CS 25 - Algorithms

9/25/95

Last time (ch. 2)
- Analysis of alg.
- Order notation
- Stated recurrences...

Today (ch. 4)
- Solving recurrences
  - substitution
  - iteration/recursion tree
  - "master" method

Announcements
- X-hour tomorrow
- Due-date change?

Last time

Order notation comment (neglected to mention last time...)

$T(n) = 2T(n/2) + O(g(n))$ means...

- $T(n) = 2T(n/2) + h(n)$ for some $h(n) \in O(g(n))$

or - $T(n) \le 2T(n/2) + c\,g(n)$ for some $c > 0$ and all $n \ge n_0 > 0$

E.g. $T(n) = 2T(n/2) + O(n^2)$ might mean

$$T(n) = 2T(n/2) + n^2 + 1{,}000{,}000\,n$$

which implies $T(n) \le 2T(n/2) + 2n^2$ $\forall\, n \ge 1{,}000{,}000$

also implies $T(n) \le 2T(\frac{n}{2}) + 1{,}000{,}001\,n^2$ $(\forall n)$

(over)

Recurrence review

Actual recurrence:

$$T(n) = \begin{cases} \Theta(1) & n=1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \forall n > 1 \end{cases}$$

We will...

- ignore floors / ceilings
- assume boundary conditions

and write

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

Today

3 methods for solving recurrences

1. substitution
2. iteration / recursion tree
3. "master"

Boundary conditions
- Usually assume $T(n) = \Theta(1)$ for suff. small n.
- Doesn't usually affect solution (if polynomially bounded).

Example where it does affect solution—exponential:

$$T(n) = (T(\tfrac{n}{2}))^2$$

$T(1) = 2 \Rightarrow T(n) = \Theta(2^n)$

$T(1) = 3 \Rightarrow T(n) = \Theta(3^n)$

$\vdots$

and $2^n = o(3^n)$.

3 methods for solving
1. substitution
2. iteration
3. master

* ⇒ Substitution method
- Guess a solution, • verify it, • solve for consts

Example: $T(n) = 4T(\tfrac{n}{2}) + n$.

Guess: $T(n) = O(n^3)$.

Assume: $T(k) \leq ck^3 \ \forall\ k < n$.

Verify: $T(n) \leq cn^3$ by induction

(Using strong induction—assuming true for all values $< n$, not just for $n-1$.)

$$T(n) = 4T(\tfrac{n}{2}) + n$$
$$\leq 4c(\tfrac{n}{2})^3 + n$$
$$= \tfrac{c}{2}n^3 + n$$
$$\Rightarrow = cn^3 - (\tfrac{c}{2}n^3 - n)$$
$$\leq cn^3 \text{ if } \tfrac{c}{2}n^3 - n \geq 0$$
$$\Leftrightarrow \tfrac{c}{2}n^2 \geq 1$$
$$\Leftrightarrow n^2 \geq \tfrac{2}{c}$$

Use $c \geq 2$ and $n \geq 1$. $\Leftarrow$ $n \geq 2/c$

GOT THIS FAR

Good approach: to prove inductive step, write exper. in form

<answer you want> − <positive quantity>

* Initial conditions - pick c big enough to handle.

This bound isn't tight.
Can show $O(n^2)$.

* Fallacious argument

Assume $T(k) \leq ck^2$.

$$T(n) = 4T(\tfrac{n}{2}) + n$$
$$\leq 4c(\tfrac{n}{2})^2 + n$$
$$= cn^2 + n \quad \{\text{WRONG! Must use same}$$
$$= O(n^2) \quad \{\text{const as in inductive assertion}$$
$$\leq cn^2 \text{ for } \underline{\text{no choice}} \text{ of } c > 0. \text{ Lose}$$

Correct idea
Subtract off a lower-order term.
Assume $T(k) \le c_1 k^2 - c_2 k$

$$\begin{aligned} T(n) &= 4T\left(\tfrac{n}{2}\right) + n \\ &\le 4\left[c_1\left(\tfrac{n}{2}\right)^2 - c_2\tfrac{n}{2}\right] + n \\ &= c_1 n^2 - 2c_2 n + n \\ &= c_1 n^2 - c_2 n - (c_2 n - n) \\ &\le c_1 n^2 - c_2 n \end{aligned}$$

; if $c_2 n - n \ge 0 \iff c_2 \ge 1$.
Pick c_1 big enough to handle initial cond

Why subtract lower-order term?
If math doesn't work out, should we increase our guess?
Not necessarily – doing induction to prove something stronger for a given value for assuming something stronger for smaller values.

