Chap 4 – Divide-and-Conquer

4.1 The maximum-subarray problem
4.2 Strassen's algorithm for matrix multiplication
4.3 The substitution method … recurrences
4.4 The recursion-tree method … recurrences
4.5 The master method … recurrences
4.6 Proof of the master theorem

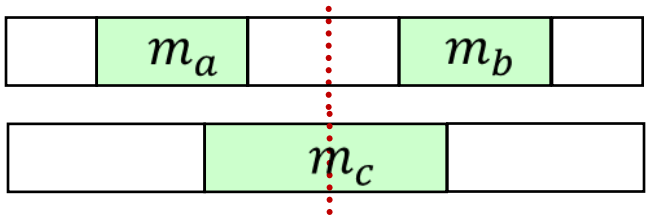
4.1 The maximum-subarray problem

- The maximum-subarray problem

  Input: An array $A[1..n]$ of numbers

  Output: Indices i and j such that $A[i..j]$ has the greatest sum of any nonempty, contiguous subarray of A, along with the sum of the values in $A[i..j]$.

- A divide-and-conquer solution



  Let

  m_a = the maximum sum of the left-half subarray

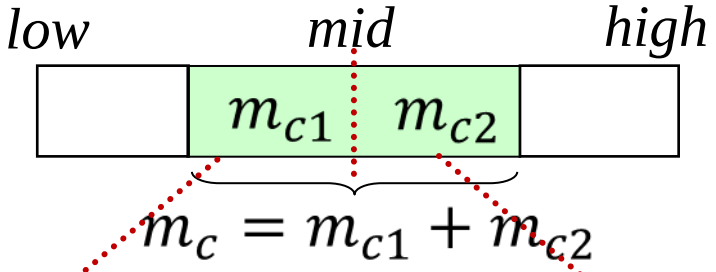
  m_b = the maximum sum of the right-half subarray

  m_c = the maximum sum crossing the border

  Then, $\max(m_a, m_b, m_c)$ is maximum sum of the entire array.

4.1 The maximum-subarray problem

- Finding m_c



Time: $\Theta(n)$

```
left-sum = −∞                          right-sum = −∞
sum = 0                                sum = 0
for i = mid downto low                 for j = mid + 1 to high
    sum = sum + A[i]                       sum = sum + A[j]
    if sum > left-sum                      if sum > right-sum
        left-sum = sum                         right-sum = sum
        max-left = i                           max-right = j
return (max-left, max-right, left-sum + right-sum)
```

4.1 The maximum-subarray problem

- Time-complexity of the algorithm

  Let $T(n)$ = the running time (for every case) of the divide-and-conquer algorithm on an array of n elements

  Then,

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

  Thus,

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

4.2 Strassen's algorithm for matrix multiplication

- Matrix multiplication

  Input: Two $n \times n$ square matrices $A = (a_{ij})$ and $B = (b_{ij})$

  Output: $C = A \cdot B = (c_{ij})$, where $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

- Traditional $\Theta(n^3)$ algorithm

  SQUARE-MAT-MULT(A, B, n)

  let C be a new $n \times n$ matrix

  for $i = 1$ **to** n

  for $j = 1$ **to** n

  $c_{ij} = 0$

  for $k = 1$ **to** n

  $c_{ij} = c_{ij} + a_{ik} b_{kj}$

  return C

4.2 Strassen's algorithm for matrix multiplication

- Simple divide-and-conquer method

  Partition each of A, B, C into four $n/2 \times n/2$ matrices

  Rewrite $C = A \cdot B$ as

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

  giving four equations

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

  8 multiplications

  4 additions

4.2 Strassen's algorithm for matrix multiplication

- Simple divide-and-conquer method

  Let $T(n) =$ the time to multiply two $n \times n$ matrices

  Then,

  $$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 8T(n/2) + \Theta(n^2) & n > 1 \end{cases}$$

  As we shall see, $T(n) = \Theta\left(n^{\lg 8}\right) = \Theta(n^3)$

- Strassen's algorithm

  $$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 7T(n/2) + \Theta(n^2) & n > 1 \end{cases}$$

