Recursion-tree method

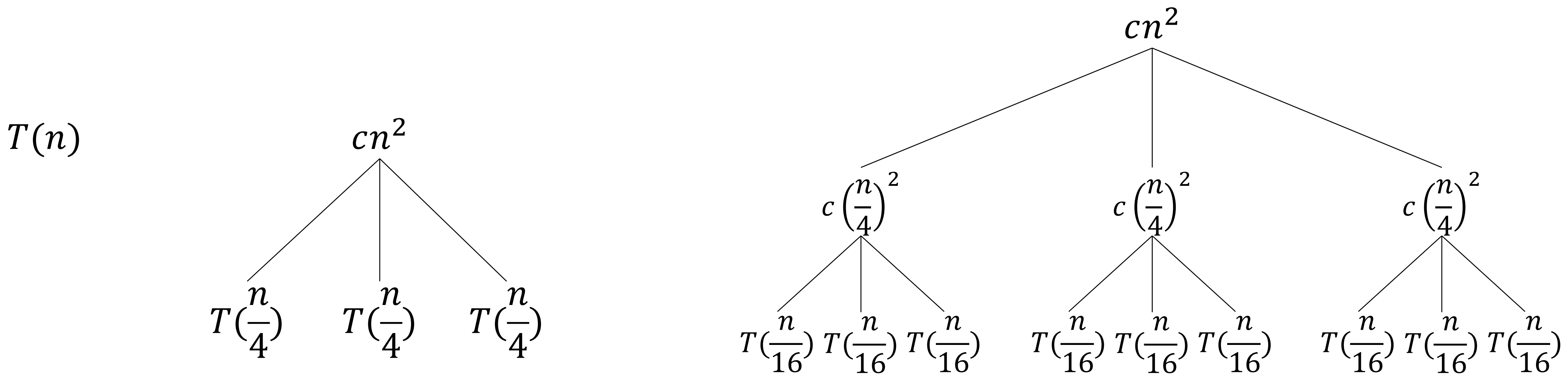
- Substitution method is useful, but it could be difficult to come up with a good ***guess***.

- Draw a ***recursion tree*** to devise a good guess.

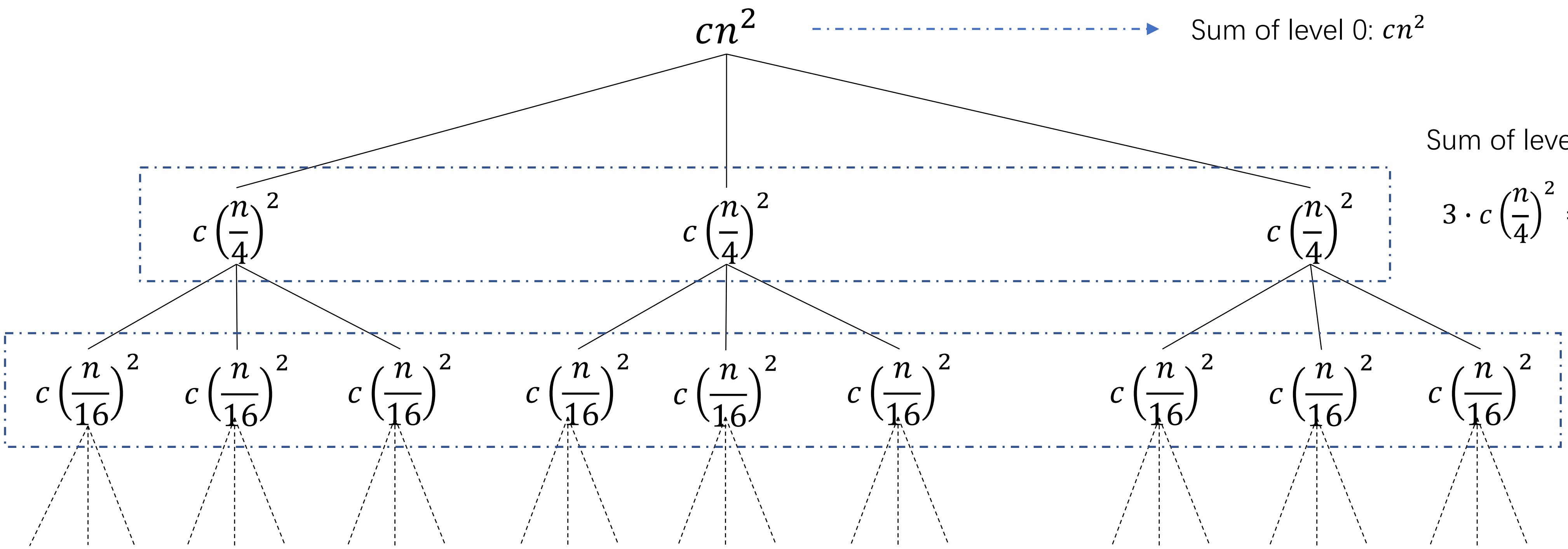
- In a recursion tree, each ***node*** represents the cost of a single subproblem.
- We sum the costs within each level to get a set of per-level costs.
- We sum all the per-level costs to determine the total cost.

