COMP 550
Algorithm and Analysis

Recurrence Relations

Based on CLRS Sec 4

Some slides are adapted from ones by prior instructors Prof. Plaisted and Prof. Osborne

Recurrence Relations

- An equation or inequality that describes a function over the integers or reals using the function itself

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \text{ (base case)} \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \text{ (recursive case)} \end{cases}$$

- Zero, one, or many functions may satisfy a recurrence
  - **Well-defined** if at least one satisfies, **ill-defined** otherwise

Algorithmic Recurrences

- $T(n)$ is an algorithmic recurrence if for every sufficiently large threshold constant $n_0 > 0$
  1. For all $n < n_0, T(n) = \Theta(1)$
  2. For all $n \geq n_0$, every path of recursion tree terminates on a defined base case within finite recursive invocations
- (1) implies for $n < n_0$, $0 \leq c_1 \leq T(n) \leq c_2$
- Not (2) implies the algorithm is incorrect!

Whenever a recurrence is stated without an explicit base case, we assume that the recurrence is algorithmic.

Algorithmic Recurrences

- Divide-and-conquer and recurrences

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

- Subproblems are not always of constant fraction of original problem

```
FindMax (A, n)
1.   if n ≤ 1
2.        return A[1]
3.   return max(A[n], FindMax(A,n-1))
```

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

Solving A Recurrence

- Substitution method
- Recursion-tree method
- Master method
- Akra-Bazzi method

Substitution Method

- Two step process
  - Guess the solution
  - Use mathematical induction to show that the guessed solution works
- Works well when you can guess the solution
- Guessing may not be always easy

Substitution Method

- Determine an asymptotic upper bound on $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$.
- Guess: $T(n) = O(n \lg n)$
  - It's better not to try prove Θ-bound directly. Why?
  - Can prove separate O- and Ω-bound instead.
- Note that $T(n) = O(n \lg n)$ means $T(n) \leq cn \lg n$ holds for $n \geq n_0$
  - We don't need to prove anything for $n < n_0$
  - n_0 should be reasonably small so that $T(n) = \Theta(1)$

Substitution Method

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

Assume that $T(m) \le c \cdot m \log m$ holds for all $n_0 \le m < n$ (n_0 to be defined later)

First consider, $n \ge 2n_0$

$$\begin{aligned} T(n) &\le 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + \Theta(n) \\ &\le 2(c(n/2)\lg(n/2)) + \Theta(n) \\ &= cn\lg(n/2) + \Theta(n) \\ &= cn\lg n - cn\lg 2 + \Theta(n) \\ &\le cn\lg n - cn + \Theta(n) \\ &\le cn\lg n . \end{aligned}$$

From CLRS

$\log_c(a \cdot b) = \log_c a + \log_c b$ (3.18 in book)
$\log_c\left(\frac{a}{b}\right) = \log_c a - \log_c b$ (not in book)

Need cn to dominate $\Theta(n)$.

Substitution Method

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

Assume that $T(m) \le c \cdot m \log m$ holds for all $n_0 \le m < n$ (n_0 to be defined later)

Now consider, $n_0 \le n < 2n_0$ (Induction base case). Looks different from recursive base case?

Pick n_0: Can we take $n_0 = 1$? Then, $T(1) \le c \cdot 1 \cdot \lg 1 = 0$. Possible?

Can we take $n_0 = 2$? Then, $T(2) \le 2c \lg 2$

For $n_0 = 2$, base case includes $2 \le n < 2 \cdot 2 = 4$. So, $n \in \{2,3\}$. $T(3) \le 3c \lg 3$

Take $c = \max(T(2), T(3))$. Then, $T(n) \le cn \lg n$, for any $n \ge n_0 = 2$

Substitution Method

- Base case handling is often ignored
  - Pretty much the same way to deal with
  - Take a n_0, then determine a large constant c so that $n_0 \le n < n_0'$ admits the inductive hypothesis (n_0' is $2n_0$ in prior example).

Substitution Method

- Steps:
  - Guess the solution
  - Prove the solution for large $n \geq {n_0}'$
  - Prove the solution for small $n, n_0 \leq n < n_0'$. (Usually done by taking $T(n) = \Theta(1)$)
  - Determine c (can be done by previous step)
- Omitting the last two steps are often fine!

Guessing Solution

- Try the solution to a similar-looking problem you've already solved
  - $T(n) = 2T\left(\frac{n}{2} + 17\right) + \Theta(n)$ looks like $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$, so try $T(n) \leq c \cdot n \, \log n$
- Try a looser solution and then narrow the bound from both ends
  - Prove the recurrence is $O(n^2)$ and $\Omega(n)$. Work from both directions to narrow the gap between upper and lower bounds
- Draw a recursion tree

Correct Guess But Math Fails

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

Guess $T(n) = O(n)$, so $T(n) \leq cn$

$$T(n) \leq 2c\left(\frac{n}{2}\right) + \Theta(1) = cn + \Theta(1)$$

The above does NOT imply $T(n) \leq cn$

Try this: subtract a lower-order term.

New Guess: $T(n) \leq cn - d$

$$T(n) \leq 2(c\left(\frac{n}{2}\right) - d) + \Theta(1)$$

$$= cn - 2d + \Theta(1)$$

$$\leq cn - d - \big(d - \Theta(1)\big)$$

$$\leq cn - d$$

Be careful about choice of c, d, and base cases

Pitfalls

- Do NOT use asymptotic notation in the inductive hypothesis.

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n)$$

Assume $T(n) = O(n)$

$$T(n) \leq 2 \cdot O\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n)$$

$$= 2 \cdot O(n) + \Theta(n)$$

$$= O(n)$$ ✗

The constant hidden by $O()$ may change.