* Can use substitution method for lower (Ω) bounds, too.

* · Can be tricky to "guess" solution.
· Usually, use another method to give idea of solution and "substitution" to prove it.

Iteration

Expand out the recurrence to generate a good guess.

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

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

$= n + 4(\frac{n}{2} + 4(\frac{n}{4} + 4T(\frac{n}{8})))$

$\vdots$

$= n + 2n + 4n + \cdots + 4^{\lg n} T(1)$

($4^{\lg n}$ because after $\lg n$ iterations, $2^{\lg n} = (2^{\lg n})^2 = n^2$)

$n + 2n + 4n + 8n + \cdots + 2^{k-1}n + 4^k T(\frac{n}{2^k})$

$= n(2^k - 1) + 4^k T(\frac{n}{2^k})$

$= n + 2n + 16T(\frac{n}{4})$

$= n + 2n + 4n + 4^3 T(\frac{n}{8})$

$= n + \cdots + 2$

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

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

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

$k^{th} \quad = n \sum_{i=0}^{k-1} 2^i + 4^k T(n/2^k)$ $\qquad \frac{n}{2^k} = 1 \iff k = \lg n$

$\lg n \quad = n \sum_{i=0}^{\lg n - 1} 2^i + 4^{\lg n} T(1)$

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

$= n(n-1) + n^2 T(1) = \Theta(n^2)$

~~for $x \neq 1$~~

$= n(\quad) + \Theta(n)$

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

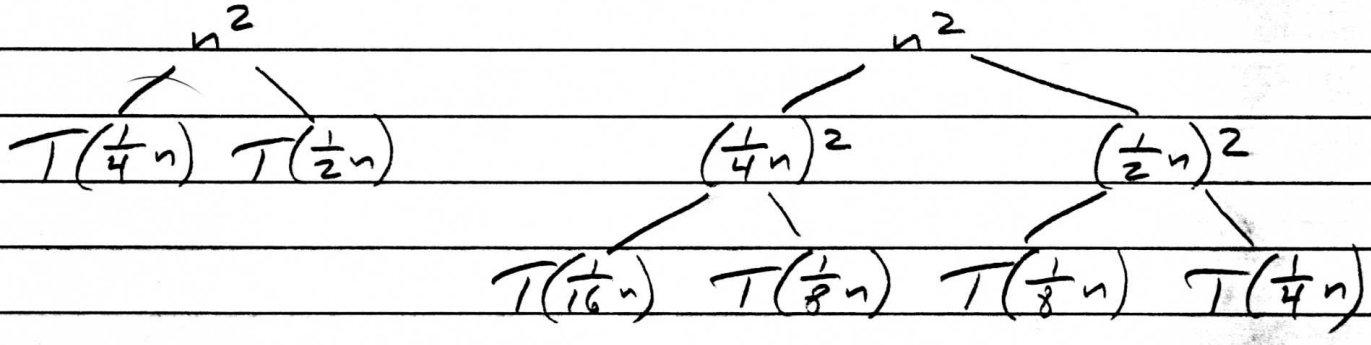
$= \Theta(n^2)$

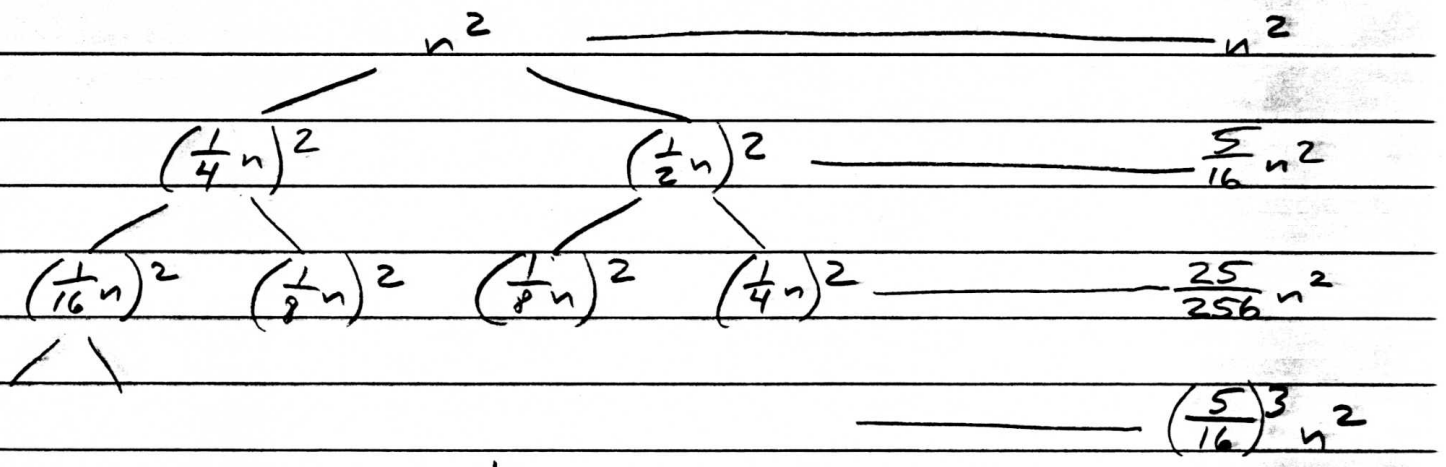
- Know your summations & rules for manipulating logs & exponents.