An example of recursion-tree method

- Given $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$, what's the upper bound of $T(n)$?
- Tolerate some sloppiness: draw a recursion tree for $\boldsymbol{T(n) = 3T(n/4) + cn^2}$



Continue expanding



Sum of level 0: cn^2

Sum of level 1:

$$3 \cdot c\left(\frac{n}{4}\right)^2 = \frac{3}{16}cn^2$$

Sum of level 2:

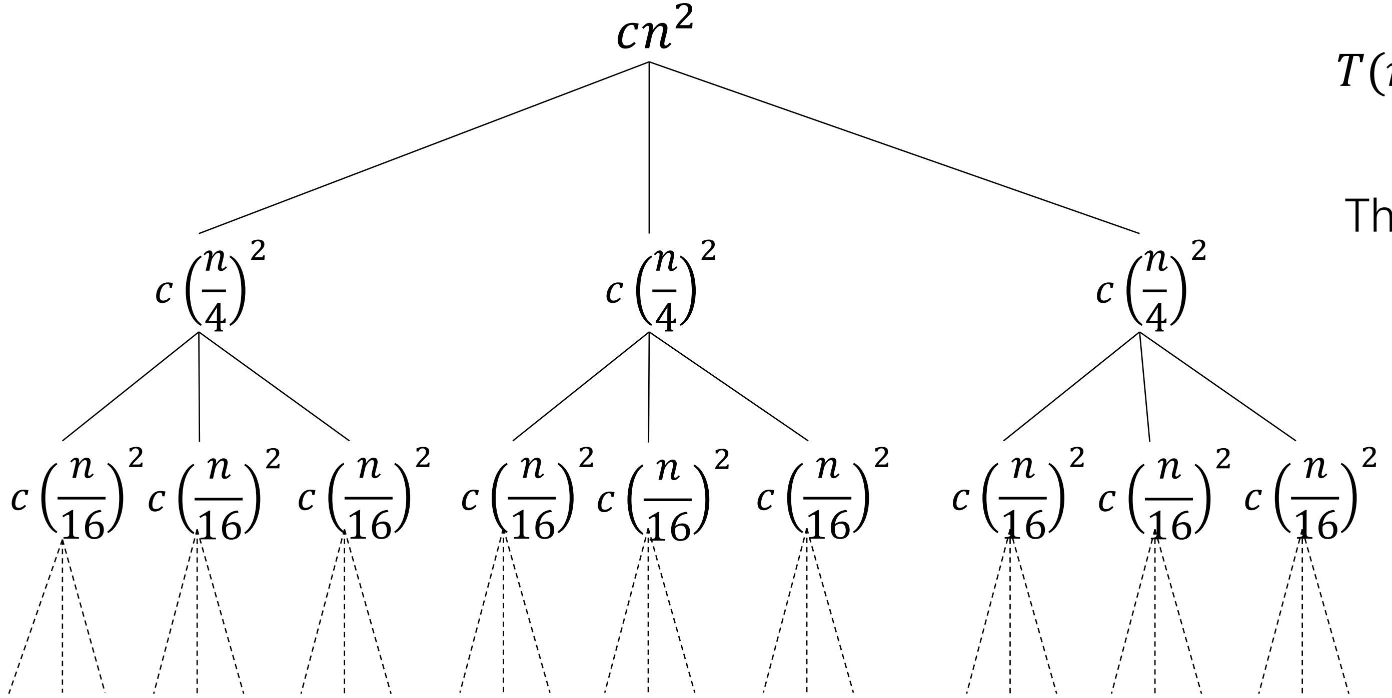
$$3^2 \cdot c\left(\frac{n}{16}\right)^2 = \left(\frac{3}{16}\right)^2 cn^2$$