To avoid pitfall,

Assume $T(n) \leq cn$

$$T(n) \leq 2 \cdot c\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n)$$

$$\leq cn + \Theta(n)$$

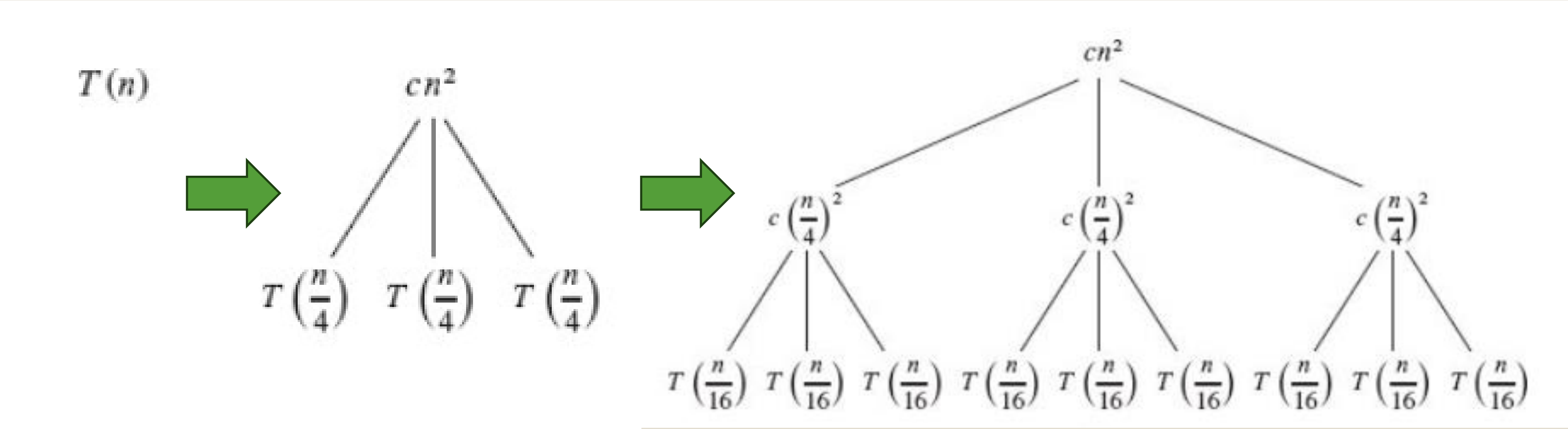
$$\nleq cn$$

Recursion-Tree Method

- Making a good guess is sometimes difficult with the substitution method.
- Use **recursion trees** to devise good guesses.
  - Better not to use it as direct proof (would need to be meticulous about expanding tree and summing costs)
  - For generating guess, some '**sloppiness**' is tolerable