- Math can be messy & hard. Use this method for generating a guess for subst. method.

Recursion tree

Visualize iterating a recurrence.

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





geometric, fraction < 1

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

$$\sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k = \frac{1}{1-\frac{5}{16}} = \frac{16}{11}$$

$\therefore \Theta(n^2)$

#(leaves) $\propto$ # internal nodes

- can ignore leaves

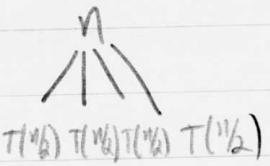
leaves: true depth $\le \log_2 n$

$\Rightarrow \le 2^{\log_2 n} = n$ leaves

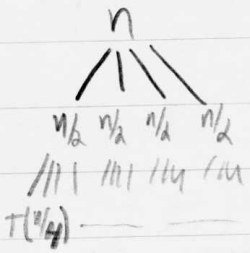
$n \cdot T(1) = n \cdot \Theta(1) = \Theta(n)$

More examples

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



$\Rightarrow$



n

$4 \cdot n/2 = 2n$

$4^2 \cdot n/2^2 = 4n$

$4^3 \cdot n/2^3 = 8n$

$\lg n - 1$

$4^{\lg n} \cdot T(1)$

$n + 2n + 4n + \cdots 2^{\lg n - 1} n + 4^{\lg n} T(1)$

just like before...

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

⋮

CS-25 Algorithms 9/27/85

Last time (chap 2,3,4)
- Solving recurrences
  - substitution
  - iteration / recursion tree
  - begun "master" method
- x-hour - math primer

Today (chap 4.3, 10)
- "Master" method in detail
- Median & Order statistics

Announcements
- New office hours

"Master" method

- general technique for solving many recurrences of the form $T(n) = aT(n/b) + f(n)$