⋮

Sum of level i?

Questions about the recursion tree

- When does the expanding end? I.e., What's the number of levels when it reaches the subproblem of the smallest size?



Subproblem sizes decrease by a factor of 4:

$$T(n) \longrightarrow T\left(\frac{n}{4}\right) \longrightarrow T\left(\frac{n}{4^2}\right) \longrightarrow T\left(\frac{n}{4^3}\right) \quad \cdots$$

The subproblem size for a node at depth i is $\frac{n}{4^i}$

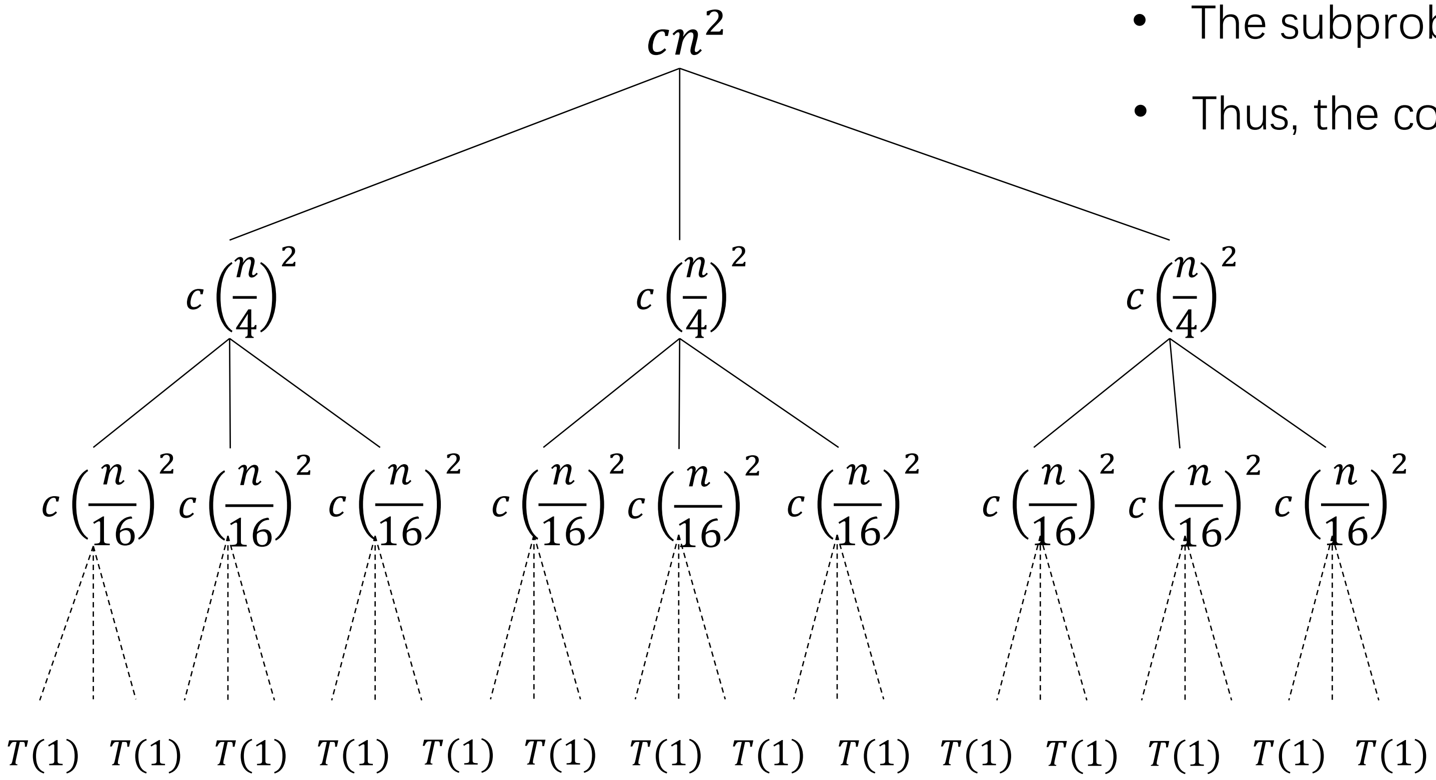
The subproblem size hits 1 when:

$$\frac{n}{4^i} = 1 \qquad \text{i.e.,} \quad i = \log_4 n$$

Thus, the tree has $\boldsymbol{\log_4 n + 1}$ levels

(at depth $0,1,2,\cdots,\log_4 n$)

What's the per-level cost?



- Each level has three times more nodes than the level above
- Thus, the number of nodes at depth i is 3^i

- The subproblem sizes reduce by a factor of 4 for each level
- Thus, the cost for each node at depth i is $c\left(\frac{n}{4^i}\right)^2$

The total cost for the level at depth i is:

$$3^i \cdot c\left(\frac{n}{4^i}\right)^2 = \left(\frac{3}{16}\right)^i cn^2$$

Bottom level cost

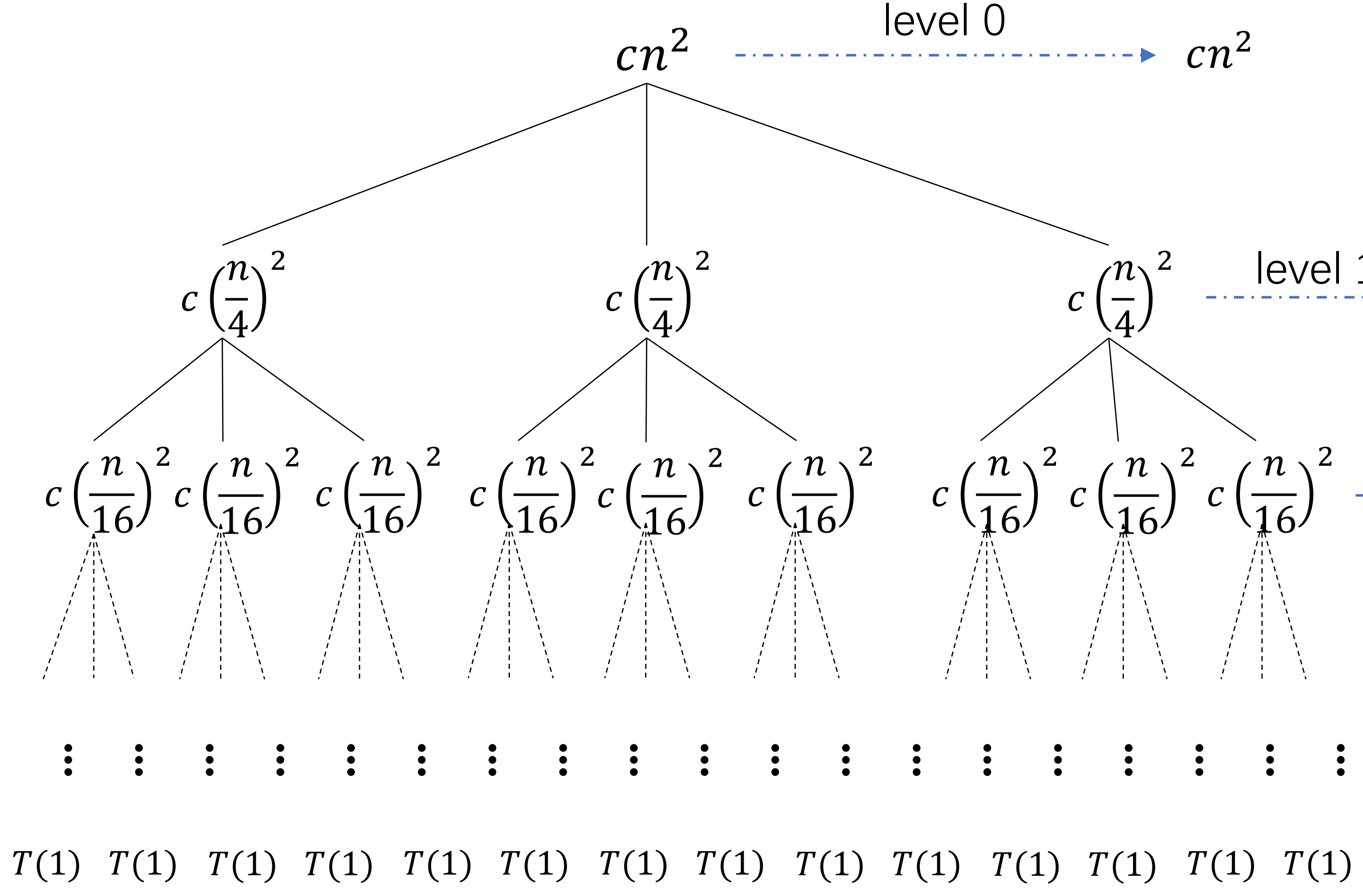
- Bottom level is at the depth $i = \log_4 n$
- The the number of leaf nodes: $3^i = 3^{\log_4 n} = n^{\log_4 3}$
- Each leaf node costs $T(1)$