  As we shall see, $T(n) = \Theta\left(n^{\lg 7}\right)$

4.2 Strassen's algorithm for matrix multiplication

- Strassen's algorithm

  Step 1: Partition each of A, B, C into four $n/2 \times n/2$ matrices

  Step 2: Create 10 matrices

$$S_1 = B_{12} - B_{22} \qquad S_6 = B_{11} + B_{22}$$
$$S_2 = A_{11} + A_{12} \qquad S_7 = A_{12} - A_{22}$$
$$S_3 = A_{21} + A_{22} \qquad S_8 = B_{21} + B_{22}$$
$$S_4 = B_{21} - B_{11} \qquad S_9 = A_{11} - A_{21}$$
$$S_5 = A_{11} + A_{22} \qquad S_{10} = B_{11} + B_{12}$$

  Add and subtract $n/2 \times n/2$matrices 10 times

  Time: $\Theta(n^2)$

4.2 Strassen's algorithm for matrix multiplication

- Strassen's algorithm

  Step 3: Create 7 matrices

  $P_1 = A_{11} \cdot S_1$ $\quad$ $P_5 = S_5 \cdot S_6$

  $P_2 = S_2 \cdot B_{22}$ $\quad$ $P_6 = S_7 \cdot S_8$

  $P_3 = S_3 \cdot B_{11}$ $\quad$ $P_7 = S_9 \cdot S_{10}$

  $P_4 = A_{22} \cdot S_4$

  Step 4: Create the matrix C

  $C_{11} = P_5 + P_4 - P_2 + P_6$

  $C_{12} = P_1 + P_2$ $\quad$ 8 additions/subtractions

  $C_{21} = P_3 + P_4$ $\quad$ Time: $\Theta(n^2)$

  $C_{22} = P_5 + P_1 - P_3 - P_7$

4.3 The substitution method

- Substitution method (Constructive induction)
  - Guess the solution
  - Use induction to find constants and show the solution works
- Example

  $T(0) = 0$

  $T(n) = n + T(n-1), \text{ if } n > 0$

  We may prove by induction that

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

  or, guess that

  $T(n) = an^2 + bn + c$

  and prove it by constructive induction to find out a, b, c

4.3 The substitution method

- Example (Cont'd)

  Inductive step

  $$T(n) = n + T(n-1)$$
  $$= n + a(n-1)^2 + b(n-1) + c$$
  $$= an^2 + (1 + b - 2a)n + a - b + c$$
  $$= an^2 + bn + c$$

  provided that

  $$1 + b - 2a = b$$

  $$a - b + c = c \Rightarrow a = b = 1/2\text{, for any } c$$

  Basis

  $T(0) = c = 0$ ·····In conclusion: $a = b = 1/2, c = 0$

4.3 The substitution method

- Example (Cont'd)

  Guess

  $T(n) = O(n^2)$

  Have to find out c and n_0 such that $T(n) \leq cn^2 \; \forall n \geq n_0$

  Inductive step

  $$T(n) = n + T(n-1)$$
  $$\leq n + c(n-1)^2$$
  $$= cn^2 - ((2c-1)n - c) \leq cn^2$$

  provided that

  $(2c-1)n - c \geq 0 \Rightarrow c > 1/2, n \geq \lceil c/(2c-1) \rceil$

  If $c \geq 1$, then $n \geq \lceil c/(2c-1) \rceil = 1$; so, choose $c \geq 1, n_0 = 1$

4.3 The substitution method

- Example (Cont'd)

  Basis: $n = 1$

  $T(1) = 1 + T(0) = 1 \leq c1^2 \Rightarrow c \geq 1$

  In conclusion, we may pick any $c \geq 1$ and $n_0 = 1$

  N.B. base case of recurrence $\neq$ base case of induction

  Alternatively, in the induction step, choose $c \geq 1, n_0 = 500$.

  Basis: $n = 500$

  $T(500) = d \quad$ for some constant d

  $\leq c \cdot 500^2 \Rightarrow c \geq \frac{d}{500^2}$

  So, pick any $c \geq \max\left\{1, \frac{d}{500^2}\right\}$

4.3 The substitution method

- Remarks

  The preceding example illustrates two points.

  Since we are interested in asymptotic solutions to recurrences

  1 The boundary cases of the recurrences are immaterial.

    ∵ $T(n)$ is always constant for any constant n

  2 The base cases of the inductive proof needn't be proven.

    ∵ It is always possible to choose base cases that work.