Recursion Tree

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

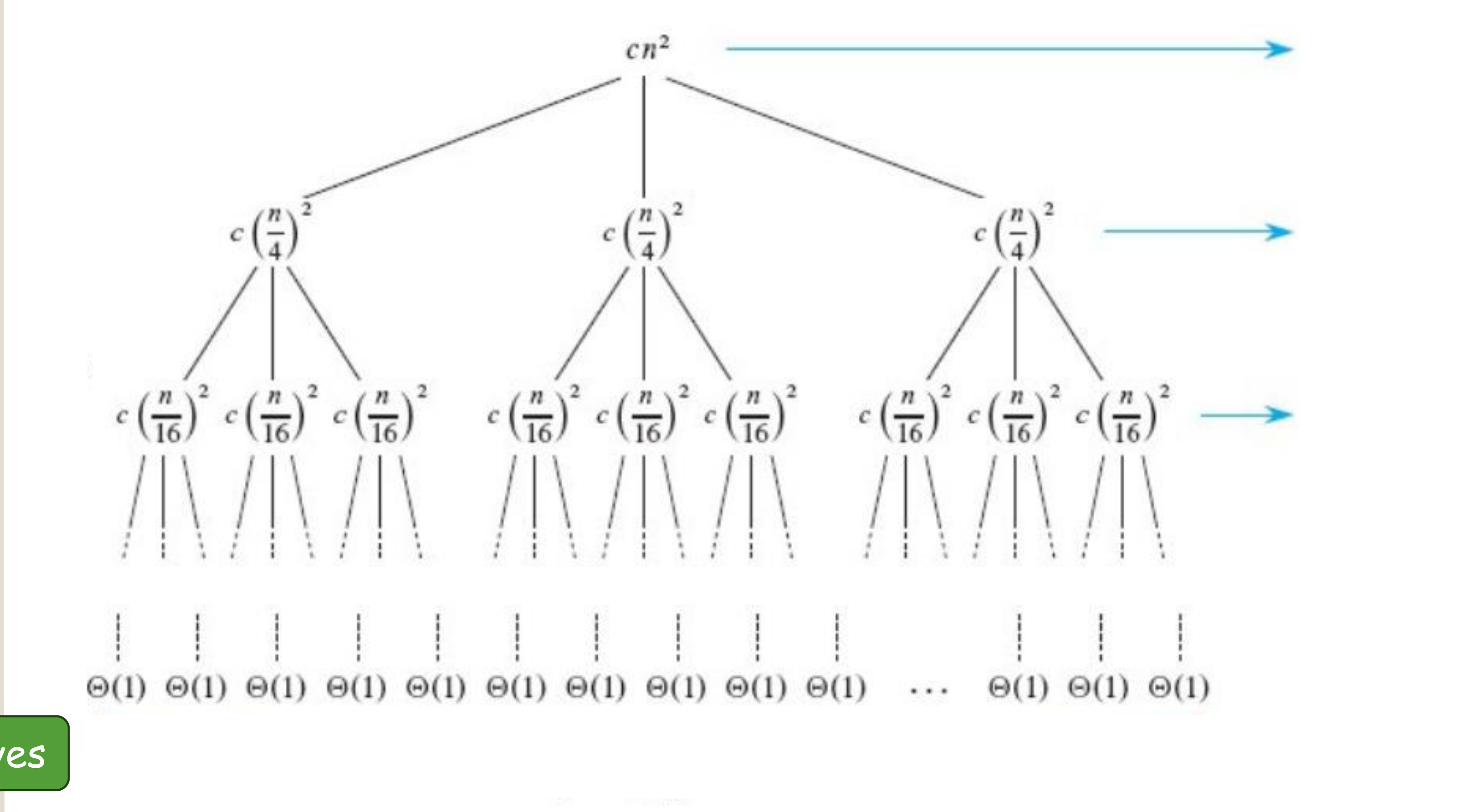


From CLRS

Recursion Tree

Number of levels

Cost per level



Number of leaves

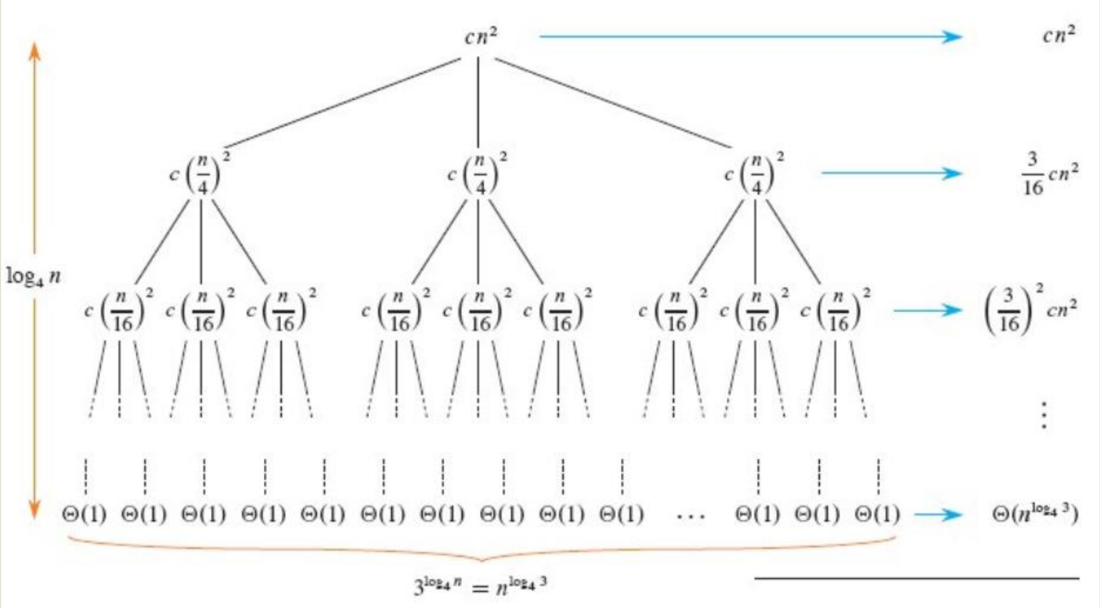
From CLRS

Recursion Tree

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

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

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



Recursion Tree

Goal: evaluate $\sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i$

$$\sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i = \frac{1}{1-\frac{3}{16}} = \frac{16}{13}$$

The summation

$$\sum_{k=0}^{\infty} x^k = 1 + x + x^2 + \cdots + \infty$$

is a *geometric series*. If $|x| < 1$, then

$$\sum_{k=1}^{n} x^k = \frac{1}{1-x}$$

Evaluating the sums

- Appendix A: Summations

Recursion Tree

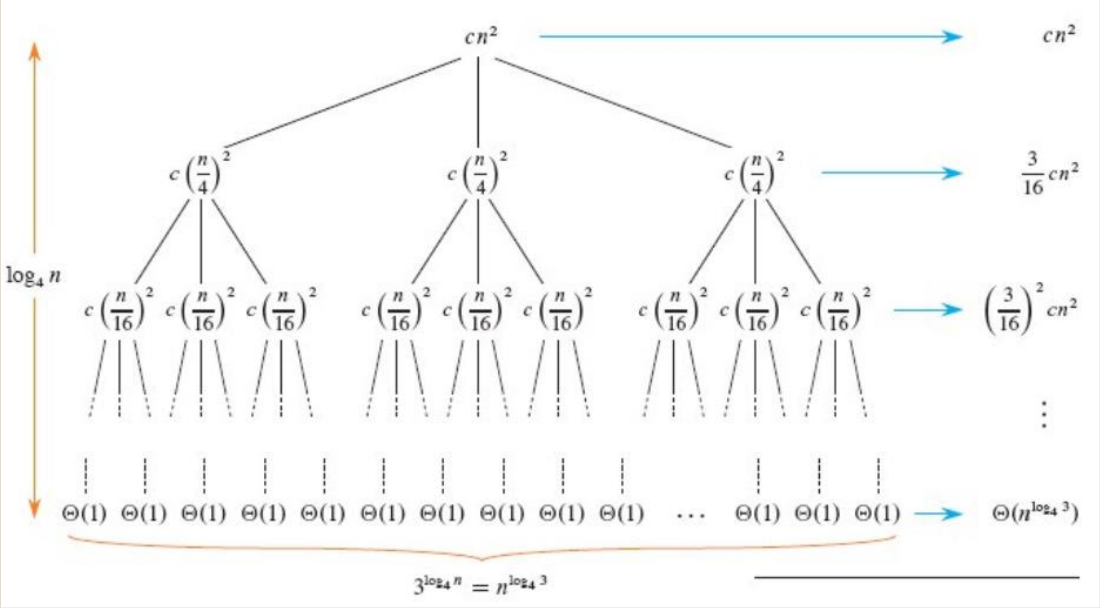
$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n-1} cn^2 + \Theta(n^{\log_4 3})$$

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

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

$$= \frac{16}{13}cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$



Recursion Tree: Verify with Substitution

- Use substitution method to prove $T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$ is $O(n^2)$

Assume the constant in $\Theta(n^2)$ is c, i.e., $\Theta(n^2) = cn^2$

Assume, $T(m) \leq dm^2$ for all $n_0 \leq m < n$.

We need to prove $T(n) \leq dn^2$

This calculation works for $n \geq 4n_0$. Why?
Show for $n_0 \leq n < 4n_0$ (Recurrence base case)