- Thus, the total cost for bottom level is: $T(1)n^{\log_4 3} = \Theta(n^{\log_4 3})$

Total cost

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

A geometric series



level 0 $\rightarrow$ cn^2

level 1 $\rightarrow$ $\frac{3}{16}cn^2$

level 2 $\rightarrow$ $\left(\frac{3}{16}\right)^2 cn^2$

level i $\rightarrow$ $\left(\frac{3}{16}\right)^i cn^2$

level $\log_4 n - 1$ $\rightarrow$ $\left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2$

$\rightarrow$ $\Theta(n^{\log_4 3})$ Bottom level

Total cost

- Summation of a geometric series

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

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

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

Find an upper bound for $T(n)$

- Use the property of geometric series

Now, we have derived a guess $T(n) = O(n^2)$ for the original recurrence:

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

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

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

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

$\log_4 3 \approx 0.792$

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

Some observation

- Recall the recurrence: $T(n) = 3T\boxed{(n/4}) + \boldsymbol{cn^2}$
- The solution is: $T(n) < \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\boxed{\log_4 3}}) = \boldsymbol{\frac{16}{13}cn^2} + \Theta(n^{\log_4 3}) = O(n^2)$
- The cost of the **root** node *dominates* the total cost.

- Think: What if the root node costs less?
  - For example: n, or constant c?
- How does the solution change?

The master method (Master theorem)

- It provides a "cookbook" for solving recurrences of the form

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

- where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.
  - It divides a problem of size n into a subproblems, each of size n/b.
  - $f(n)$: the cost of dividing and combining.

- 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 $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Intuitive understanding of master method

- In each of the three cases, we compare $f(n)$ with the function $n^{\log_b a}$.

- If case 1, $n^{\log_b a}$ is the larger, then $T(n) = \Theta(n^{\log_b a})$.
- If case 3, $f(n)$ is the larger, then $T(n) = \Theta(f(n))$.
- If case 2, the two functions are the same size, we just multiply by a logarithmic factor, and the solution is $T(n) = \Theta(f(n)\lg n) = \Theta\left(n^{\log_b a} \lg n\right)$.

Some technicalities

- In case 1, not only must $f(n)$ be smaller than $n^{\log_b a}$, it must be ***polynomially*** smaller.
  - E.g., $f(n)$ must be asymptotically smaller than $n^{\log_b a}$ by a factor of n^{ϵ}.