Let's consider $\boxed{T(n) = aT(n/b) + \Theta(n^c)}$ (will generalize to $f(n)$ later)

e.g. mergesort $T(n) = 2T(n/2) + \Theta(n)$

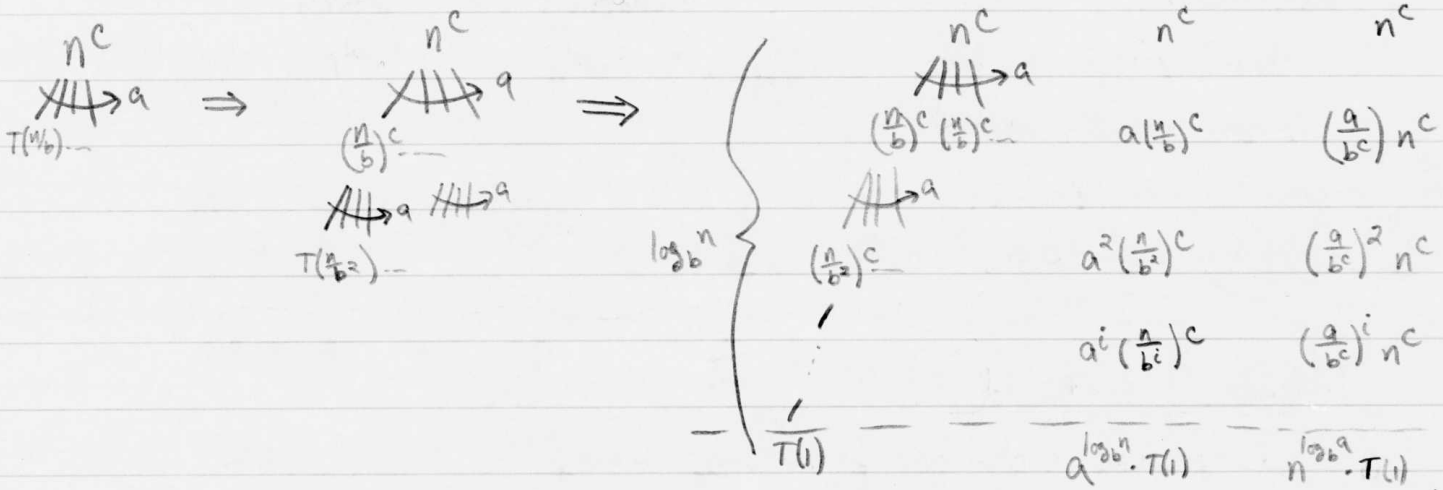
Strassen's $T(n) = 7T(n/2) + \Theta(n^2)$

binary search $T(n) = T(n/2) + \Theta(1)$

Slightly different treatment than in book...

$$T(n) = aT(n/b) + n^c \qquad \text{(for simplicity, eliminate } \Theta\text{)}$$

Recursion tree:



So, total is $n^c \sum_{i=0}^{\log_b n - 1} (\frac{a}{b^c})^i + \Theta(n^{\log_b a})$

To solve sum, we must note whether $\frac{a}{b^c} > 1$ | $\frac{a}{b^c} = 1$ | $\frac{a}{b^c} < 1$

Case 1 | Case 2 | Case 3

Equivalently, compare c with $\log_b a$ $\quad c < \log_b a$ | $c = \log_b a$ | $c > \log_b a$

(note: either c or $\log_b a$ will be exponent in answer...)

Case 2 $\quad c = \log_b a \iff a/b^c = 1$ - work constant at each level

$$\text{sum} = n^c \sum_{i=0}^{\log_b n - 1} (1)^i + \Theta(n^{\log_b a}) = n^c \log_b n + \Theta(n^{\log_b a}) = \Theta(n^c \log_b n)$$

$\therefore$ work at each level is n^c $(= n^{\log_b a})$; $\log_b n$ levels;

answer is $\Theta(n^c \log_b n)$

Case 1 $c < \log_b a \iff \frac{a}{b^c} > 1$ – work increases geometrically

$$\text{sum} = n^c \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i + \Theta\left(n^{\log_b a}\right)$$

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

$$= \Theta\left(n^c \frac{a^{\log_b n}}{(b^c)^{\log_b n}}\right) + \Theta\left(n^{\log_b a}\right)$$

$(b^c)^{\log_b n}$ = $b^{c \cdot \log_b n}$ = $(b^{\log_b n})^c$ = n^c

$$= \Theta\left(n^c \frac{n^{\log_b a}}{n^c}\right) + \Theta\left(n^{\log_b a}\right)$$

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

∴ work at each level increases geometrically; constant fraction of work is in leaves...