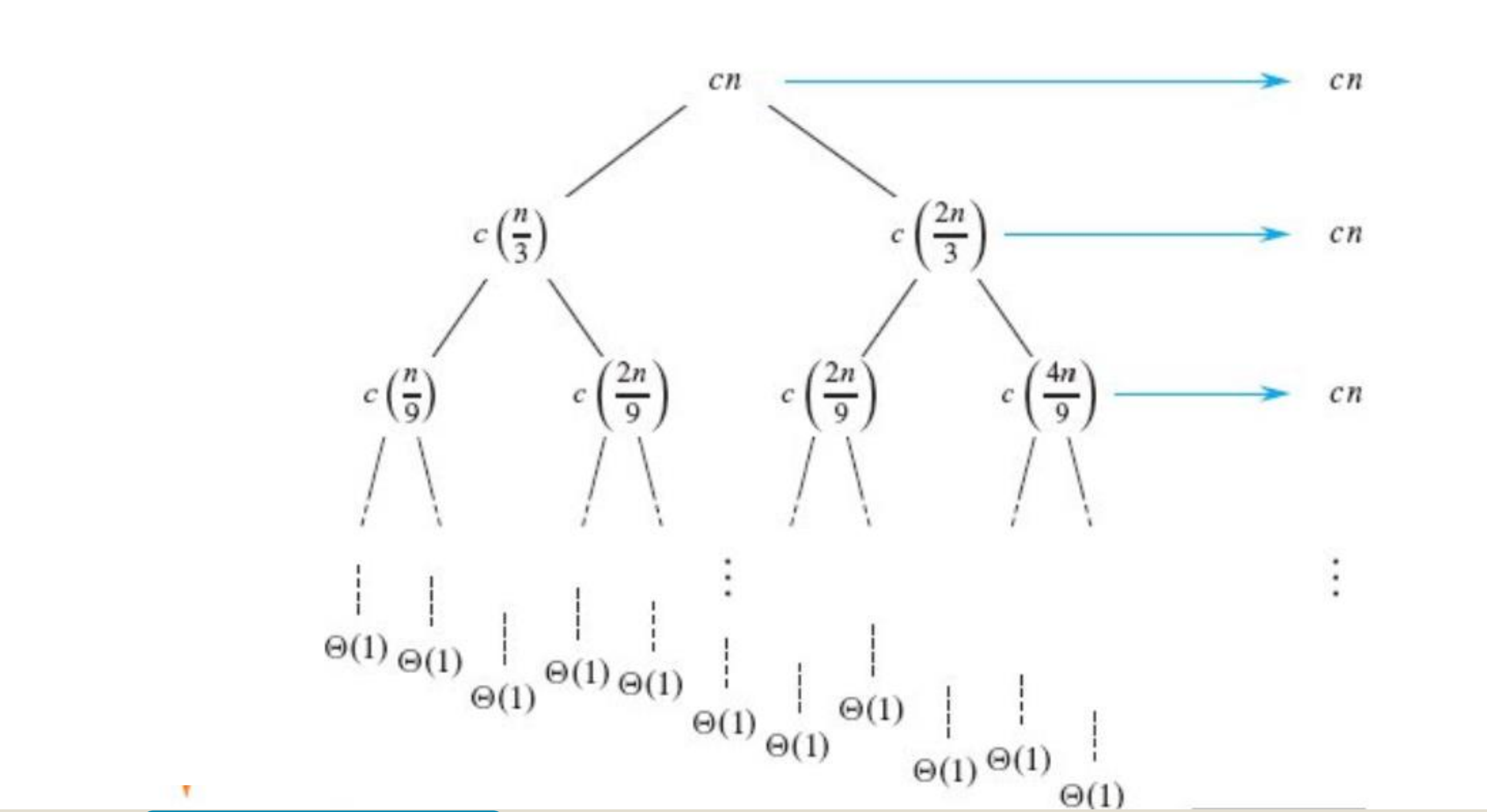
Now consider n,

$$T(n) = 3T\left(\frac{n}{4}\right) + cn^2 \leq 3d\left(\frac{n}{4}\right)^2 + cn^2 = \left(\frac{3d}{16} + c\right)n^2$$

Can we pick value of d so that $\frac{3d}{16} + c \leq d$?

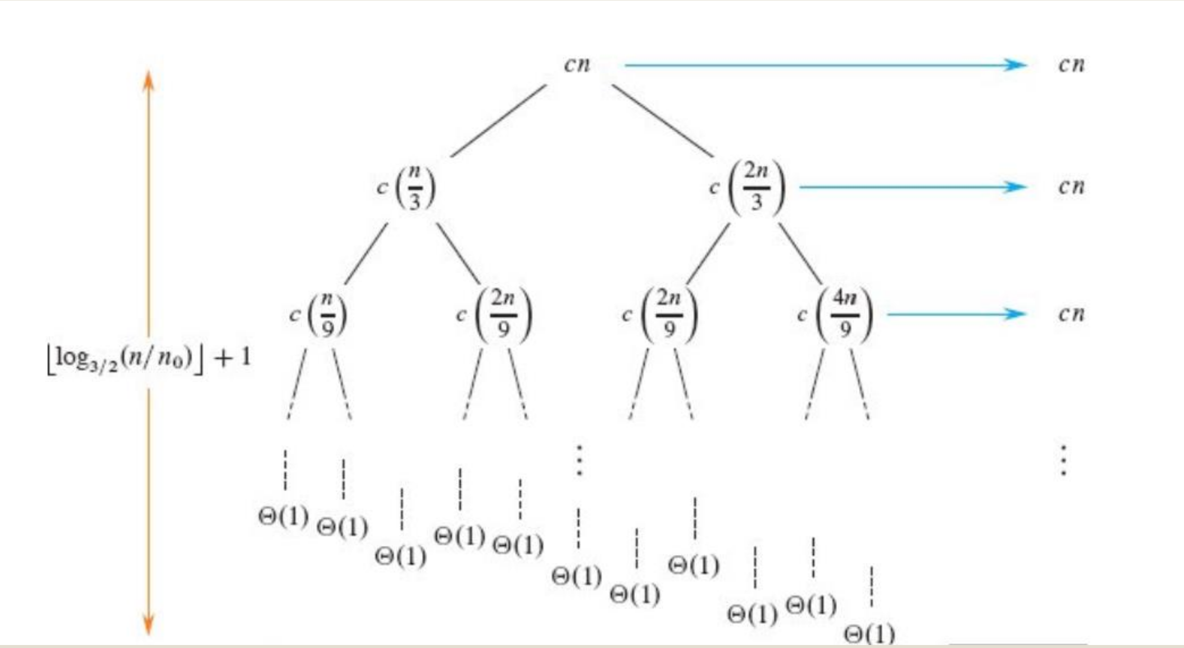
Recursion Tree

An irregular example: $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n)$



From CLRS

Recursion Tree



- Height of the tree = $\Theta(\lg n)$
- Cost per level = $O(n)$
- Guess, $T(n) = O(n \lg n)$
  - Try this by substitution method
- How many leaves in total?
  - Assuming complete binary tree, # of leaves $= 2^{\lfloor \log_{3/2} n \rfloor + 1} + 1 \leq 2n^{\lg_{\frac{3}{2}} 2} = O(n^{1.71})$
  - This is larger than tight running time $O(n \lg n)$
- Takeaway: Over-approximating #of leaves may cause running time to be dominated by the costs of leaves leading to a loose running time bound.

Master Method

- A master recurrence is in form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, where $a > 0$ and $b > 1$ are constants
- Divides a problem of size n into a subproblems, each of size $\frac{n}{b}$
- $aT\left(\frac{n}{b}\right)$ actually means $a'T\left(\left\lfloor\frac{n}{b}\right\rfloor\right) + a''T\left(\left\lceil\frac{n}{b}\right\rceil\right)$ for $a', a'' \geq 0$ and $a' + a'' = a$
- $f(n)$ = cost of dividing and combining.
- $f(n)$ is referred to as the driving function.

