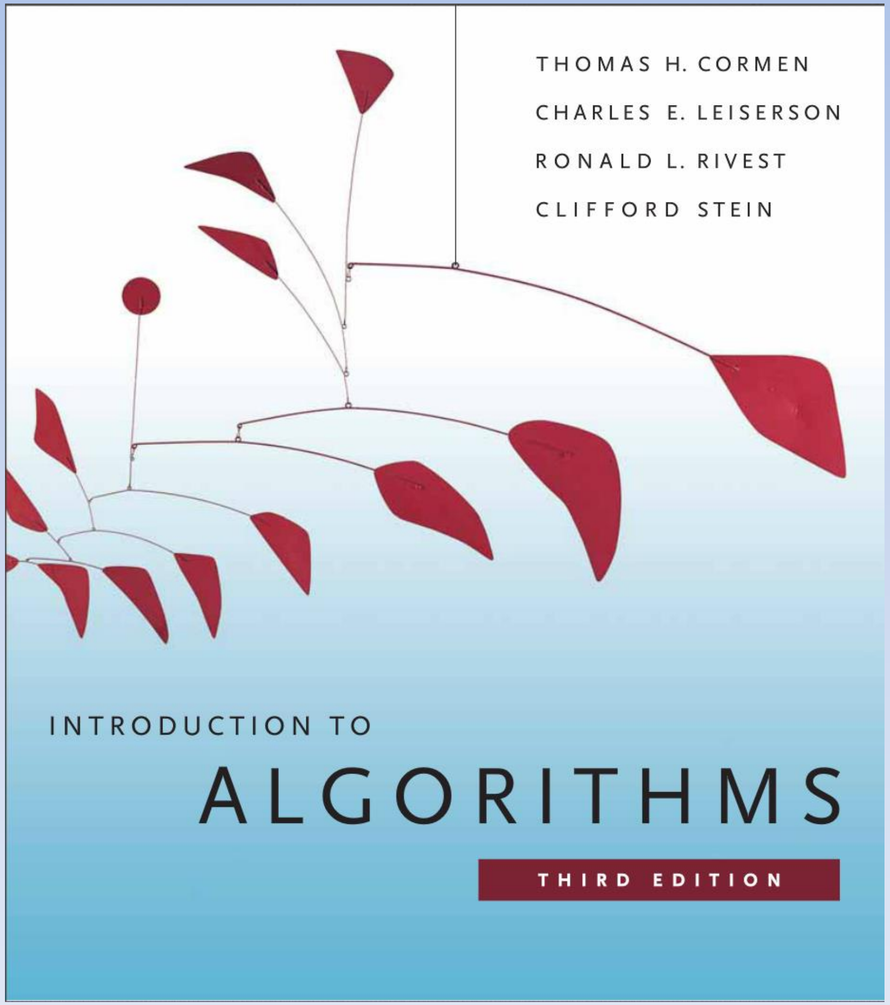
Design and Analysis of Algorithms

Topic: Divide & Conquer



Divide & Conquer w/ Recurrences

- Analysis of Divide & Conquer
  - Problems broken down into some number of smaller problems:
    - Problem of size N broken down into **a** subproblems
    - Each subproblem is of size N/b
  - Problems then need to be merged doing some of amount of work…

Mergesort

- Each problem broken down into 2 subproblems
- Each subproblem is ½ the size of the original problem
- O(n) work down merging subproblems.
- $T(N) = 2T(N/2) + O(N)$
- Review methods for solving these recurrences!

Guess & Prove

- $T(N) = 2T(N/2) + O(N)$
- GUESS: $O(N\lg N)$

- Prove w/ Induction
  1. Prove a base case
  2. Prove $T(K)$ w/ assuming $T(k-1)$

$T(N) = 2T(N/2) + n$ w/ Constants Ignored

- Prove $T(N) \leq N\lg N$
- Start w/ Inductive Step:
  - $T(K) = 2T(K/2) + K$
- By our assumption:
  - $T(K/2) = (k/2)\lg(k/2) + k/2$
- SO:
  - $T(K) = 2(\ (k/2)\lg(k/2) + k/2\ )\ + K$
  - $= K\lg(K/2) + 2K$
  - $= K\lg K - K\lg 2 + 2K$
  - $\leq K\lg K + cK$
  - $\leq ck\mathrm{Lg}K$
  - $= O(K\lg K)$

$T(N) = 2T(N/2) + n$ w/ Base Case

- A little care required w/ Base Case
- $T(2) = 2\lg 2 = 2$
  - correct

Good Guesses

- Unfortunately, Good Guesses are Difficult!
- No general way to guess!

- If recurrence is similar to one seen before, guessing similar solution reasonable!

Quick Question: 4.3-1

- Show $T(N) = T(N-1)+N$ is $O(N^2)$

Quick Question: 4.3-1

- Show T(N) = T(N-1)+N is O(N^2)