    For O, simply pick c large enough to handle the base case

    For Ω, simply pick c small enough to handle the base case

    (as shown in the next example).

4.3 The substitution method

- Example (Cont'd)

  Guess

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

  Have to find out c and n_0 such that $T(n) \geq cn^2 \; \forall n \geq n_0$

  Inductive step

  $$T(n) = n + T(n-1)$$
  $$\geq n + c(n-1)^2$$
  $$= cn^2 + (1-2c)n + c \geq cn^2$$

  provided that

  $(1-2c)n + c \geq 0 \Rightarrow c \leq 1/2, n \geq 0$

  So, choose $c \leq 1/2, n_0 = 0$

4.3 The substitution method

- Example (Cont'd)

  Basis: $n = 0$

  $T(0) = 0 \geq c0^2$ for any c

  In conclusion, we may pick any $c \leq 1/2$ and $n_0 = 0$.

  Alternatively, in the induction step, choose $c \leq 1/2$, $n_0 = 500$

  Basis: $n = 500$

  $T(500) = d$ for some constant d

  $$\geq c \cdot 500^2 \Rightarrow c \leq \frac{d}{500^2}$$

  So, pick any $c \leq \min\left\{\frac{1}{2}, \frac{d}{500^2}\right\}$

4.3 The substitution method

- Example

$T(1) = 1$

$T(n) = 2T(\lfloor n/2 \rfloor) + n, \text{ if } n > 1$

Guess $T(n) = O(n \lg n)$

Assume that $T(n) \leq cn \lg n$

Inductive step

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - (c-1)n \leq cn \lg n \end{aligned}$$

provided that $(c-1)n \geq 0 \Rightarrow c \geq 1, n \geq 1$

4.3 The substitution method

- Example (Cont'd)

  The basis step below may be omitted.

  $n = 1$ can't be the base case.

  $\because T(1) = 1 \leq c \cdot 1 \lg 1 = 0$ fails.

  $n = 2$ alone as the base case isn't enough.

  Bases: $n = 2, 3$

  $T(2) = 4 \leq c \cdot 2 \lg 2 = 2c \Rightarrow c \geq 2$

  $T(3) = 5 \leq c \cdot 3 \lg 3 \Rightarrow c \geq 5/3 \lg 3 \approx 1.05$

  All together, we may pick any $c \geq 2$ and and $n_0 = 2$.

4.3 The substitution method

- Example (Cont'd)

  Guess $T(n) = \Omega(n \lg n)$

  Assume that $T(n) \geq cn \lg n$

  Inductive step

  $$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\geq 2c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor + n \\ &= 2c\lfloor n/2 \rfloor (2 + \lg\lfloor n/2 \rfloor) - 4c\lfloor n/2 \rfloor + n \\ &\geq 2c((n-1)/2) \lg(4\lfloor n/2 \rfloor) - 4c\lfloor n/2 \rfloor + n \\ &= cn \lg(4\lfloor n/2 \rfloor) + n - c \lg(4\lfloor n/2 \rfloor) - 4c\lfloor n/2 \rfloor \\ &\geq cn \lg n, \text{ as long as } c \text{ is small enough and } n \text{ is large enough} \end{aligned}$$

  The point: floors and ceilings are tedious.

4.3 The substitution method

- Remark

  For asymptotic solutions, ceilings and floors may be omitted under certain conditions to make proof easier.

  For example,

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

  Guess $T(n) = \Omega(n \lg n)$

  Assume that $T(n) \geq cn \lg n$

  $$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\geq 2c(n/2)\lg(n/2) + n \\ &= cn \lg n + (1-c)n \\ &\geq cn \lg n, \quad \text{as long as } 1 \geq c \end{aligned}$$

4.3 The substitution method

- Remark (Cont'd)

  Guess $T(n) = O(n \lg n)$

  Assume that $T(n) \leq cn \lg n$

  $$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2c(n/2)\lg(n/2) + n \\ &= cn \lg n - (c-1)n \\ &\leq cn \lg n, \quad \text{as long as } c \geq 1 \end{aligned}$$

  What we have proven so far is $T(n) = \Theta(n \lg n)$, for $n = 2^k$.

  However, as we shall see, we may conclude that

  $T(n) = \Theta(n \lg n)$.