Case 3 $c > \log_b a \iff \frac{a}{b^c} < 1$ - work decreases geometrically

$$\text{sum} = n^c \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i + \Theta\left(n^{\log_b a}\right)$$

$$= n^c \, \Theta(1) + \Theta\left(n^{\log_b a}\right)$$

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

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

Note:

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

$$\Rightarrow \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i = \Omega(1)$$

② $\sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i < \sum_{i=0}^{\infty} \left(\frac{a}{b^c}\right)^i$

$$= \frac{1}{1 - a/b^c} \quad \text{(constant)}$$

$$\Rightarrow \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i = O(1)$$

$$\therefore \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i = \Theta(1)$$

$\therefore$ work at each level decreases geometrically;
constant fraction of work is at root...

Idea: compare c to $\log_b a$

Case 1: $c < \log_b a \Rightarrow \Theta\left(n^{\log_b a}\right)$

Case 2: $c = \log_b a \Rightarrow \Theta\left(n^c \log_b n\right) = \Theta\left(n^{\log_b a} \log_b n\right)$

Case 3: $c > \log_b a \Rightarrow \Theta(n^c)$

Examples

(1) Mergesort $T(n) = 2T(n/2) + \Theta(n)$

$\log_2 2 = 1$ vs. 1

Case 2 $\Rightarrow$ $T(n) = \Theta(n \lg n)$

(2) Strassen's $T(n) = 7T(n/2) + \Theta(n^2)$

$\log_2 7 \approx 2.81$ vs. 2

Case 1 $\Rightarrow$ $T(n) = \Theta(n^{\log_2 7})$

(3) Binary Search $T(n) = T(n/2) + \Theta(1)$

$\log_2 1 = 0$ vs. 0

Case 2 $\Rightarrow$ $T(n) = \Theta(n^0 \lg n) = \Theta(\lg n)$

(4) How about... $T(n) = 4T(n/2) + \Theta(n^3)$

$\log_2 4 = 2$ vs. 3

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

More generally (see text):

$$T(n) = a\,T(n/b) + f(n) \quad \leftarrow \text{not necessarily a simple polynomial}$$

e.g. $T(n) = 4T(n/2) + \Theta(n^2 \log n)$

Case 1: $\frac{f(n)}{n^{\log_b a}} \leq O(n^{-\varepsilon})$ for some $\varepsilon > 0$ | $f(n) = O(n^{\log_b a - \varepsilon})$

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

Case 2: $\frac{f(n)}{n^{\log_b a}} \leq \Theta(\log^k n)$ for some $k \geq 0$ | $f(n) = \Theta(n^{\log_b a} \log^k n)$

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

Case 3: $\frac{f(n)}{n^{\log_b a}} \geq \Omega(n^{\varepsilon})$ for some $\varepsilon > 0$ | $f(n) = \Omega(n^{\log_b a + \varepsilon})$

(regularity condition) Then, as long as $a\,f(n/b) \leq \alpha\, f(n)$ for some const. $\alpha < 1$,

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

Examples

(1) $T(n) = 4T(n/2) + \Theta(n^2 \log n)$

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^2 \log n}{n^{\log_2 4}} = \log n$$

Case 2 : $T(n) = \Theta(n^2 \log^2 n)$

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

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n^{\log_2 4}} = \frac{n^3}{n^2} = n$$

Case 3 : Check regularity condition

$4(n/2)^3 \leq \alpha n^3$ for some $\alpha < 1$?