- In case 3, not only must $f(n)$ be greater than $n^{\log_b a}$, it must also be ***polynomially*** larger,
- And in addition satisfy the "regularity" condition that $af\left(\frac{n}{b}\right) \leq cf(n)$ (which is satisfied by most of the polynomially bounded fucntions).

- The three cases do not cover all the possibilities for $f(n)$.
  - There is a gap between case 1 and 2 when $f(n)$ is smaller than $n^{\log_b a}$, but not polynomially smaller.
  - There is a gap between case 2 and 3 when $f(n)$ is larger than $n^{\log_b a}$, but not polynomially larger.
- We cannot use master method when $f(n)$ falls into the gaps.

Example with master method

- Maximum subarray problem (and merge sort)

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

- We have $a = 2$, $b = 2$, $f(n) = \Theta(n)$.
- Thus, we have $n^{\log_b a} = n = \Theta(n)$
- And thus **case 2** applies.
- Therefore the solution is $T(n) = \Theta(n \lg n)$.

Other examples

- Consider the recurrence $T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$
- $a = 2$, $b = 2$ → $n^{\log_b a} = n$, which means $f(n) = n \lg n$ is asymptotically larger than $n^{\log_b a}$.
- Does case 3 apply?
- It looks like so, but the problem is that $f(n)$ is **not** *polynomially* larger.

- Can we find a $\epsilon > 0$, such that $f(n) \geq cn^{\log_b a+\epsilon}$
- I.e., $n \lg n \geq cn^{1+\epsilon}$ → $\lg n \geq cn^{\epsilon}$
- However, we cannot find such a c, because $\lg n$ grows slower than any polynomial function n^{ϵ} when $\epsilon > 0$.

Other examples (cont.)

- Consider $T(n) = 9T\left(\frac{n}{3}\right) + n$
- $a = 9$, $b = 3$ → $n^{\log_b a} = n^2$
- $f(n) = n = O\left(n^{\log_b a - \epsilon}\right) = O(n^{2-\epsilon})$, when $\epsilon = 1$
- Thus, case 1 can be applied, and $T(n) = \Theta(n^2)$.

Other examples (cont.)

- Consider $T(n) = T\left(\frac{2n}{3}\right) + 1$
- $a = 1$, $b = 3/2$, → $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1 = f(n)$
- Case 2 applies, and the solution is $T(n) = \Theta(\lg n)$.

Other examples (cont.)

- Consider $T(n) = 3T\left(\frac{n}{4}\right) + n\lg n$
- $a = 3$, $b = 4$ → $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
- Since $f(n) = n\lg n = \Omega(n^{\log_4 3+\epsilon})$ is satisfied as long as $\epsilon < 0.207$, because $\lg n$ is smaller than any polynomial function
- Then case 3 applies if we can also show that the regularity condition holds
- $af\left(\frac{n}{b}\right) = 3\left(\frac{n}{4}\right)\lg(\frac{n}{4}) = \frac{3}{4}n\lg n - \frac{3n}{2} \le cf(n) = cn\lg n$
- The above holds if $c = \frac{3}{4}$, for sufficiently large n.
- Therefore, the solution is $T(n) = \Theta(n\lg n)$.

Proof of the master theorem

- The proof has two steps: lemma 1 and lemma 2.
- Lemma 1 reduces the problem of solving the recurrence to the problem of evaluating an expression that contains a <u>summation</u>.
- Lemma 2 determines <u>bounds</u> on this summation.