Master Method

- <u>Theorem 4.1 (Master Theorem)</u>:
  - Solves master recurrences, $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
  - 3 cases based on comparing $f(n)$ with $n^{\log_b a}$
  - $n^{\log_b a}$ is called the watershed function

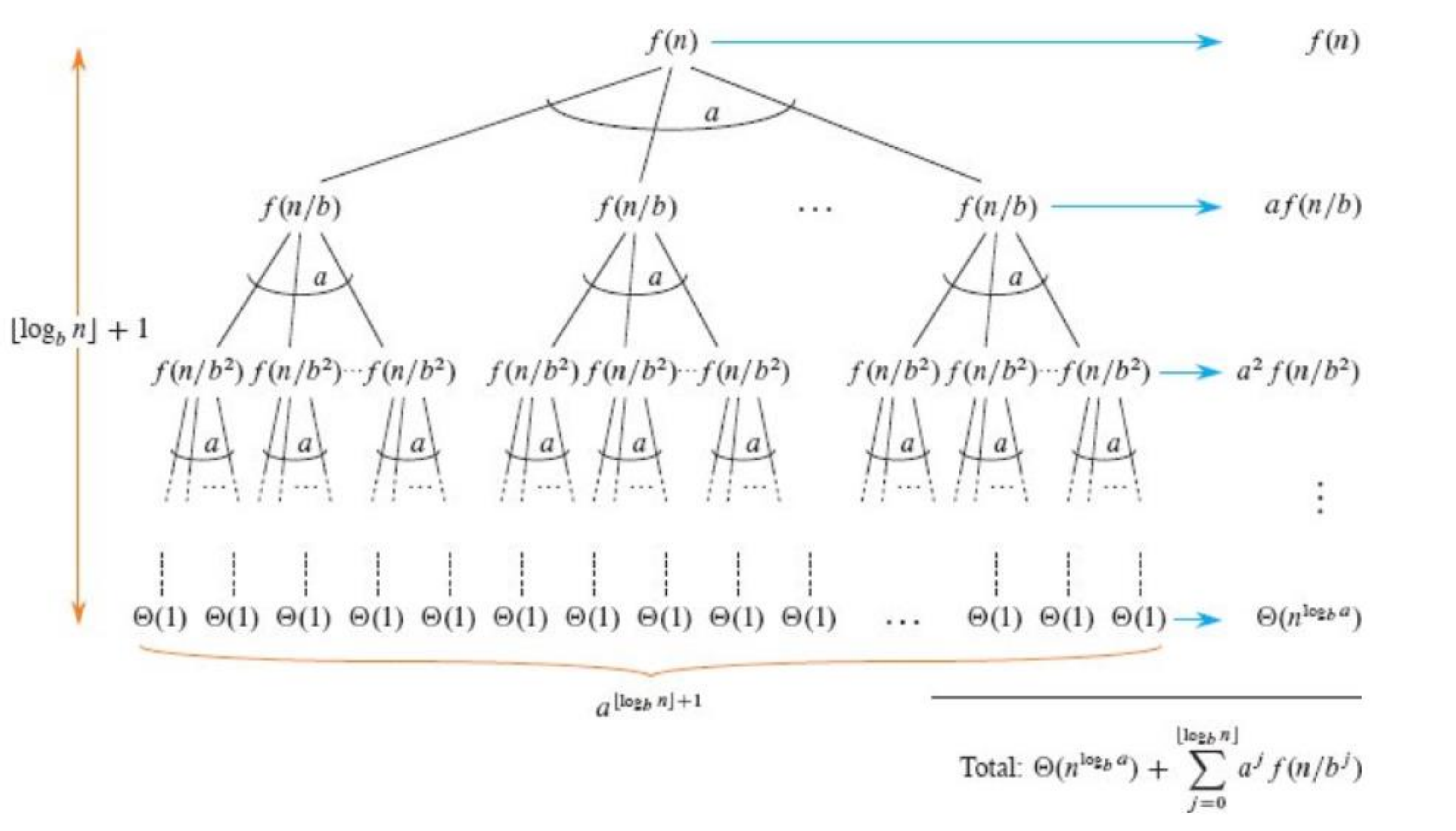
Master Method

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

	Condition	Solution
Case 1	There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = \Theta(n^{\log_b a})$
Case 2	There exists constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$	$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
Case 3	There exists constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$,	$T(n) = \Theta(f(n))$

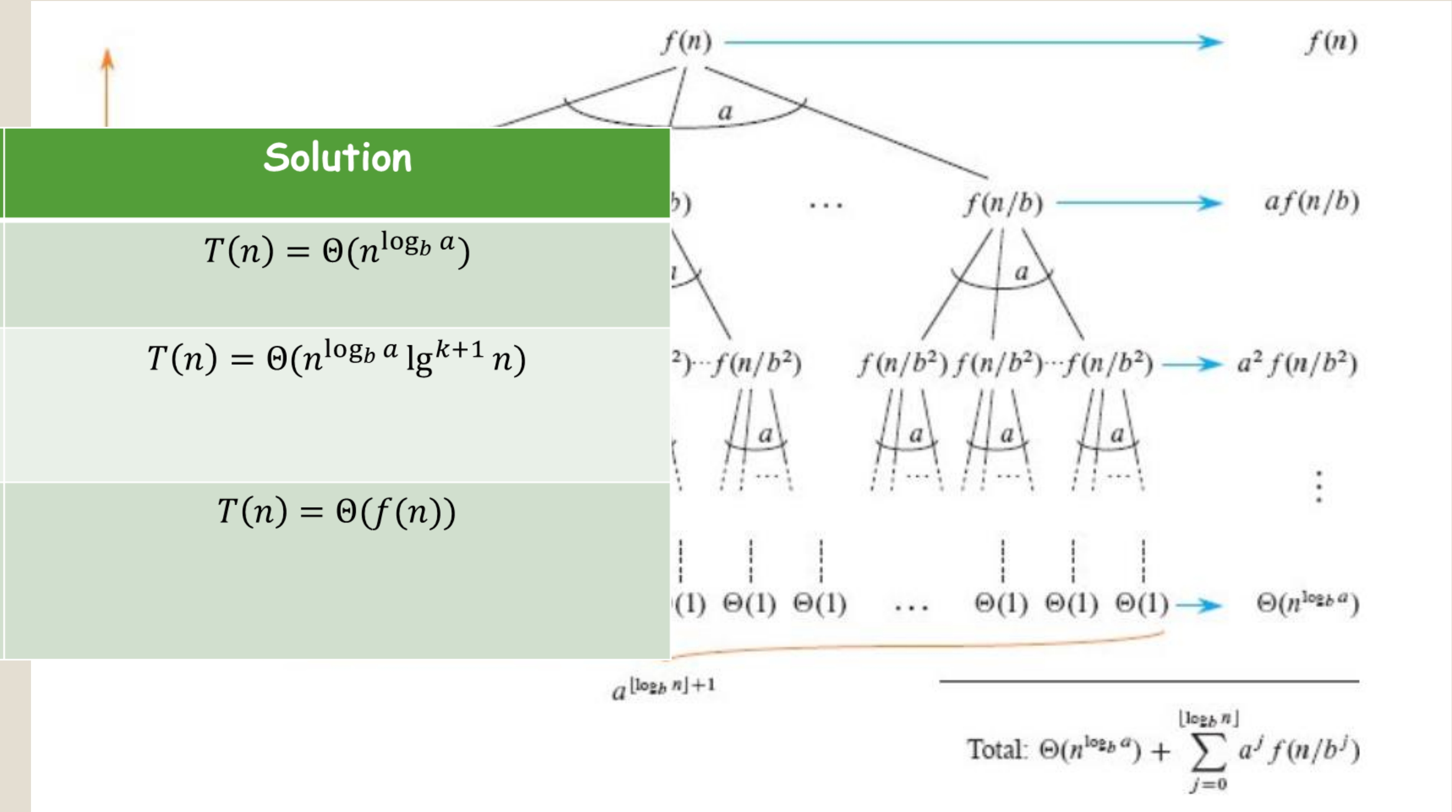
Master Method

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



Master Method

	Condition	Solution
Case 1	There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a-\epsilon})$	$T(n) = \Theta(n^{\log_b a})$
Case 2	There exists constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$	$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
Case 3	There exists constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a+\epsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$,	$T(n) = \Theta(f(n))$