$4 \frac{n^3}{8} \leq \alpha n^3$

$\alpha \geq 1/2$ ✓

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

(3) $T(n) = 4T(n/2) + \frac{n^2}{\lg n}$

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^2/\lg n}{n^{\log_2 4}} = \frac{1}{\lg n}$$

Case 1 — no $\quad \frac{1}{\lg n} = O(n^{-\varepsilon}) \iff \lg n = \Omega(n^{\varepsilon})$ ✗

Case 2 — no $\quad \frac{1}{\lg n} = \Theta(\log^k n)$ for $k \geq 0$ ✗

Case 3 — no $\quad \frac{1}{\lg n} = \Omega(n^{\varepsilon})$ ✗

No case applies ... can show $T(n) = \Theta(n^2 \log\log n)$ by careful iteration.

Solving Recurrences via Iteration

Consider the recurrence $T(n) = 4T(n/2) + n^2/\lg n$. In order to solve the recurrence, I would first suggest rewriting the recurrence with the recursive component *last* and using a generic parameter not to be confused with n. We may think of the following equation as our general pattern, which holds for any value of $\square$.

$$T(\square) = \frac{\square^2}{\lg \square} + 4T(\square/2) \tag{1}$$

Since our pattern (Equation 1) is valid for any value of $\square$, we may use it to "iterate" the recurrence as follows.

$$\begin{aligned} T(n) &= \frac{n^2}{\lg n} + 4T(n/2) \\ &= \frac{n^2}{\lg n} + 4\left(\frac{(n/2)^2}{\lg(n/2)} + 4T(n/2^2)\right) \\ &= \frac{n^2}{\lg n} + \frac{n^2}{\lg(n/2)} + 4^2T(n/2^2) \end{aligned} \tag{2}$$

Always simplify the expression, eliminating parentheses as in Equation 2, before expanding further. Continuing...

$$\begin{aligned} T(n) &= \frac{n^2}{\lg n} + \frac{n^2}{\lg(n/2)} + 4^2\left(\frac{(n/2^2)^2}{\lg(n/2^2)} + 4T(n/2^3)\right) \\ &= \frac{n^2}{\lg n} + \frac{n^2}{\lg(n/2)} + \frac{n^2}{\log(n/2^2)} + 4^3T(n/2^3) \\ &\vdots \\ &= \frac{n^2}{\lg n} + \frac{n^2}{\lg(n/2)} + \frac{n^2}{\lg(n/2^2)} + \ldots + \frac{n^2}{\lg(n/2^{k-1})} + 4^kT(n/2^k) \\ &= \sum_{j=0}^{k-1} \frac{n^2}{\lg(n/2^j)} + 4^kT(n/2^k) \end{aligned}$$

We will next show that the pattern we have established is correct, by induction.

Claim 1 *For all $k \geq 1$, $T(n) = \sum_{j=0}^{k-1} \frac{n^2}{\lg(n/2^j)} + 4^kT(n/2^k)$.*

Proof: The proof is by induction on k. The base case, $k = 1$, is trivially true since the resulting equation matches the original recurrence. For the inductive step, assume that the statement is true for $k = i - 1$; i.e.,

$$T(n) = \sum_{j=0}^{i-2} \frac{n^2}{\lg(n/2^j)} + 4^{i-1}T(n/2^{i-1}).$$

Our task is then to show that the statement is true for $k = i$; i.e.,

$$T(n) = \sum_{j=0}^{i-1} \frac{n^2}{\lg(n/2^j)} + 4^i T(n/2^i).$$

This may be accomplished by starting with the inductive hypothesis and applying the definition of the recurrence, as follows.

$$\begin{aligned}
T(n) &= \sum_{j=0}^{i-2} \frac{n^2}{\lg(n/2^j)} + 4^{i-1} T(n/2^{i-1}) \\
&= \sum_{j=0}^{i-2} \frac{n^2}{\lg(n/2^j)} + 4^{i-1} \left[\frac{(n/2^{i-1})^2}{\lg(n/2^{i-1})} + 4T(n/2^i) \right] \\
&= \sum_{j=0}^{i-2} \frac{n^2}{\lg(n/2^j)} + 4^{i-1} \frac{n^2/4^{i-1}}{\lg(n/2^{i-1})} + 4^i T(n/2^i) \\
&= \sum_{j=0}^{i-2} \frac{n^2}{\lg(n/2^j)} + \frac{n^2}{\lg(n/2^{i-1})} + 4^i T(n/2^i) \\
&= \sum_{j=0}^{i-1} \frac{n^2}{\lg(n/2^j)} + 4^i T(n/2^i)
\end{aligned}$$