- Guessing T(N) ≤ cn^2 for some constant c>0

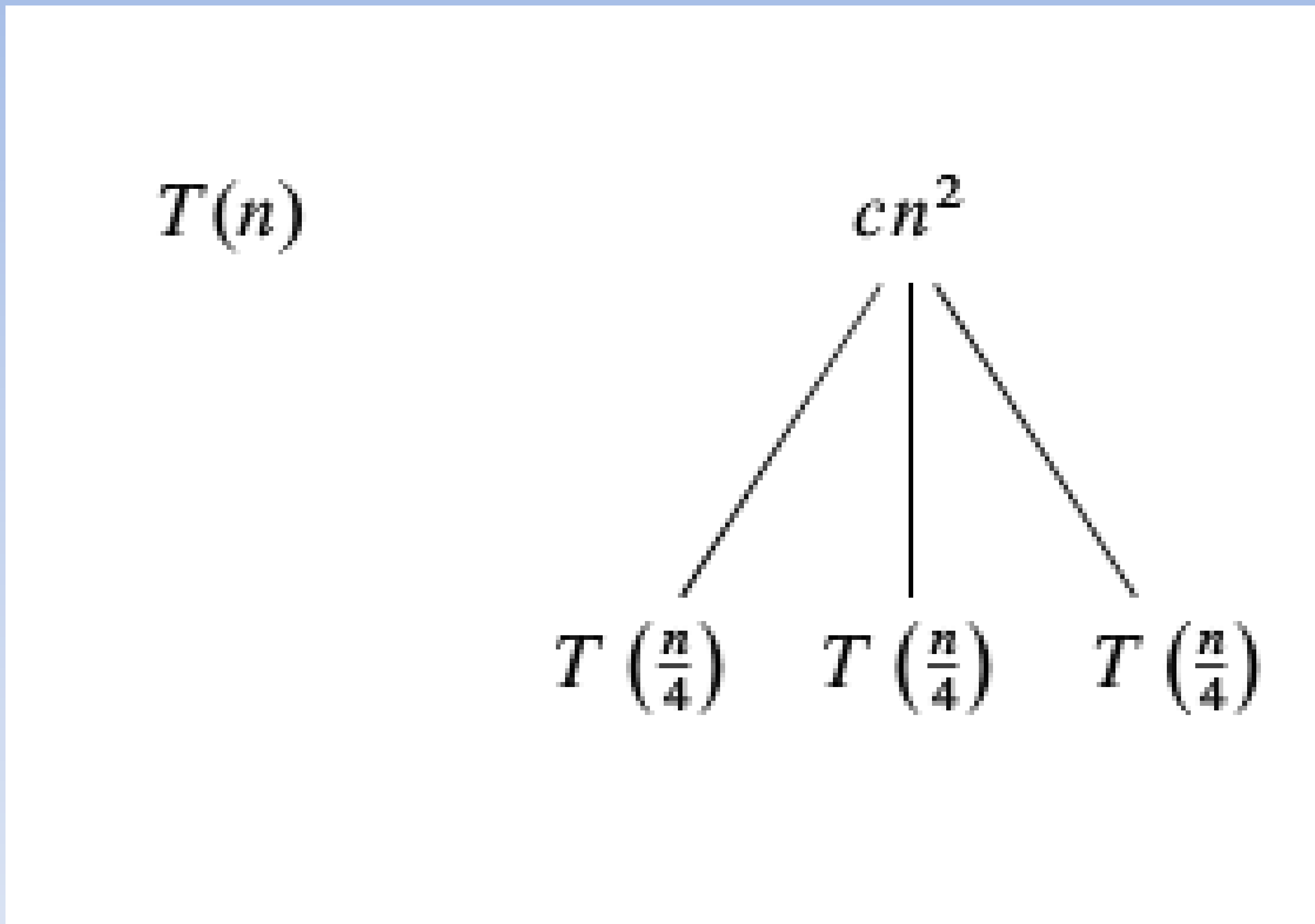
$$T(N) = T(N-1) + N$$
$$\leq c(N-1)^2 + N$$
$$\leq c(N^2 - 2N + 1) + N$$
$$\leq cN^2 - 2cN + c + N$$
$$\leq cN^2 - c(1 - 2N) + N$$

- $cN^2 - c(1-2N) + N \leq cN^2$
  - w/ c ≥ N/(2N-1)
- T(1) = 1 $\leq c1^2$
- SO:
  - n_0 = 1
  - c = 1

Solving Recurrences w/ Recursion-Tree Method

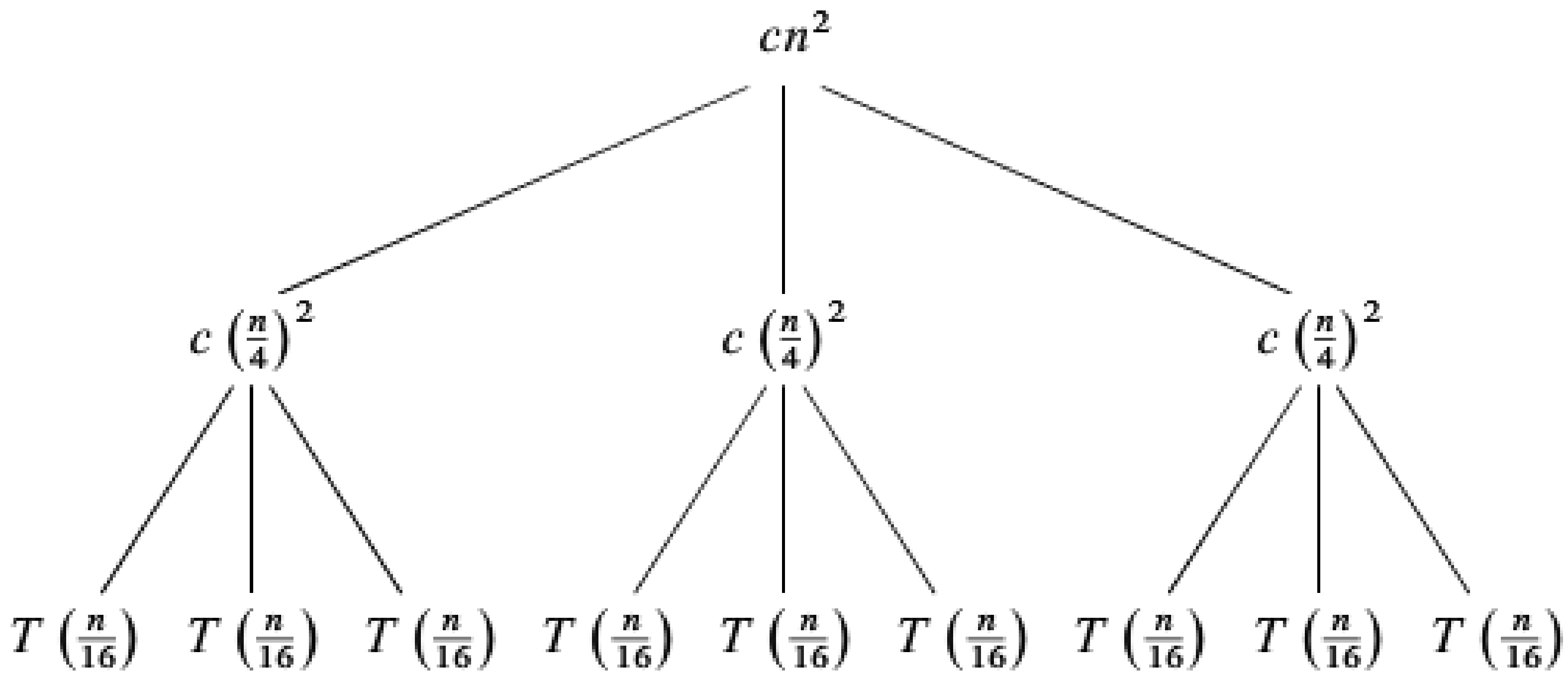
- Drawing the recursion tree helps to understand the recursive relations
- Recursion tree helps devise a good guess!
- Each node represents the cost of a single subproblem somewhere in the set of recursive invocations.
- Sum costs within each level of tree
- Count the number of levels in tree