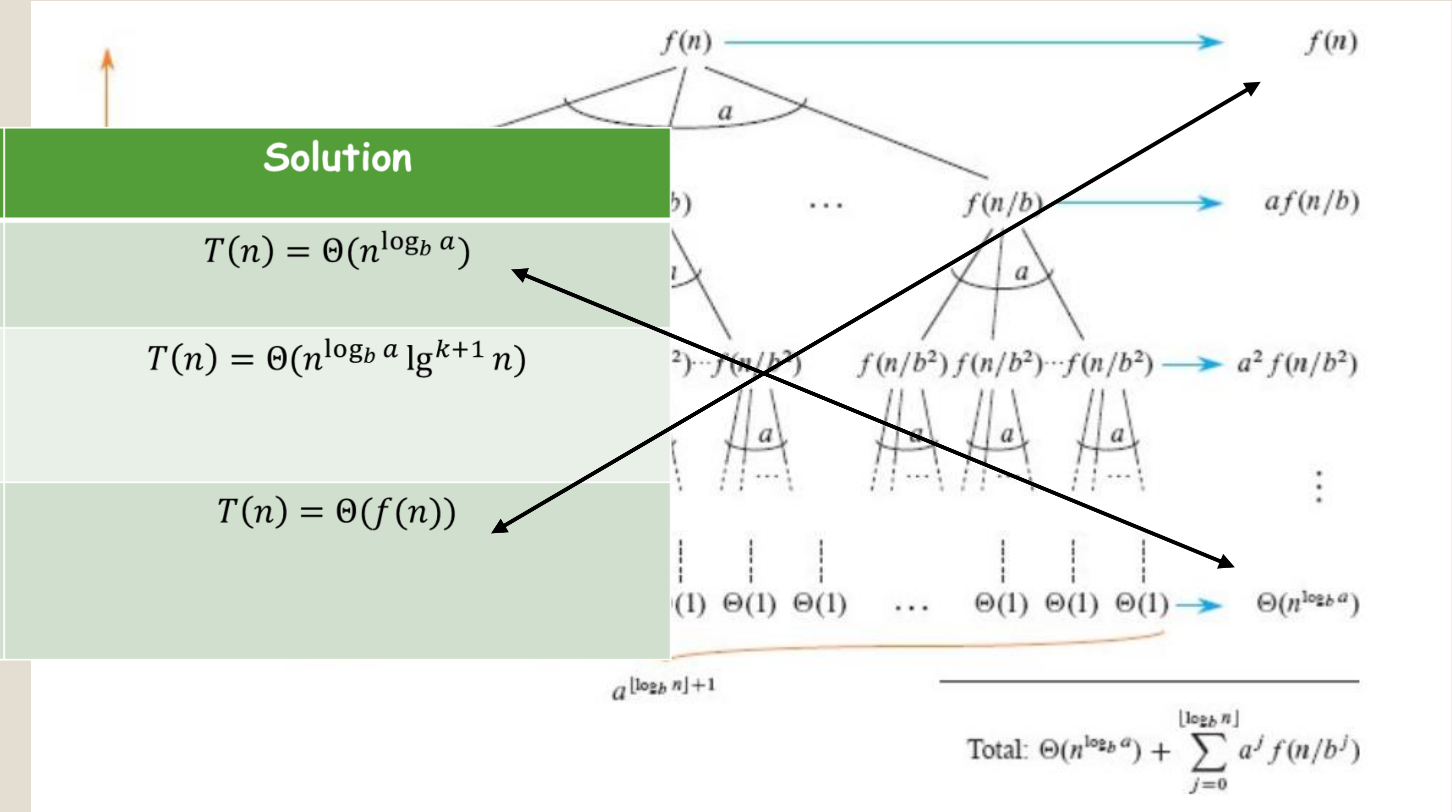


Any idea what' happening in each case?

(Hint: look at the solution and the cost of the levels in recursion tree)

