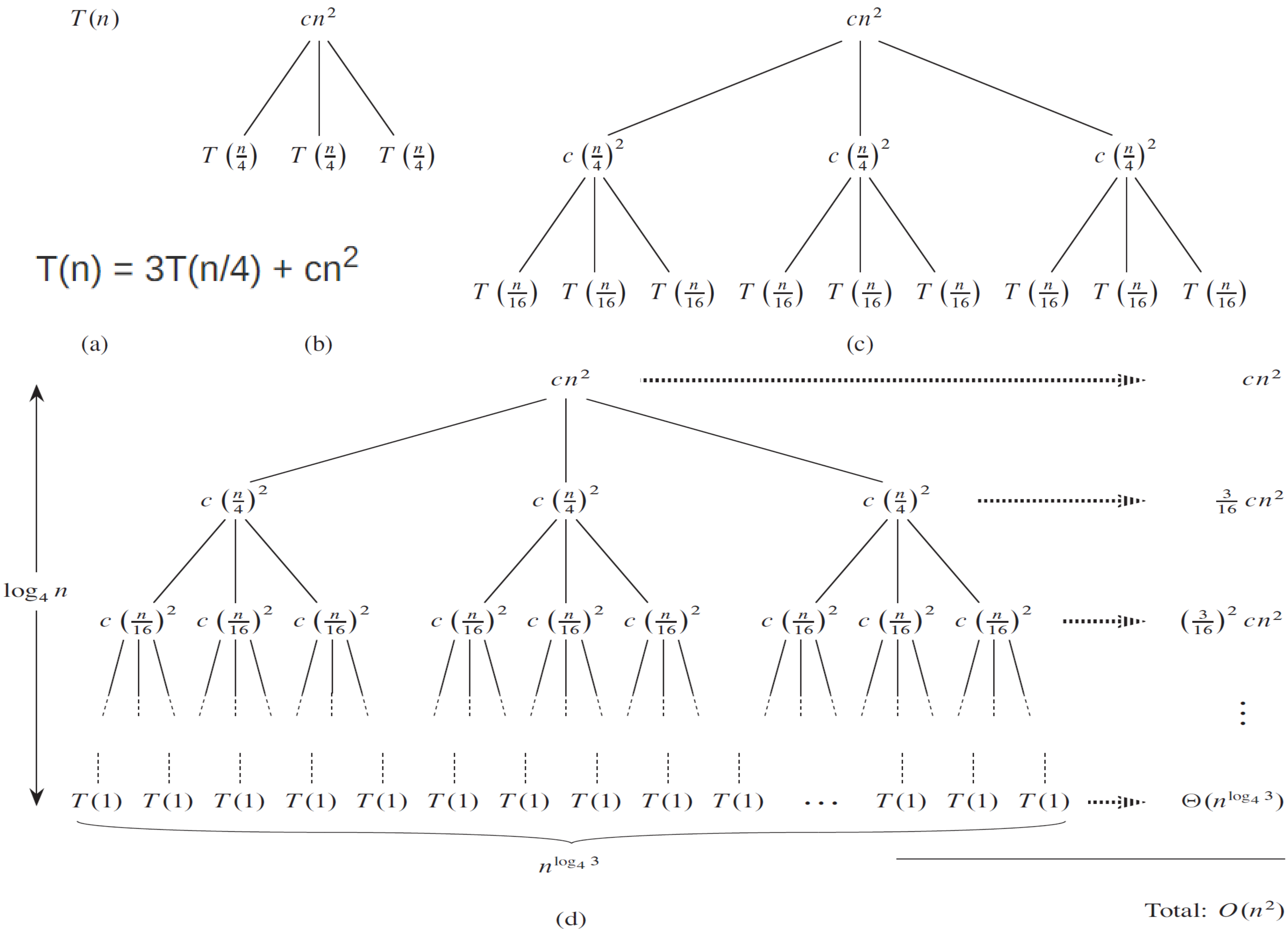
Total Costs:

$$T(n) = \underbrace{(cn + cn + \cdots)}_{\text{for } \log_2 n \text{ levels}} + O(n)$$

$$= cn\log_2 n + O(n)$$

$$= O(n\log_2 n) + O(n)$$

$$= O(n\log_2 n)$$



$T(n)$
cn^2
$T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$
$T(n) = 3T(n/4) + cn^2$
$c\left(\frac{n}{4}\right)^2$
$T\left(\frac{n}{16}\right)$
(a)
(b)
(c)
$c\left(\frac{n}{16}\right)^2$
$\log_4 n$
$T(1)$
$\frac{3}{16}cn^2$
$\left(\frac{3}{16}\right)^2 cn^2$
$\Theta(n^{\log_4 3})$
$n^{\log_4 3}$
Total: $O(n^2)$
(d)