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



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

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



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

(c)

Depth of Tree

- N
  - N/4
    - $N/4^2$
      - $N/4^3$

```
Depth = 0
  while N>=b:
    Depth +=1
    N = N/b
print Depth
```

Depth of Tree

- N
  - N/4
    - $N/4^2$
      - $N/4^3$

Depth = 0

while N>=b:

Depth +=1

N = N/b

print Depth

- depth = $\text{Log}_4 N$
  - $4^{\text{depth}} = N$

Depth of Tree

- $N=4^k$
  - $N/4 = 4^{k-1}$
    - $N/4^2 = 4^{k-2}$
      - $N/4^3 = 4^{k-3}$

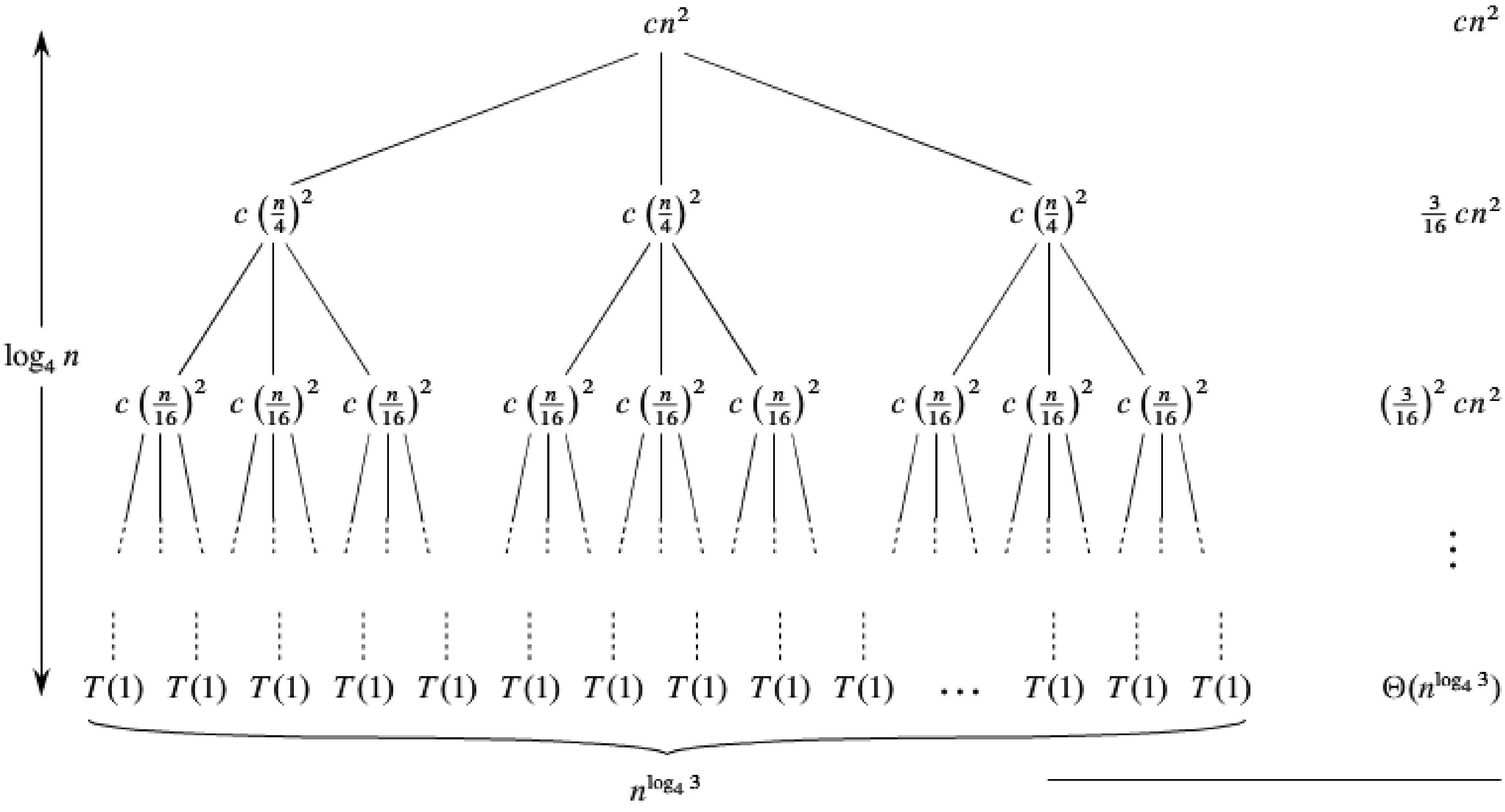
Depth = 0

while 4^k >=b:

Depth +=1

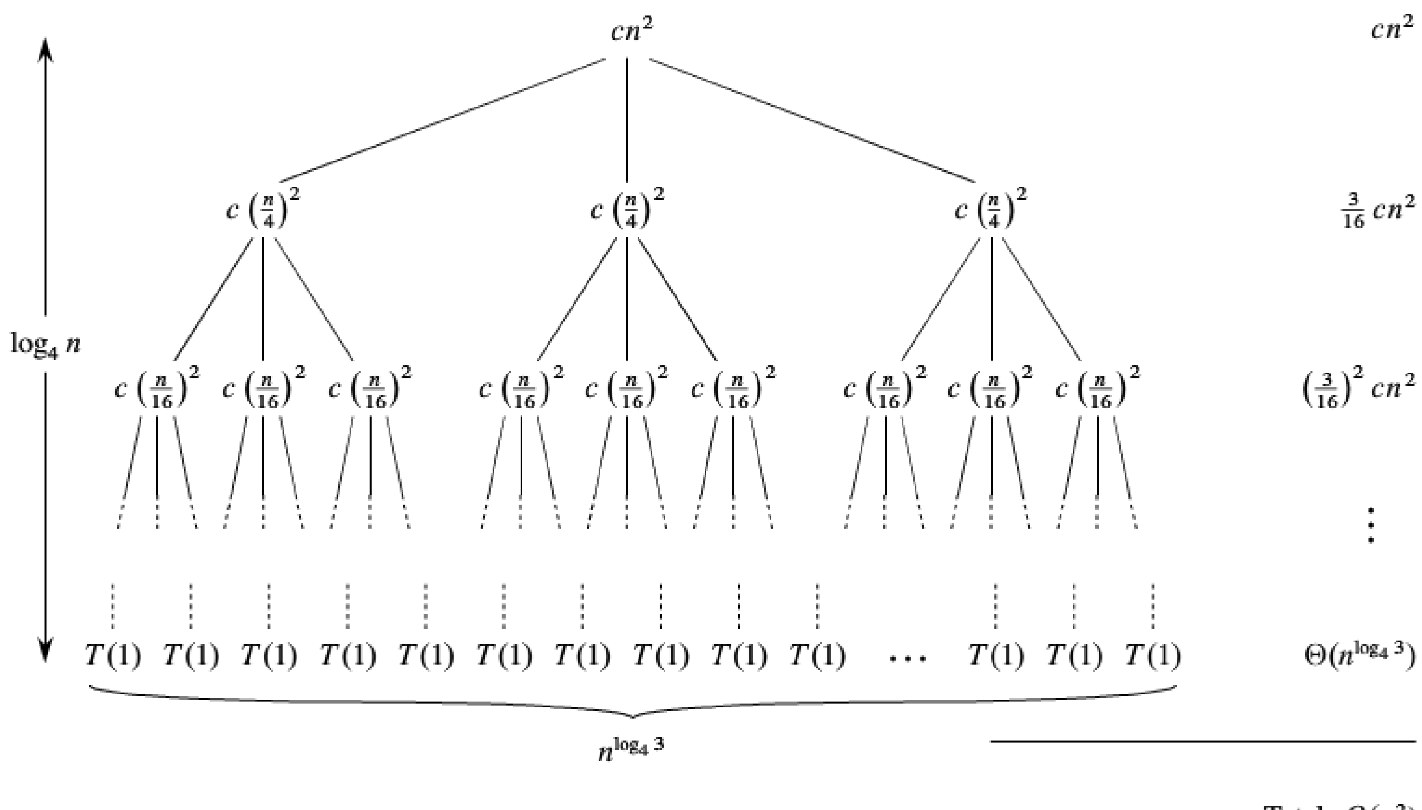
N = N/b

- Depth = k



cn^2
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$\log_4 n$
$T(1)$
$n^{\log_4 3}$
cn^2
$\frac{3}{16}\,cn^2$
$\left(\frac{3}{16}\right)^2 cn^2$
$\Theta(n^{\log_4 3})$
Total: $O(n^2)$

(d)



(d)

NOTE: Leaves = $branching^{height} = 3^{\log_4 n}$

$$3^{\log_4 n} = n^{\log_4 3}$$

$$\log_4 3^{\log_4 n} = \log_4 n^{\log_4 3}$$

$$\log_4 n \log_4 3 \quad = \log_4 3 \log_4 n$$

Uneven Tree:

$$T(N) = T(N/3) + T(2n/3) + cn$$

Uneven Tree: $T(N) = T(N/3) + T(2n/3) + cn$

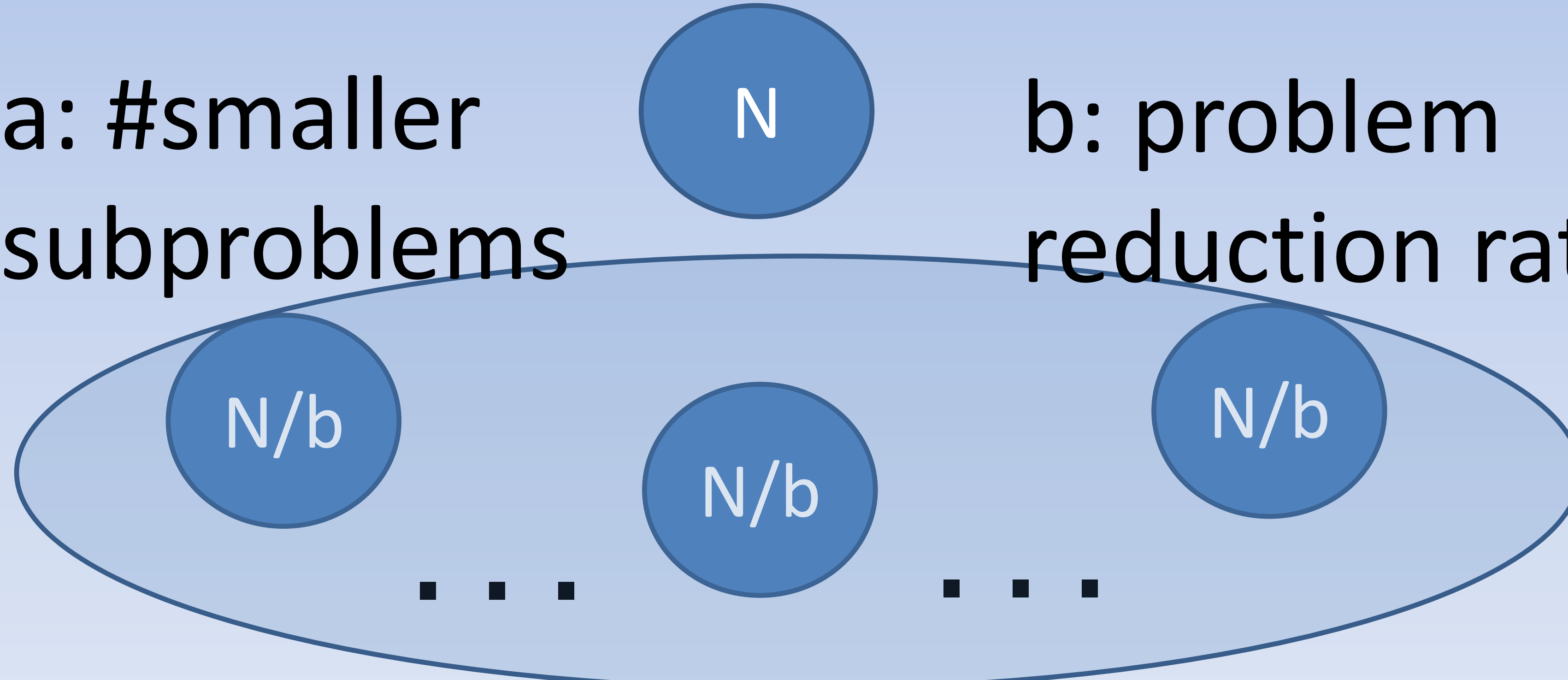
- Need to look for the longest path

- n -> (2/3)n -> $(2/3)^2$n -> $(2/3)^3$n
- Since $(2/3)^k n=1$ when $k=\log_{3/2}n$
  - Height(T) = $\log_{3/2}n$
- Cost at each level = n
- Total cost = $cn*\log_{3/2}n = O(n\lg n)$