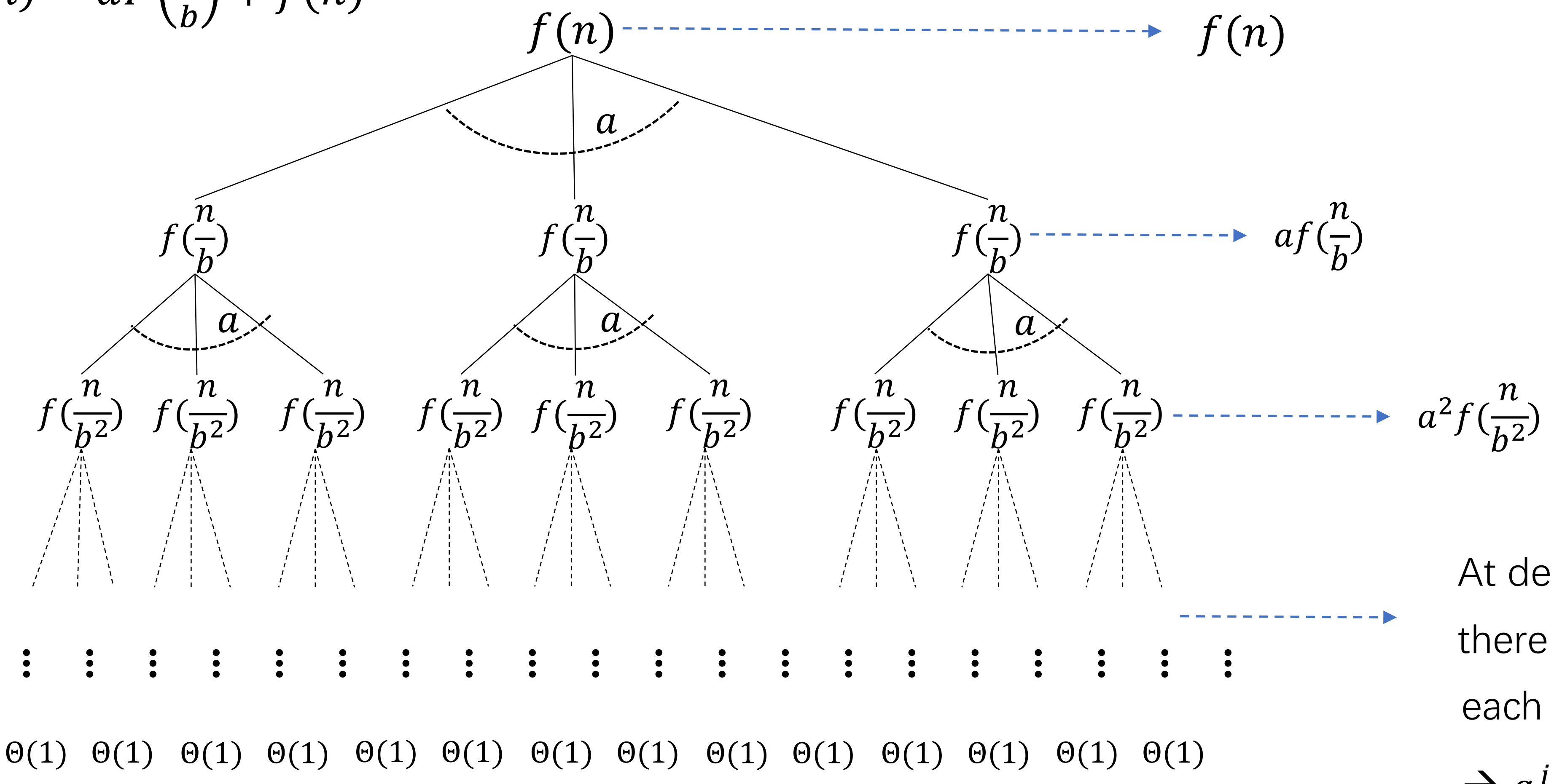
- **Lemma 1**
- Let $a \geq 1$ and $b > 0$ be constants, and let $f(n)$ be a nonnegative function defined on exact power of b (simplification). Define $T(n)$ as:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + f(n), & \text{if } n = b^i \end{cases}$$

- Then $T(n) = \Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$

Draw the recursion tree

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



At depth j

there are a^j nodes

each has cost $f(\frac{n}{b^j})$

→ $a^j f(\frac{n}{b^j})$

Summing the costs of all nodes

- Leaf nodes
  - When $\frac{n}{b^j} = 1$, → $j = \log_b n$, → Leaf nodes are at depth $\log_b n$
  - There are $a^{\log_b n} = n^{\log_b a}$ leaf nodes
  - Sum of all leaf nodes: $n^{\log_b a} \cdot \Theta(1) = \Theta(n^{\log_b a})$