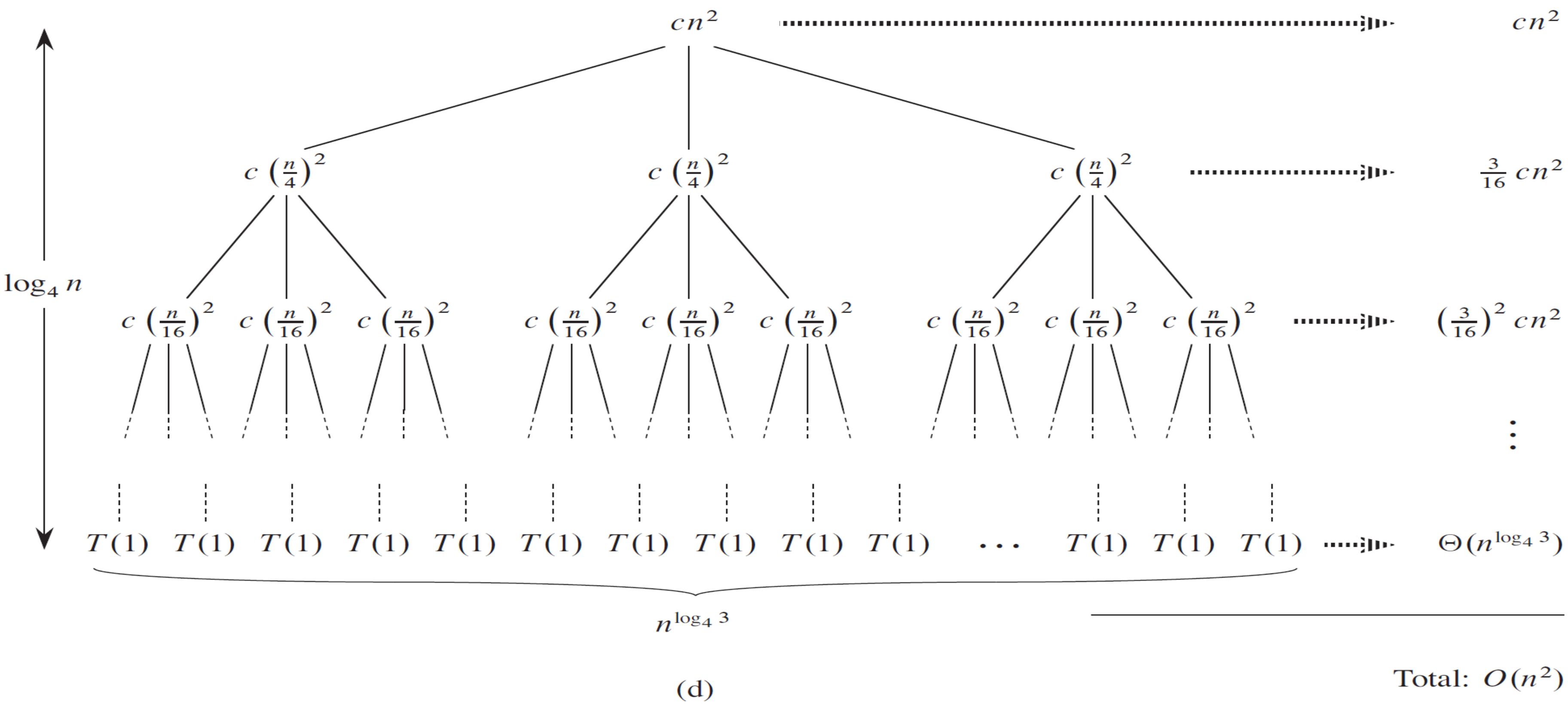