Solving Recurrences w/ Master Method

- Perhaps we can come up with some general formula for recurrences:

a: #smaller subproblems



b: problem reduction rate

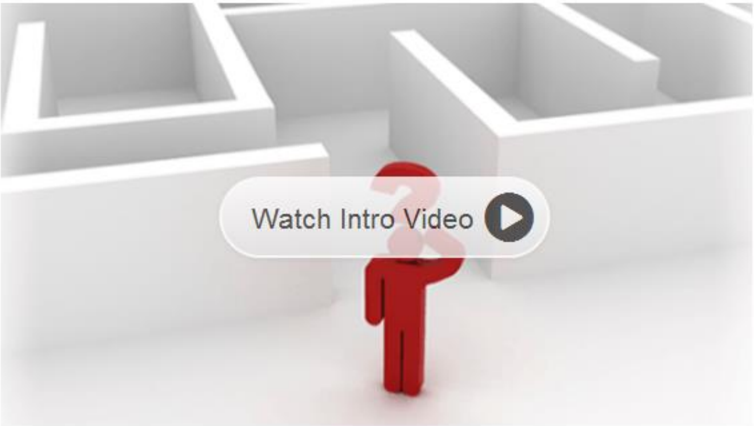
Tim Roughgraden w/ Coursera

Stanford

Algorithms: Design and Analysis, Part 1

In this course you will learn several fundamental principles of algorithm design: divide-and-conquer methods, graph algorithms, practical data structures (heaps, hash tables, search trees), randomized algorithms, and more.

Preview Lectures



coursera

Catalog

Tim Roughgarden

Associate Professor

Computer Science

Stanford University

Solving Recurrences w/ Master Method

- Perhaps we can come up with some general formula for recurrences:
- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!
  - Book based around $n^{\log_b a}$
  - Look at relationship between $\log_b a$ and d

Solving Recurrences w/ Master Method

- Perhaps we can come up with some general formula for recurrences:
- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!
- 3 Cases to think about!

Master Method w/ Case 1

- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!

- Assume $a = b^d$
  - b^d subproblems each level
  - n^d work to merge
- $O(N^d \log N)$
  - Balance between work at each level and depth

- Example:
  - $a = 2$, $b=2$, $d=1$
  - $b^d = 2^1 = 2 = a$
  - $N\lg_2 N$

Master Method w/ Case 2

- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!

- Assume $a < b^d$
  - Number of subproblems is growing slowly
- $O(N^d)$
  - Total work dominated by the combine step at root!
  - Good (subproblem simplification) is beating evil (subproblem proliferation)!

- $T(N) = 2T(N/2) + O(n^2)$
  - Less work is being done at each level
  - Work at level 2 is $2T(N/2) = 2(N/2)^2 = 2(N^2/4) =$ $\mathbf{N^2/2}$
    - If 4 problem were being created: $4 = b^d$ so $4(N^2/4) =$ **N^2 at each level!**

- $a = 2, b=2 , d=2$
  - $a = 2 < b^d = 2^2 = 4$
- Work in total is dominated by root!
  - $O(N^2)$

Master Method w/ Case 3

- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!

- Assume $a > b^d$
- $T(N) = 4T(N/2) + O(N)$
  - More work is being done at each level
    - Level 2 is now $4(N/2) =$ **2N**
      - Each level now has more work!
- a = 4 b=2 , d=1
  - $a = 4 > b^d = 2^1 = 2$
- Work down at the leaves is overwhelming all other levels.
- How many leaves?
  - Next Slide

Master Method w/ Case 3

- Assume $a > b^d$
- $T(N) = 4T(N/2) + O(n)$
  - More work is being done at each level
- $a = 4$ $b=2$, $d=1$
  - $a = 4 > b^d = 2^1 = 2$
- How many leaves
  - Each node has 4 children
  - 4^{depth} number of leaves
- Depth = $\log_b N$
  - $4^{\log_2 N} = N^{\log_2 4}$

Master Method w/ Case 3

- Depth = $\log_b N$
  - $4^{\log_2 N} = N^{\log_2 4}$

$$4^{\log_2 N} = N\log{}_2{}^4$$

$$\log_2(4^{\log_2 N}) = \log_2(N^{\log_2 4})$$

$$\log_2 N \log_2 4 = \log_2 4 \log_2 N$$

- $4^{\log_2 N} = N^{\log_2 4} = N^2$

Solving Recurrences w/ Master Method

If $T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$

then

$$T(n) = \begin{cases} O\left(n^d \log n\right) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Solving Recurrences w/ Master Method

If $T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$

then

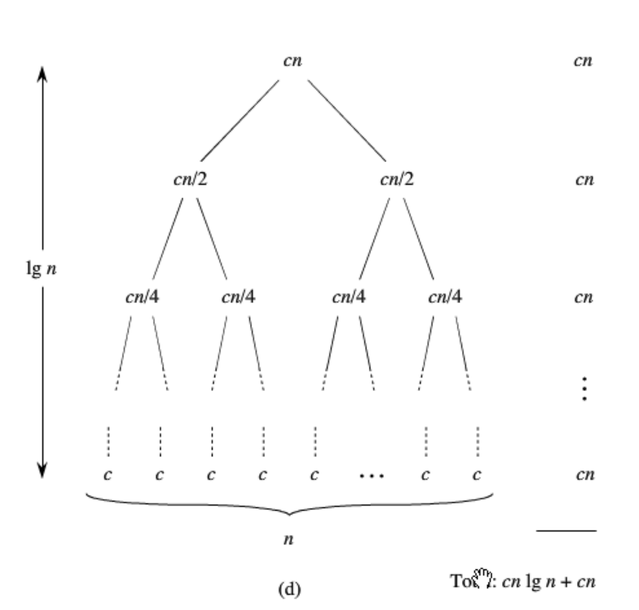
$$T(n) = \begin{cases} O\left(n^d \log n\right) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Solving Recurrences w/ Master Method

- Perhaps we can come up with some general formula for recurrences:
- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!

- 3 Cases to think about!