- Internal nodes
  - Depth j ranges from 0 to $\log_b n - 1$
  - Sum of all internal nodes:

$$\sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})$$

$$\boxed{T(n) = \Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})}$$

- Lemma 1 is proved

What does Lemma 1 tell us?

$$T(n) = \underbrace{\Theta\left(n^{\log_b a}\right)}_{} + \underbrace{\sum_{j=0}^{\log_b n-1} a^j f\left(\frac{n}{b^j}\right)}_{}$$

We want to know the asymptotic bounds on this part

The costs of solving $n^{\log_b a}$ subproblems of size 1

The costs of dividing and combining

Master theorem actually describes how the total cost is distributed:

- Case 1: dominated by the costs in leaf nodes
- Case 2: evenly distributed among all levels of the recursion tree
- Case 3: dominated by the cost of the root

Lemma 2 gives the bounds

- Let $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$, it has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $g(n) = O(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $g(n) = \Theta(f(n))$.

Proof of Lemma 2 (case 1)

- For case 1, $f(n) = O(n^{\log_b a-\epsilon})$ implies that $f\left(\frac{n}{b^j}\right) = O(\left(\frac{n}{b^j}\right)^{\log_b a-\epsilon})$. Substituting into $g(n)$:

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j}) = O\left(\boxed{\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a-\epsilon}}\right)$$

$$= n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b} \cdot \frac{b^\epsilon}{b^{\log_b a}}\right)^j = n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} (b^\epsilon)^j$$

$$= n^{\log_b a-\epsilon}\left(\frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}\right)$$

$$= n^{\log_b a-\epsilon}\left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right) = n^{\log_b a-\epsilon} O(n^\epsilon) = O(n^{\log_b a})$$

Proof of Lemma 2 (case 2)

- For case 2, $f(n) = \Theta(n^{\log_b a})$ implies that $f\left(\frac{n}{b^j}\right) = \Theta(\left(\frac{n}{b^j}\right)^{\log_b a})$. Substituting into $g(n)$:

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) = \Theta\left(\boxed{\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}}\right)$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} (1)^j$$

$$= n^{\log_b a} \cdot \log_b n$$

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

Proof of Lemma 2 (case 3)

- For case 3, $af\left(\frac{n}{b}\right) \leq cf(n) \rightarrow f\left(\frac{n}{b}\right) \leq \frac{c}{a}f(n)$
- Iterate many times: $f\left(\frac{n}{b^2}\right) \leq \frac{c}{a}f\left(\frac{n}{b}\right)$, $f\left(\frac{n}{b^3}\right) \leq \frac{c}{a}f\left(\frac{n}{b^2}\right)$, …, $f\left(\frac{n}{b^i}\right) \leq \frac{c}{a}f\left(\frac{n}{b^{i-1}}\right)$
- $\rightarrow f\left(\frac{n}{b^i}\right) \leq \left(\frac{c}{a}\right)^i f(n)$, or $a^i f\left(\frac{n}{b^i}\right) \leq c^i f(n)$. Substituting into $g(n)$:

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f\left(\frac{n}{b^j}\right) \leq \sum_{j=0}^{\log_b n-1} c^j f(n) \leq f(n)\sum_{j=0}^{\infty} c^j = f(n)\frac{1}{1-c} = O(f(n))$$

From the form of $g(n)$, we know that $g(n) = \Omega(f(n))$

Therefore, $g(n) = \Theta(f(n))$

Lemma 2 is proved!

Combining Lemma 1 and 2

- Lemma 1 tells: $T(n) = \Theta\left(n^{\log_b a}\right) + g(n)$
- Case 1, $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$,
- $T(n) = \Theta\left(n^{\log_b a}\right) + O\left(n^{\log_b a}\right) = \Theta(n^{\log_b a})$

- Case 2, $f(n) = \Theta(n^{\log_b a})$,
- $T(n) = \Theta\left(n^{\log_b a}\right) + \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(n^{\log_b a} \lg n\right)$

- Case 3, $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$,
- $T(n) = \Theta\left(n^{\log_b a}\right) + \Theta(f(n)) = \Theta(f(n))$

- Master theorem proved!