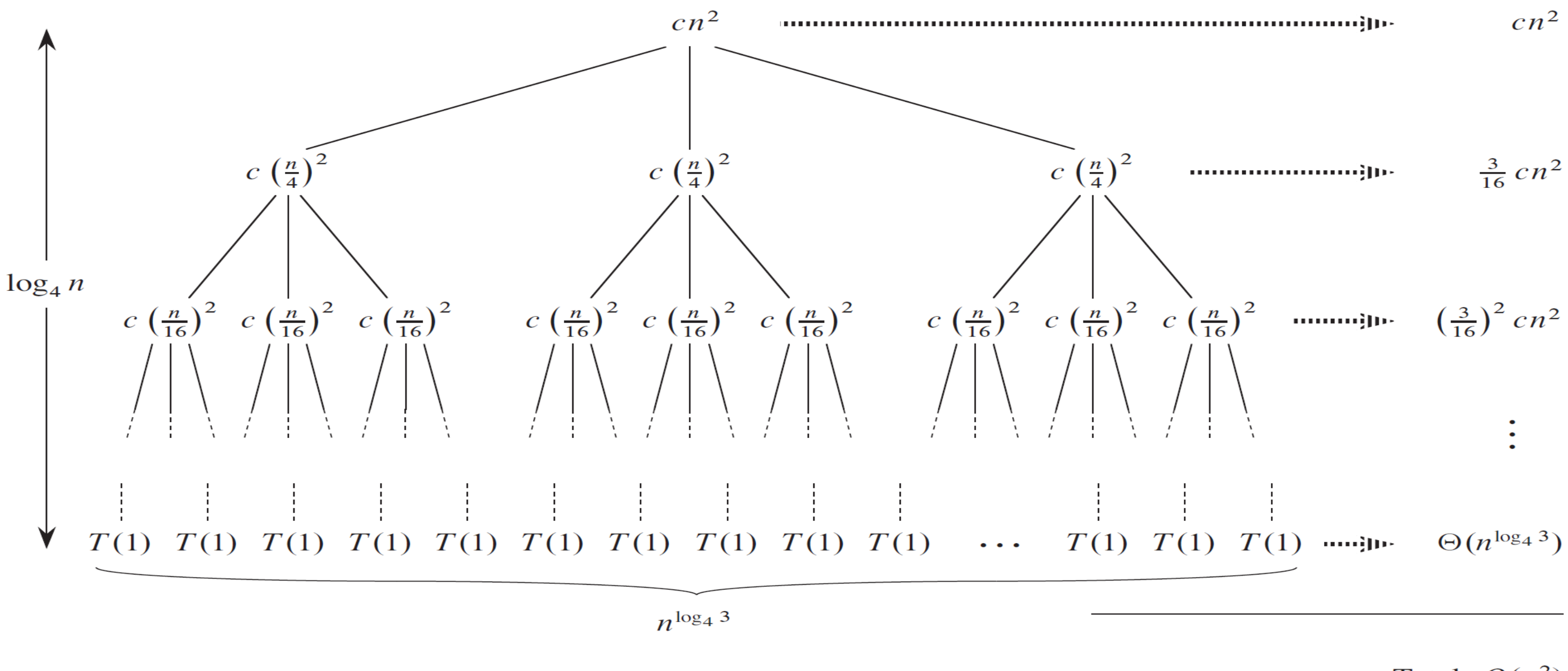