Master Method

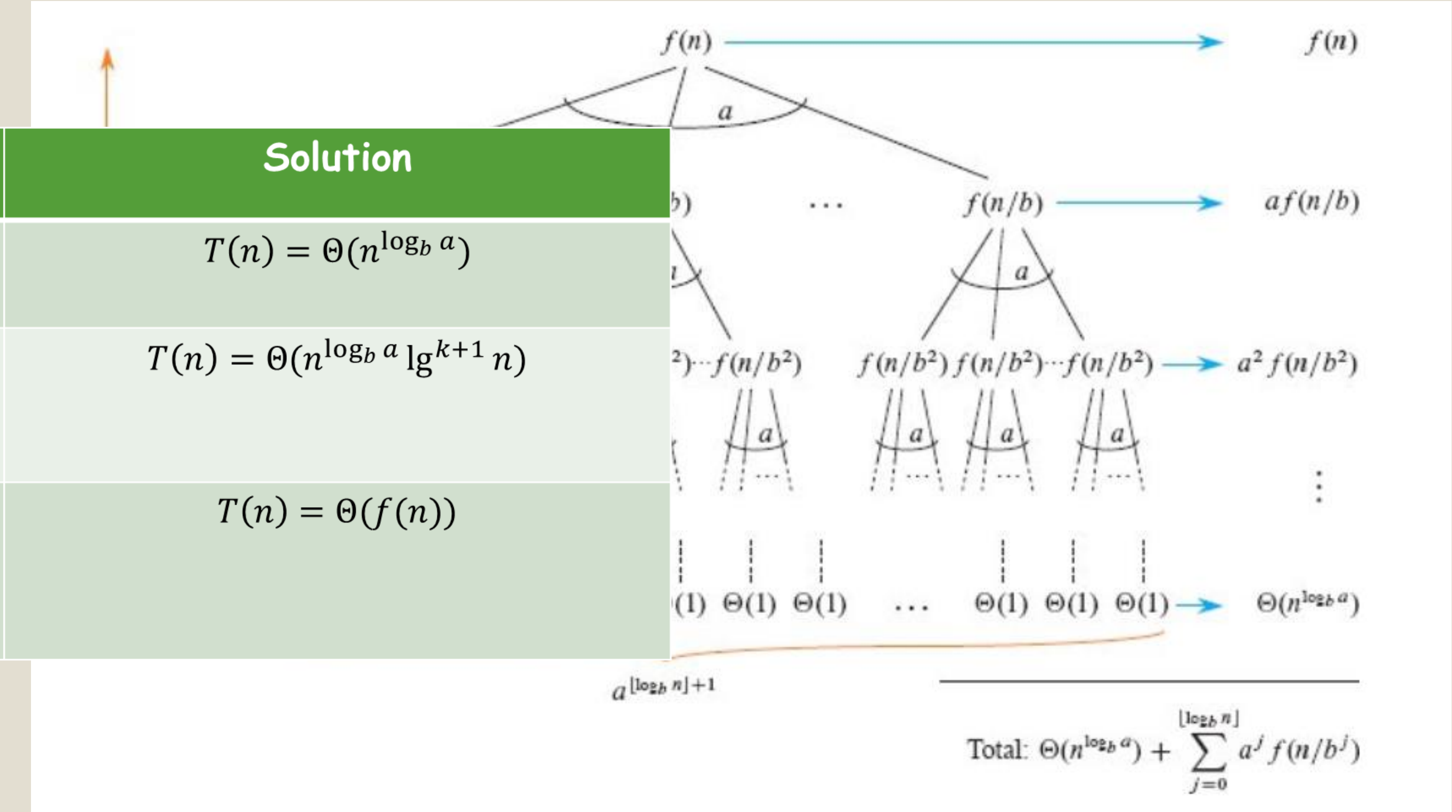
	Condition	Solution
Case 1	There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = \Theta(n^{\log_b a})$
Case 2	There exists constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$	$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
Case 3	There exists constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$,	$T(n) = \Theta(f(n))$



Case 1: Running time is dominated by leaves. When this happens?

Case 3: Running time is dominated by the root. When this happens?

Master Method



	Condition	Solution
Case 1	There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = \Theta(n^{\log_b a})$
Case 2	There exists constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$	$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
Case 3	There exists constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$,	$T(n) = \Theta(f(n))$

Case 2: Each level (with internal nodes) has asymptotically same cost (Just like merge sort and closest pair of points)

Total running time = cost per level * number of levels

Master Method

Case 1: There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$

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

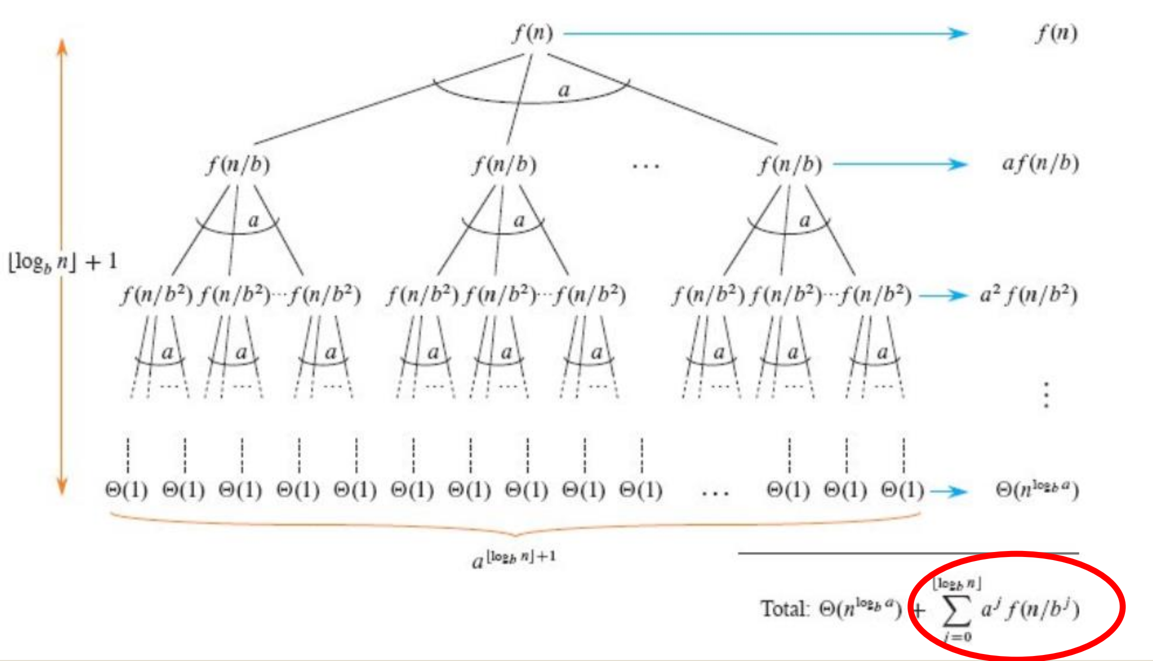
- $f(n)$ is polynomially smaller than $n^{\log_b a}$
  - $f(n)$ is asymptotically smaller than $n^{\log_b a}$ by a factor of $O(n^{\epsilon})$ for $\epsilon > 0$

Master Method

<u>Case 1</u>: There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a-\epsilon})$

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

- Per-level cost increases as we go down the recursion tree
- Cost of leaves dominates costs of internal nodes
  - Cost of leaves = $\Theta(n^{\log_b a})$
  - Cost of internal nodes = $O(n^{\log_b a})$
  - Total cost =$\Theta(n^{\log_b a})$