4.3 The substitution method

- Smooth functions

DEFINITION

A function f is b-smooth, where $b \geq 2$ is an integer, if it is monotonically increasing and $f(bn) = O(f(n))$

LEMMA

If f is b-smooth for some $b \geq 2$, then it is c-smooth $\forall c \geq 2$

Proof

Basis: $c = 2$

$f(2n) \leq f(bn)$ $\quad \because f$ is monotonically increasing

$= O(f(n))$

It follows that $f(2n) = O(f(n))$

4.3 The substitution method

- Smooth functions

  Inductive step

  $f((c+1)n) \le f(2cn)$ $\because f$ is monotonically increasing

  $\le d_1 f(cn)$ $\because f$ is 2-smooth, $f(2cn) = O(f(cn))$

  $\le d_1 d_2 f(n)$ $\because f$ is c-smooth, $f(cn) = O(f(n))$

  Thus, $f((c+1)n) = O(f(n))$

  By this lemma, we may simply say smooth functions.

  Polynomials are smooth. $(bn)^k = b^k n^k = O(n^k)$

  Polylogarithms are smooth. $\lg^k(bn) = (\lg b + \lg n)^k = O(\lg^k n)$

  Exponentials aren't smooth. $a^{bn} \neq O(a^n),\ b \ge 2, a > 1$

  In fact, $a^n = o(a^{bn})$

4.3 The substitution method

- Smooth functions

LEMMA

If f is smooth and t is monotonically increasing satisfying $t(n) = \Theta(f(n))$ for $n = b^k, b \geq 2$, then $t(n) = \Theta(f(n))$

Proof

For sufficient large n satisfying $b^k \leq n < b^{k+1}$, we have

$$
\begin{aligned}
t(n) &\leq t(b^{k+1}) && \because t \text{ is monotonically increasing} \\
&\leq c_1 f(b^{k+1}) && \because t(n) = O(f(n)) \text{ for } n = b^k \\
&\leq c_1 c_2 f(b^k) && \because f \text{ is smooth, } f(bn) = O(f(n)) \\
&\leq c_1 c_2 f(n) && \because f \text{ is monotonically increasing}
\end{aligned}
$$

$$\Rightarrow t(n) = O(f(n))$$

4.3 The substitution method

- Smooth functions

  LEMMA (Cont'd)

  For sufficient large n satisfying $b^k \leq n < b^{k+1}$, we have

  $t(n) \geq t(b^k)$ $\quad \because t$ is monotonically increasing

  $\geq c_3 f(b^k)$ $\quad \because t(n) = \Omega(f(n))$ for $n = b^k$

  $\geq (c_3/c_2) f(b^{k+1})$ $\quad \because f(b^{k+1}) \leq c_2 f(b^k)$

  $\geq (c_3/c_2) f(n)$ $\quad \because f$ is monotonically increasing

  $\Rightarrow t(n) = \Omega(f(n))$

4.3 The substitution method

- Smooth functions

Example

$T(n) = n$, if $n = 2^k$

$= n^2$, otherwise

$T(n)$ isn't monotonically increasing.

$T(n) = \Theta(n)$, for $n = 2^k$.

But, $T(n) = O(n^2)$.

Example

$T(1) = a$, where $a > 1$

$T(n) = T(\lceil n/2 \rceil)^2$, if $n > 1$

For $n = 2^k$, $T(n) = a^n = a^{2^{\lg n}}$

4.3 The substitution method

- Smooth functions

  Three steps for the closed form of the recurrence involving ceiling (or floor):