Figure 4.5 Constructing a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which progressively expands in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes in the last level
5. Cost of the last level



(d)

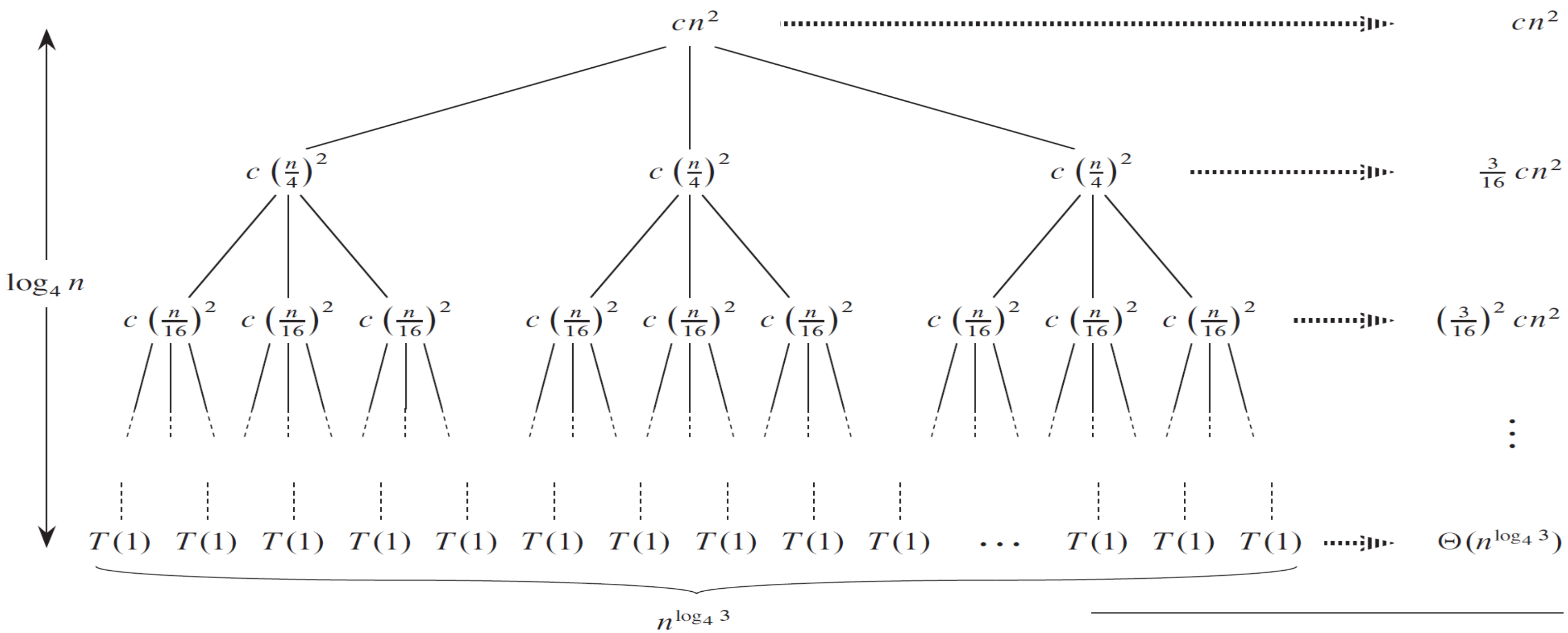
Steps:

1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes in the last level
5. Cost of the last level

- Size of sub-problem at level-0 = $n/4^0$
- Size of sub-problem at level-1 = $n/4^1$
- Size of sub-problem at level-2 = $n/4^2$

Size of sub-problem at level-i = $n/4^i$

$\log_4 n$ (it has $\log_4 n + 1$ levels).



Total: $O(n^2)$

(d)

Steps:
1. Construct the tree
2. Cost of each level
3. Total number of levels
4. Number of nodes in the last level
5. Cost of the last level

- Level-0 has 3^0 nodes i.e. 1 node
- Level-1 has 3^1 nodes i.e. 3 nodes
- Level-2 has 3^2 nodes i.e. 9 nodes

Level-$\log_4 n$ has $3^{\log_4 n}$ nodes i.e. $n^{\log_4 3}$ nodes

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

Issues with Recurrence Tree?

The recurrence tree method may not be able to accurately model the running time of the algorithm:

- If the recursive step of the recursion makes a variable number of recursive calls, e.g. overlapping sub-problem
- The process of summing the costs across the tree's levels can sometimes be ambiguous
  - especially if the amount of work at each level doesn't form a clear geometric or arithmetic pattern.
  - If base case of the recursion does not take a constant amount of time,
- Some recurrences generate trees with infinite depths. While the recurrence continues indefinitely, determining a closed-form solution from such a tree can be challenging.
- If the time it takes to make each recursive call is not a function of the size of the input

The substitution method

1. Guess a solution
2. Use induction to prove that the solution works

Substitution method

- Guess a solution
  - $T(n) = O(g(n))$
  - Induction goal: apply the definition of the asymptotic notation
    - $T(n) \le d\, g(n)$, for some $d > 0$ and $n \ge n_0$
  - Induction hypothesis: $T(k) \le d\, g(k)$ for all $k < n$

Prove that if the guess is true for $T(k) \le d\, g(k)$, for all $k < n$, then this implies that $T(n) \le d\, g(n)$, for some $d > 0$ and $n \ge n0$

- We substitute the **guessed solution** for the function when applying the inductive hypothesis to smaller values; hence the name "substitution method."
- This method is powerful, but we must be able to guess the form of the answer in order to apply it.
- It's a powerful way to establish lower or upper bounds on a recurrence