Master Method

Case 1: There exists constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$

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

Example: $9T\left(\frac{n}{3}\right) + n$

- Dissect the Recurrence

$f(n) = n; \quad a = 9, b = 3: \quad n^{\log_b a} = n^{\log_3 9} = n^2$

- Check case requirement

$f(n) = n = O(n^{2-1})$

- Give the solution

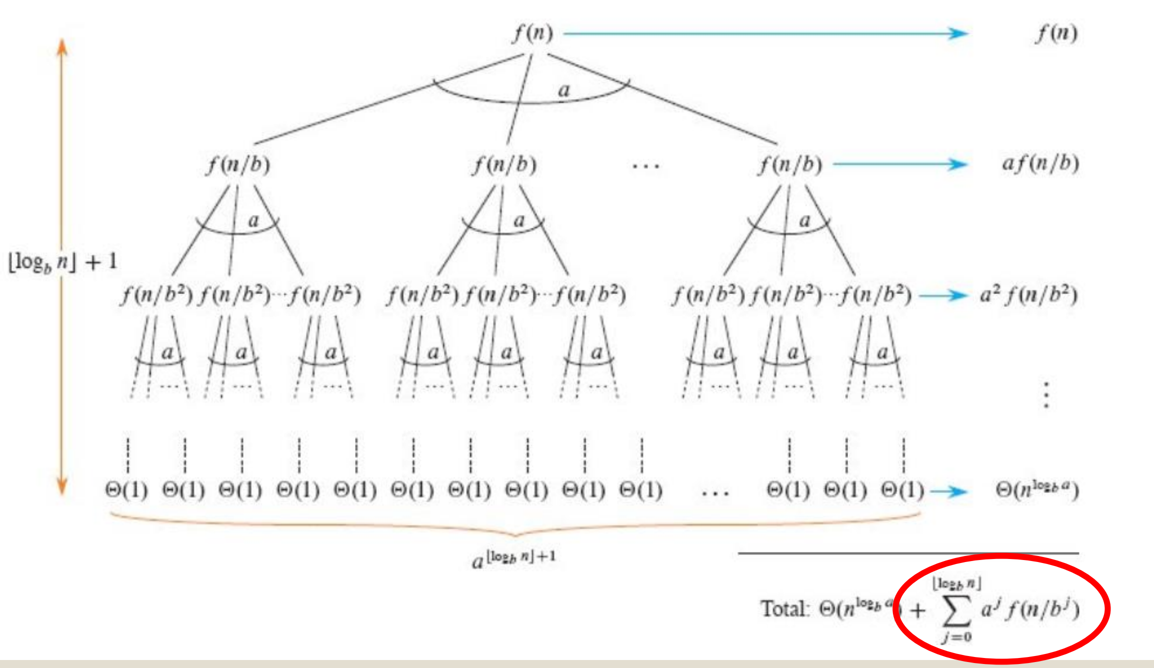
$T(n) = \Theta(n^2)$

Master Method

Case 2: There exists constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$$

- $f(n)$ is within a polylog factor of $n^{\log_b a}$
- Cost at each level with internal nodes = $f(n) = \Theta\left(n^{\log_b a} \lg^k n\right)$
- Cost of all internal nodes $=\Theta\left(n^{\log_b a} \lg^{k+1} n\right)$



Master Method

Case 2: There exists constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$$

Example: $T(n) = 27T\left(\frac{n}{3}\right) + n^3 \log_2 n$

- Dissect the Recurrence

$f(n) = n^3 \lg n; \quad a = 27, b = 3: \; n^{\log_b a} = n^{\log_3 27} = n^3$

- Check case requirement

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

- Give the solution

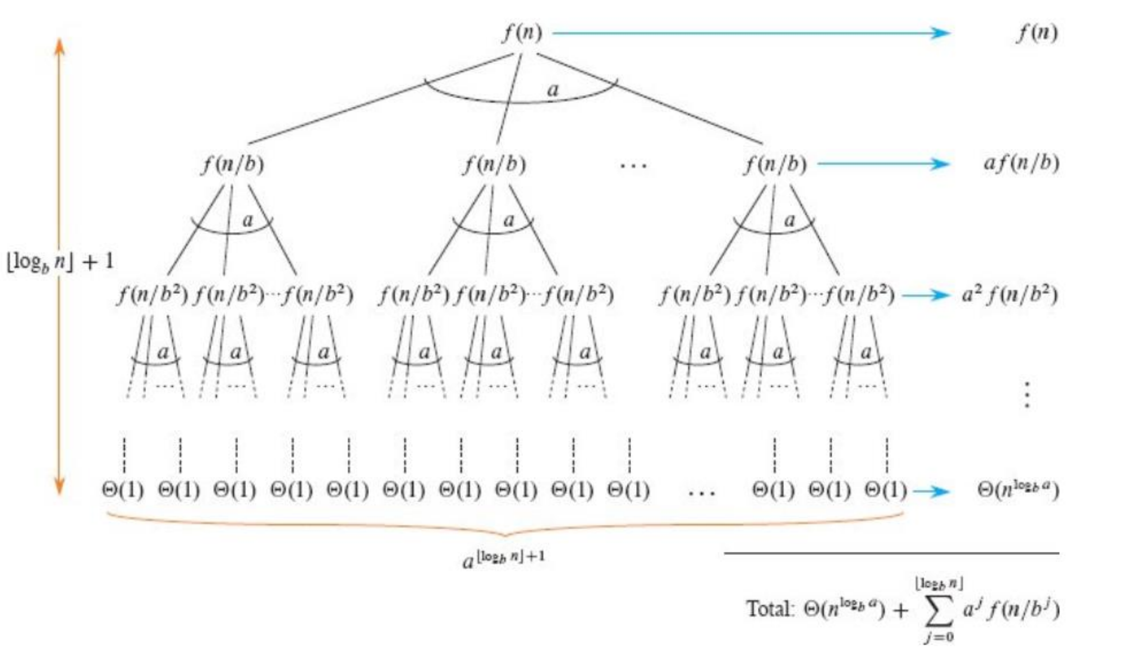
$T(n) = \Theta(n^3 \lg^2 n)$

Master Method

Case 3: There exists constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a+\epsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$,

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

- Mirrors Case 1, $f(n)$ is polynomially greater than $n^{\log_b a}$
- $af(n/b) \leq cf(n)$: Regularity condition
- Per-level cost decreases as we go down the recursion tree
- Root's cost = $\Theta(f(n))$, Other's cost = $O(f(n))$
- Total cost = $\Theta(f(n))$



Master Method

Case 3: There exists constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$,

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

Example: $T(n) = 5T\left(\frac{n}{2}\right) + n^3$

- Dissect the Recurrence

$f(n) = n^3; \quad a = 5, b = 2: \; n^{\log_b a} = n^{\log_2 5} = n^3$

- Check case requirement

$f(n) = n^3 = \Omega(n^{\log_2 5})$ and $5f\left(\frac{n}{2}\right) = 5\left(\frac{n}{2}\right)^3 \leq \frac{5}{8}n^3$

- Give the solution

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

Master Method: Not Applicable Case

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

- Dissect the Recurrence

$f(n) = n/\lg n\,; \quad a = 2, b = 2\text{:} \;\; n^{\log_b a} = n^{\log_2 2} = n$

- Check case requirement

$f(n) = \frac{n}{\lg n} \neq O(n^{1-\epsilon})$, i.e., $\frac{n}{\lg n}$ is not polynomially smaller than n

  - Case 1 not applicable

$f(n) = n \lg^{-1} n\,, k < 0$ required to match Case 2.

  - Case 2 not applicable

$f(n) = n/\lg n \neq \Omega(n^{1+\epsilon})$

  - Case 3 not applicable

Thank You!