Master Method Case 1



- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!
- Assume $a = b^d$
  - a (or equivalently b^d) subproblems each level
  - Each subproblem has size $\frac{N}{b^d}$ and there are b^d of them for a total of N
  - n^d work to merge
- $O(N^d \log N)$
  - Balance between work at each level and depth
- Example:
  - a = 2, b=2 , d=1
  - $b^d = 2^1 = 2 = a$
  - $N\lg_2 N$

Master Method w/ Case 2

- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!
- Assume $a < b^d$
  - Number of subproblems is growing slowly
- $O(N^d)$
  - Total work dominated by the combine step at root!
  - Good (subproblem simplification) is beating evil (subproblem proliferation)!
- $T(N) = 2T(N/2) + O(n^2)$
  - Less work is being done at each level
  - Work at level 2 is $2T(N/2) = 2(N/2)^2 = 2(N^2/4) =$ **$N^2/2$**
    - If 4 problem were being created: $4 = b^d$ so $4(N^2/4) =$ **N^2 at each level!**
- $a = 2, b=2 , d=2$
  - $a = 2 < b^d = 2^2 = 4$
- Work in total is dominated by root!
  - $O(N^2)$

Master Method w/ Case 3

- $T(N) = aT(N/b) + O(N^d)$
  - N^d work down to merge subproblems!

- Assume $a > b^d$
- $T(N) = 4T(N/2) + O(N)$
  - More work is being done at each level
    - Level 2 is now $4(N/2)$ = **2N**
      - Each level now has more work!
- a = 4 b=2 , d=1
  - $a = 4 > b^d = 2^1 = 2$
- Work down at the leaves is overwhelming all other levels.
- How many leaves?
  - Next Slide

Master Method w/ Case 3

- Assume $a > b^d$
- $T(N) = 4T(N/2) + O(n)$
  - More work is being done at each level
- $a = 4$ $b=2$, $d=1$
  - $a = 4 > b^d = 2^1 = 2$
- How many leaves
  - Each node has 4 children
  - 4^{depth} number of leaves
- Depth = $\log_b N$
  - $4^{\log_2 N} = N^{\log_2 4}$

Master Method w/ Case 3

- Depth = $\log_b N$
  - $4^{\log_2 N} = N^{\log_2 4}$

$$4^{\log_2 N} = N\log_2{}^{4}$$

$$\log_2(4^{\log_2 N}) = \log_2(N^{\log_2 4})$$

$$\log_2 N \log_2 4 = \log_2 4 \log_2 N$$

- $4^{\log_2 N} = N^{\log_2 4} = N^2$

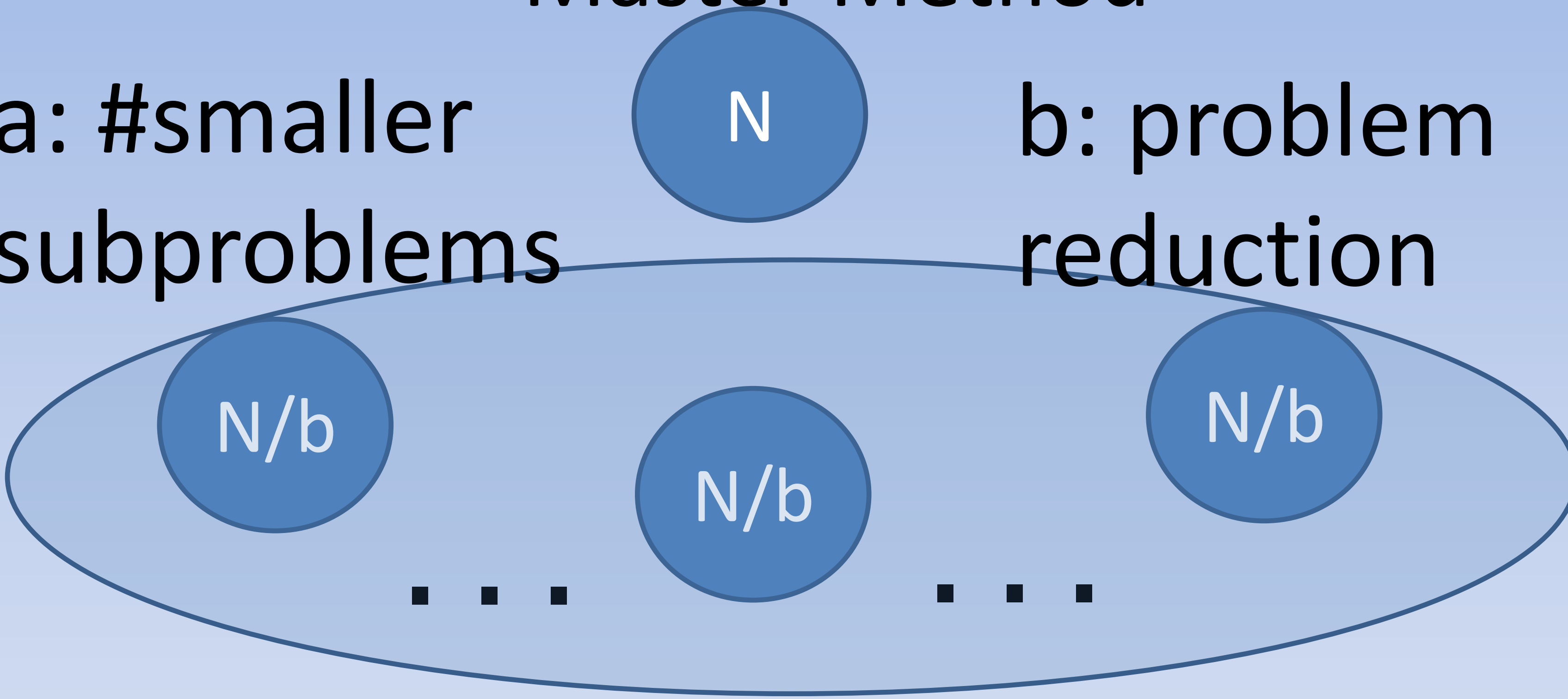
Master Method w/ Proof!