□

We thus have that $T(n) = \sum_{j=0}^{k-1} \frac{n^2}{\lg(n/2^j)} + 4^k T(n/2^k)$ for all $k \geq 1$. We next choose a value of k which causes our recurrence to reach a known base case. Since $n/2^k = 1$ when $k = \lg n$, and $T(1) = \Theta(1)$, we have

$$\begin{aligned}
T(n) &= \sum_{j=0}^{\lg n - 1} \frac{n^2}{\lg(n/2^j)} + 4^{\lg n} T(1) \\
&= n^2 \sum_{j=0}^{\lg n - 1} \frac{1}{\lg n - j} + n^{\lg 4} \Theta(1) \\
&= n^2 \sum_{\ell=1}^{\lg n} \frac{1}{\ell} + \Theta(n^2) \\
&= n^2 \, \Theta(\ln \lg n) + \Theta(n^2) \\
&= \Theta(n^2 \log \log n).
\end{aligned}$$

The Simplified Master Method for Solving Recurrences

Consider recurrences of the form

$$T(n) = aT(n/b) + n^c$$

for constants $a \geq 1$, $b > 1$, and $c \geq 0$. Recurrences of this form include *mergesort*,

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

Strassen's algorithm for matrix multiplication,

$$T(n) = 7T(n/2) + n^2,$$

binary search,

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

and so on.

We can solve the general form of this recurrence via iteration. Rewriting the recurrence with the recursive component *last* and using a generic parameter not to be confused with n, we obtain:

$$T(\square) = \square^c + aT(\square/b) \tag{1}$$

Since our pattern (Equation 1) is valid for any value of $\square$, we may use it to "iterate" the recurrence as follows.

$$\begin{aligned}
T(n) &= n^c + aT(n/b) \\
&= n^c + a\left[(n/b)^c + a\,T(n/b^2)\right] \\
&= n^c + a(n/b)^c + a^2\,T(n/b^2) \\
&= n^c + a(n/b)^c + a^2\left[(n/b^2)^c + a\,T(n/b^3)\right] \\
&= n^c + a(n/b)^c + a^2(n/b^2)^c + a^3\,T(n/b^3) \\
&= n^c + a(n/b)^c + a^2(n/b^2)^c + a^3\left[(n/b^3)^c + a\,T(n/b^4)\right] \\
&= n^c + a(n/b)^c + a^2(n/b^2)^c + a^3(n/b^3)^c + a^4\,T(n/b^4)
\end{aligned}$$

Pulling out the n^c term common in each of the first four factors, we may simplify this expression and obtain a pattern as follows.

$$\begin{aligned}
T(n) &= n^c + n^c(a/b^c) + n^c(a/b^c)^2 + n^c(a/b^c)^3 + a^4\,T(n/b^4) \\
&\vdots \\
&= n^c \sum_{j=0}^{k-1} \left(\frac{a}{b^c}\right)^j + a^k\,T(n/b^k)
\end{aligned}$$

We will next show that the pattern we have established is correct, by induction.

Claim 1 *For all* $k \geq 1$, $T(n) = n^c \sum_{j=0}^{k-1} \left(\frac{a}{b^c}\right)^j + a^k\,T(n/b^k)$.