  - Generate the first few values

n	1	2	3	4	5	6	7	8	9~16
$T(n)$	a	a^2	a^4	a^4	a^8	a^8	a^8	a^8	a^{16}

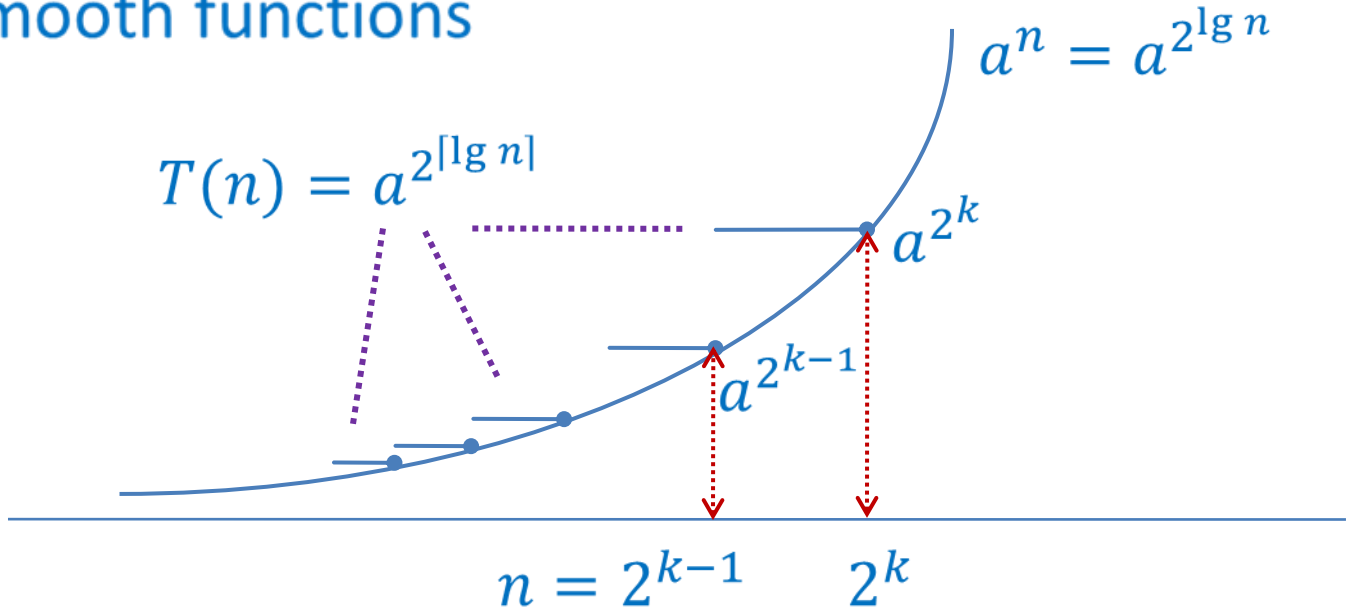
  - Guess the pattern

    It is easily seen that $T(n) = a^{2^{\lceil \lg n \rceil}}$, for all n

  - Prove by induction

4.3 The substitution method

- Smooth functions



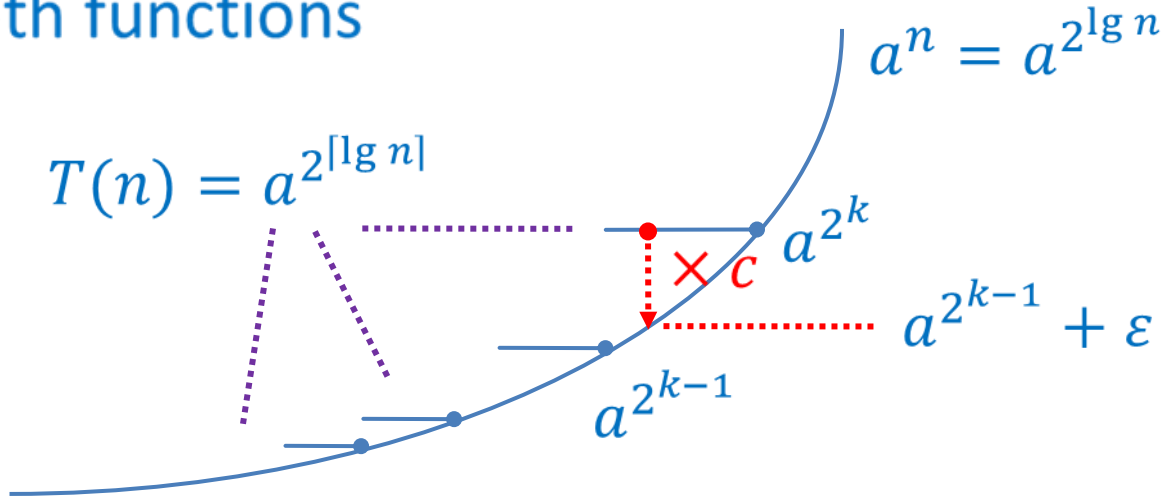
Clearly, $a^n = O\left(a^{2^{\lg n}}\right) = O\left(a^{2^{\lceil \lg n \rceil}}\right)$