- Simplifying Assumptions
- $T(1) \le c$
- $T(N) \le aT(N/b) + cn^d$
  - and n is power of b
    - Otherwise need floor & ceiling

Solving Recurrences w/ Master Method

a: #smaller subproblems

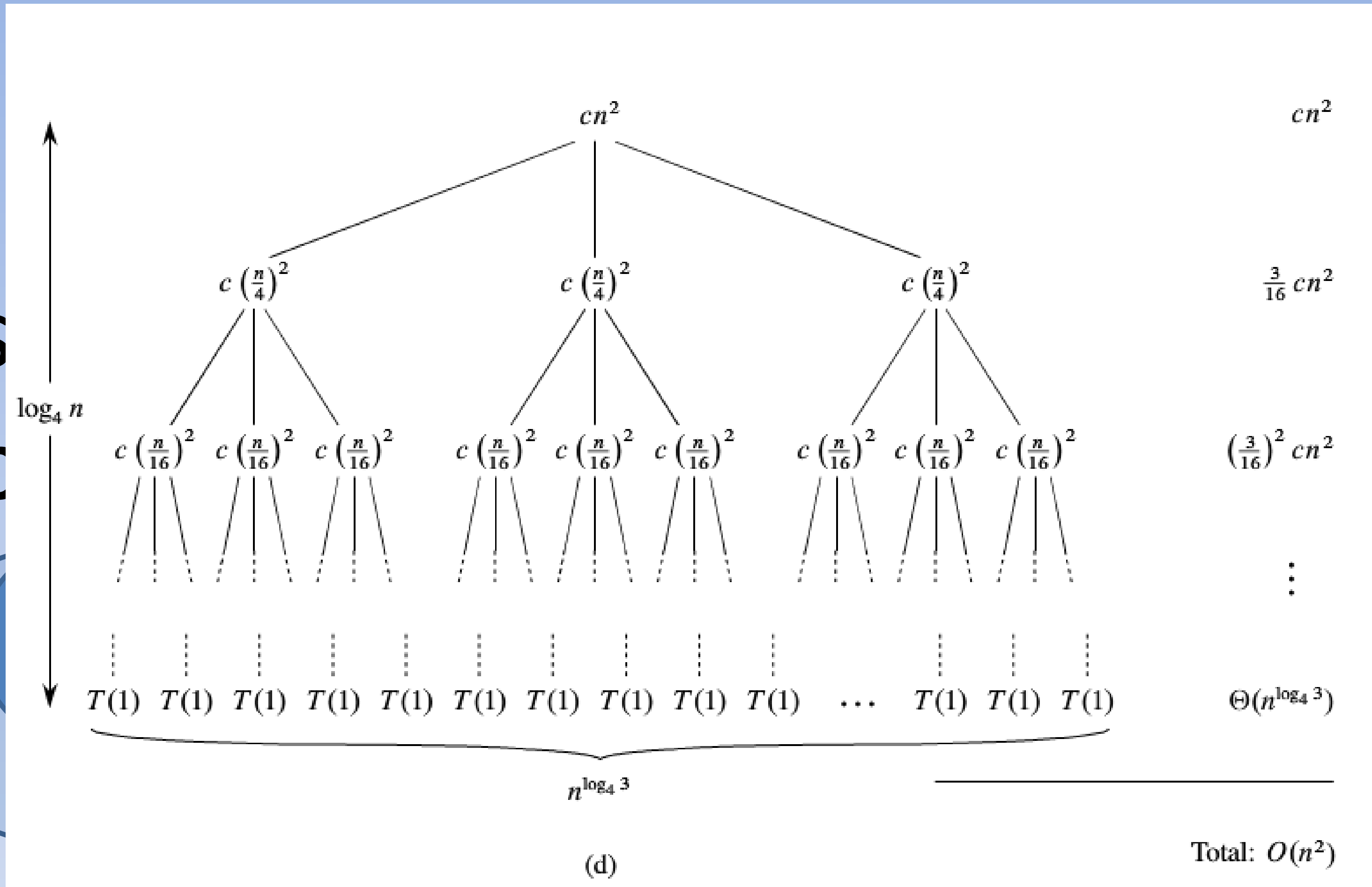
b: problem reduction



- Work($level_j$) ≤ $a^j * c(N/b^j)^d$
  - a^j: This is the number of nodes at level j
  - $c(N/b^j)^d$: Each of the nodes do this much work
    - (N^d) where N is reduced to N/b^j

a: #s

subp



(d)

- Work(level_j) ≤ $a^j * c(N/b^j)^d$
  - a^j: This is the number of nodes at level j
  - $c(N/b^j)^d$: Each of the nodes do this much work
    - (N^d) where N is reduced to N/b^j

Solving Recurrences w/ Master Method

- Work($level_j$) ≤ $a^j * c(N/b^j)^d$
  - a^j: This is the number of nodes at level j
  - $c(N/b^j)^d$: Each of the nodes do this much work
    - (N^d) where N is reduced to N/b^j

- Now sum over all the levels!
- Levels = Depth = $\log_b N$

Solving Recurrences w/ Master Method

- Work(level$_j$) ≤ a^j * $c(N/b^j)^d$
  - a^j: This is the number of nodes at level j
  - $c(N/b^j)^d$: Each of the nodes do this much work
    - (N^d) where N is reduced to N/b^j
- Levels = Depth = $\log_b N$

$$Total\ Work = cN^d \cdot \sum_{j=0}^{j=\log_b N} \left(\frac{a}{b^d}\right)^j$$

- NOTE: $a = b^d$ means $a/b^d = 1$
  - Master Method case 1: $cN^d \cdot \log_b N = O(N^d \log N)$

Solving Recurrences w/ Master Method Case 2

$$Total\ Work = cN^d \cdot \sum_{j=0}^{j=\log_b N} \left(\frac{a}{b^d}\right)^j$$

- IF: $a < b^d$ means $a/b^d < 1$
- SO: $(a/b^d)^j$ is getting smaller
- SO Asymptotically : Total work dominated by root where j=0 and $T(N) = cN^d$

Solving Recurrences w/ Master Method Case 3

$$Total\ Work = cN^d \cdot \sum_{j=0}^{j=\log_b N} \left(\frac{a}{b^d}\right)^j$$