Proof: The proof is by induction on k. The base case, $k = 1$, is trivially true since the resulting equation matches the original recurrence. For the inductive step, assume that the statement is true for $k = i - 1$; i.e.,

$$T(n) = n^c \sum_{j=0}^{i-2} \left(\frac{a}{b^c}\right)^j + a^{i-1}\, T(n/b^{i-1}).$$

Our task is then to show that the statement is true for $k = i$; i.e.,

$$T(n) = n^c \sum_{j=0}^{i-1} \left(\frac{a}{b^c}\right)^j + a^i\, T(n/b^i).$$

This may be accomplished by starting with the inductive hypothesis and applying the definition of the recurrence, as follows.

$$\begin{aligned}
T(n) &= n^c \sum_{j=0}^{i-2} \left(\frac{a}{b^c}\right)^j + a^{i-1}\, T(n/b^{i-1}) \\
&= n^c \sum_{j=0}^{i-2} \left(\frac{a}{b^c}\right)^j + a^{i-1}\, \left[(n/b^{i-1})^c + a\, T(n/b^i)\right] \\
&= n^c \sum_{j=0}^{i-2} \left(\frac{a}{b^c}\right)^j + n^c \left(\frac{a}{b^c}\right)^{i-1} + a^i\, T(n/b^i) \\
&= n^c \sum_{j=0}^{i-1} \left(\frac{a}{b^c}\right)^j + a^i\, T(n/b^i)
\end{aligned}$$

□

We thus have that $T(n) = n^c \sum_{j=0}^{k-1} \left(\frac{a}{b^c}\right)^j + a^k\, T(n/b^k)$ for all $k \geq 1$. We next choose a value of k which causes our recurrence to reach a known base case. Since $n/b^k = 1$ when $k = \log_b n$, and $T(1) = \Theta(1)$, we have

$$\begin{aligned}
T(n) &= n^c \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^j + a^{\log_b n}\, T(1) \\
&= n^c \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^j + n^{\log_b a}\, \Theta(1) \\
&= n^c \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^j + \Theta(n^{\log_b a})
\end{aligned}$$

The solution to our recurrence involves the geometric series $\sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^j$. In order to bound this series, we must consider three cases: $a/b^c > 1$, $a/b^c = 1$, and $a/b^c < 1$. This is equivalent to considering the cases $c < \log_b a$, $c = \log_b a$, and $c > \log_b a$.

Case 1: $c < \log_b a \Leftrightarrow a/b^c > 1$.

If $a/b^c > 1$, we then have

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

From this we may concude that

$$\begin{aligned} T(n) &= n^c \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^j + \Theta(n^{\log_b a}) \\ &= n^c \cdot \Theta\left(\frac{n^{\log_b a}}{n^c}\right) + \Theta(n^{\log_b a}) \\ &= \Theta(n^{\log_b a}). \end{aligned}$$

Case 2: $c = \log_b a \Leftrightarrow a/b^c = 1$.

If $a/b^c = 1$, we then have

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

Noting that $c = \log_b a$, we may then conclude that

$$\begin{aligned} T(n) &= n^c \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^j + \Theta(n^{\log_b a}) \\ &= n^c \cdot \log_b n + \Theta(n^{\log_b a}) \\ &= \Theta(n^c \log n) = \Theta(n^{\log_b a} \log n) \end{aligned}$$

Case 3: $c > \log_b a \Leftrightarrow a/b^c < 1$.

If $a/b^c < 1$, we then have

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

and

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

which yields

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

Noting that $c > \log_b a$, we may then conclude that

$$\begin{aligned} T(n) &= n^c \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^j + \Theta(n^{\log_b a}) \\ &= n^c \cdot \Theta(1) + \Theta(n^{\log_b a}) \\ &= \Theta(n^c) \end{aligned}$$

Note that in each case, either c or $\log_b a$ appears in the exponent of the solution, and it is the *larger* of these two values which appears. If these terms are equal, then an extra log factor appears as well. In summary, we have

Case 1:	$c < \log_b a$	$T(n) = \Theta(n^{\log_b a})$
Case 2:	$c = \log_b a$	$T(n) = \Theta(n^c \log n) = \Theta(n^{\log_b a} \log n)$
Case 3:	$c > \log_b a$	$T(n) = \Theta(n^c)$