Example: Binary Search

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

Induction hypothesis: $T(k) \le d\, g(k)$ for all $k < n$

- Guess: $T(n) = O(\lg n)$
  - Induction goal: $T(n) \le d \lg n$, for some d and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le d \lg(n/2)$ We will use $k = \frac{n}{2}$
- Proof of induction goal:

$$T(n) = T(n/2) + c \le d \lg(n/2) + c$$

$$= d \lg n - d + c \le d \lg n$$

if: $-d + c \le 0$, $d \ge c$

Induction hypothesis: $T(k) \le d\, g(k)$ for all $k < n$

$T(n) = T(n-1) + n$

- Guess: $T(n) = O(n^2)$
  - Induction goal: $T(n) \le d\, n^2$, for some d and $n \ge n_0$
  - Induction hypothesis: $T(k-1) \le d(k-1)^2$ for all $k < n$ let's $k = n - 1$
- Proof of induction goal:

$$T(n) = T(n-1) + n \le d\,(n-1)^2 + n$$

$$= dn^2 - (2dn - d - n) \le dn^2$$

$$\text{if: } 2dn - d - n \ge 0 \Leftrightarrow d \ge n/(2n-1) \Leftrightarrow d \ge 1/(2 - 1/n)$$

  - For $n \ge 1 \Rightarrow 2 - 1/n \ge 1 \Rightarrow$ any $d \ge 1$ will work

Induction hypothesis: $T(k) \leq d\, g(k)$ for all $k < n$

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

- Guess: $T(n) = O(n\lg n)$
  - Induction goal: $T(n) \leq dn \lg n$, for some d and $n \geq n_0$
  - Induction hypothesis: $T(n/2) \leq dn/2 \lg(n/2)$ We will use $k = \frac{n}{2}$
- Proof of induction goal:

$$T(n) = 2T(n/2) + n \leq 2d\,(n/2)\lg(n/2) + n$$

$$= dn \lg n - dn + n \leq dn \lg n$$

$$\text{if: } -dn + n \leq 0 \Rightarrow d \geq 1$$

Induction hypothesis: $T(k) \le d\, g(k)$ for all $k < n$

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

- Guess: $T(n) = O(n)$
  - Induction goal: $T(n) \le dn$, for some d and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le d * n/2$

We will use $k = \frac{n}{2}$

- Proof of induction goal:

$$T(n) = 2T(n/2) + n \le 2d\,(n/2) + n$$

$$= dn + n = (d + 1)\,n \not\le d.n$$

The above inequality does not hold because d+1 cannot be less than d. Hence, $T(n) \neq O(n)$

Induction hypothesis: $T(k) \leq d\, g(k)$ for all $k < n$

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

We want to show that $T(n) \leq dn^2$ for some constant $d > 0$. By the induction hypothesis, we have that $T(\lfloor n/4 \rfloor) \leq d\lfloor n/4 \rfloor^2$. So using the same constant $c > 0$ as before, we have

$$\begin{aligned}
T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\
&\leq 3d(n/4)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\leq dn^2 \quad (\text{when } c \leq (13/16)d, \text{ i.e. } d \geq (16/13)c)
\end{aligned}$$

Making a Good Guess

There is no general way to guess the correct solution to recurrences. Guessing a solution takes experience and, occasionally, creativity.

There are some heuristics that can help us make a good guess. (e.g. Recursion Tree)

If a recurrence is similar to the one we have seen before, then guessing a similar solution is reasonable. For example: $T(n) = 2\,T(\lfloor {}^{n}/_{2} \rfloor + 17) + n$, we make the guess that $T(n) = O(n \lg n)$

Another way to make a good guess is to prove the loose upper and lower bounds on the recurrence and then reduce the range of uncertainty. For example:

- Start with and prove the initial lower bound of $T(n) = \Omega(n)$ for the recurrence.
- Start with and prove the initial upper bound of $T(n) = O(n^2)$ for the recurrence.
- Then gradually lower the upper bound and raise the lower bound until convergence to the correct, asymptotically tight solution of $T(n) = \Theta(n \lg n)$.

Sometimes the correct guess at an asymptotic bound on the solution of a recurrence doesn't work. This can be solved by revising the guess and subtracting a lower-order term in the guess.

Changing variables

Solve the recurrence $T(n) = 2T(\sqrt{n}) + \lg n$.

Solution: Change of variables. Let $m = \lg n$ and $S(m) = T(2^m)$.

Note that $n = 2^m$ and $\sqrt{n} = 2^{m/2}$.

Then we get:

$$\begin{aligned} T(n) &= T(2^m) \\ &= 2T(2^{m/2}) + \lg(2^m) \\ &= 2S(m/2) + m \\ &= O(m \log m) \\ &= O(\log n \log \log n) \end{aligned}$$