- Finally: a > b^d means $a/b^d > 1$
- SO: $(a/b^d)^j$ is growing exponentially!
- Total work is going to be dominated asymptotically by leaves:

$$Total\ Work \le\ cN^d \cdot \left(\frac{a}{b^d}\right)^{log_b N}$$

Solving Recurrences w/ Master Method Case 3

$$Total\ Work \le cN^d \cdot \left(\frac{a}{b^d}\right)^{log_b N}$$

$$cN^d \cdot \left(\frac{a}{b^d}\right)^{log_b N} = cN^d \cdot \mathrm{a}^{\log_\mathrm{b} \mathrm{N}} \cdot b^{-\mathrm{d}\log_\mathrm{b}\mathrm{N}}$$

$$b^{-\mathrm{d}\log_\mathrm{b}\mathrm{N}} = N^{-\mathrm{d}}$$

$$cN^d \cdot \mathrm{a}^{\log_\mathrm{b} \mathrm{N}} \cdot b^{-\mathrm{d}\log_\mathrm{b}\mathrm{N}} = c\,\frac{N^d}{N^\mathrm{d}} \cdot \mathrm{a}^{\log_\mathrm{b}\mathrm{N}}$$

$$O(\mathrm{a}^{\log_\mathrm{b}\mathrm{N}})$$

Solving Recurrences w/ Master Method Case 3

$$Total\ Work =$$

$$O\left(\mathrm{a}^{\log_\mathrm{b}\mathrm{n}}\right) = O\left(n^{\log_\mathrm{b}\mathrm{a}}\right)$$

$$log_b(\mathrm{a}^{\log_\mathrm{b}\mathrm{n}}) = log_b(n^{\log_\mathrm{b}\mathrm{a}})$$

$$log_b n \cdot log_b \mathrm{a} \quad = log_b a \cdot log_b n$$

Total Work = $O\left(n^{\log_\mathrm{b}\mathrm{a}}\right)$

Textbook

The master theorem

The master method depends on the following 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 $T(n) = \Theta(f(n))$. ∎

The master theorem

The master method depends on the following 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 $T(n) = \Theta(f(n))$. ∎

- Consider $f(n) = n^d$
- Case 1: $n^d < cn^{\lg_b a}$
  - $\log_n(n^d) < \log_n(n^{\lg_b a})$
  - $d < \lg_b a$
  - $b^d < b^{\lg_b a}$
  - $b^d < a$
- $\Theta(n^{\log_b a})$
  - Case 3 from earlier!
  - Work dominated by leaves!!

The master theorem

The master method depends on the following 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 $T(n) = \Theta(f(n))$. ■

- Consider $f(n) = n^d$
- Case 3: $f(n) = \Omega(n^{\log_b a + \in})$
- $n^d > cn^{\log_b a + \in}$
  - $\log_n(n^d) > \log_n(n^{\log_b a + \in}) \geq \log_n(n^{\log_b a})$
  - $d > \lg_b a$
  - $b^d > b^{\lg_b a}$
  - $b^d > a$
- $\Theta(f(n)) = O(n^d)$

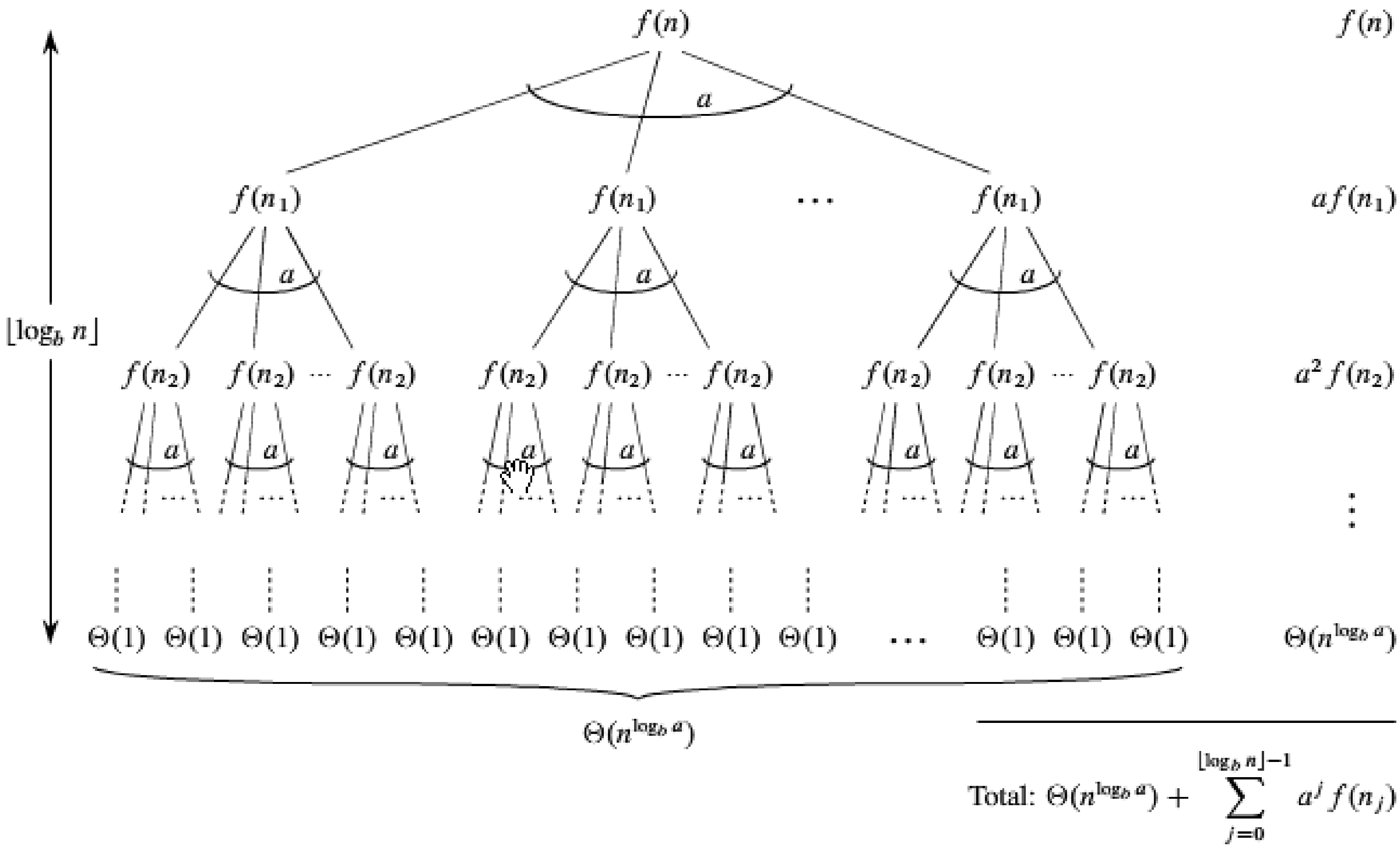


Figure 4.8 The recursion tree generated by $T(n) = aT(\lceil n/b \rceil) + f(n)$. The recursive argument n_j is given by equation (4.27).