Also, $a^n \neq o\left(a^{2^{\lceil \lg n \rceil}}\right)$, $\because a^n = a^{2^{\lceil \lg n \rceil}}$, for $n = 2^k$

But, $a^n \neq \Theta\left(a^{2^{\lceil \lg n \rceil}}\right)$!!! $\because a^n \neq \Omega\left(a^{2^{\lceil \lg n \rceil}}\right)$ See next page

4.3 The substitution method

- Smooth functions



FACT $a^n \neq \Omega\left(a^{2^{\lceil \lg n \rceil}}\right)$

$$\because c \cdot a^{2^k} \leq a^{2^{k-1}} + \varepsilon \Rightarrow c \leq \frac{a^{2^{k-1}} + \varepsilon}{a^{2^k}} \rightarrow \frac{1}{a^{2^{k-1}}}, \text{ as } \varepsilon \rightarrow 0$$

$$\therefore \text{As } k \rightarrow \infty, \frac{1}{a^{2^{k-1}}} \rightarrow 0 \Rightarrow c \text{ doesn't exist.}$$

4.3 The substitution method

- Asymptotic notation in recurrence

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

Upper bound: Guess $T(n) = O(n \lg n)$

Assume that $T(n) \le dn \lg n$

$$
\begin{aligned}
T(n) &= 2T(n/2) + O(n) \\
&= 2T(n/2) + f(n), \text{ where } f(n) = O(n) \\
&\le 2T(n/2) + cn, \text{for some constant } c \\
&\le 2d\frac{n}{2}\lg\frac{n}{2} + cn \\
&= dn \lg n - (d-c)n \\
&\le dn \lg n
\end{aligned}
$$

provided that $(d-c)n \ge 0 \Rightarrow d \ge c$

4.3 The substitution method

- Asymptotic notation in recurrence

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

Lower bound: Guess $T(n) = \Omega(n \lg n)$

Assume that $T(n) \geq dn \lg n$

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Omega(n) \\
&= 2T(n/2) + f(n), \text{ where } f(n) = \Omega(n) \\
&\geq 2T(n/2) + cn, \text{for some constant } c \\
&\geq 2d\frac{n}{2}\lg\frac{n}{2} + cn \\
&= dn \lg n + (c-d)n \\
&\geq dn \lg n
\end{aligned}
$$

provided that $(c-d)n \geq 0 \Rightarrow c \geq d$

4.3 The substitution method

- Asymptotic notation in recurrence

  Alternatively, for upper bound, we may simply write

  $$T(n) \le dn \lg n - (dn - O(n))$$
  $$\le dn \lg n$$

  because we can pick d large enough so that dn dominates the $O(n)$ term, i.e. $d \ge c$ previously;

  and, for lower bound,

  $$T(n) \ge dn \lg n + \Omega(n) - dn$$
  $$\ge dn \lg n$$

  because we can pick d small enough so that dn is dominated by the $\Omega(n)$ term, i.e. $c \ge d$ previously.

4.3 The substitution method

- Subtleties

  Prove the exact form of the inductive hypothesis

  $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

  Guess $T(n) = O(n)$

  Try $T(n) \leq cn$

  $T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$

  $= cn + 1 \not\leq cn$ NO!

  Next, try $T(n) \leq cn - d$

  $T(n) \leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1$

  $= cn - d - (d - 1)$

  $\leq cn - d$, as long as $d - 1 \geq 0 \Rightarrow d \geq 1$

4.3 The substitution method

- Subtleties

  Another example

  $T(n) = 8T(n/2) + O(n^2) \leq 8T(n/2) + cn^2$

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

  Try $T(n) \leq dn^3$

  $T(n) \leq 8d(n/2)^3 + cn^2 = dn^3 + cn^2 \not\leq dn^3$ NO!

  Next, try $T(n) \leq dn^3 - d'n^2$

  $$\begin{aligned} T(n) &\leq 8(d(n/2)^3 - d'(n/2)^2) + cn^2 \\ &= dn^3 - 2d'n^2 + cn^2 \\ &= dn^3 - d'n^2 - (d'n^2 - cn^2) \\ &\leq dn^3 - d'n^2 \end{aligned}$$

  as long as $d'n^2 - cn^2 \geq 0 \Rightarrow d' \geq c$

4.4 The recursion-tree method

- The tree-recursion method

  Use to generate a guess.

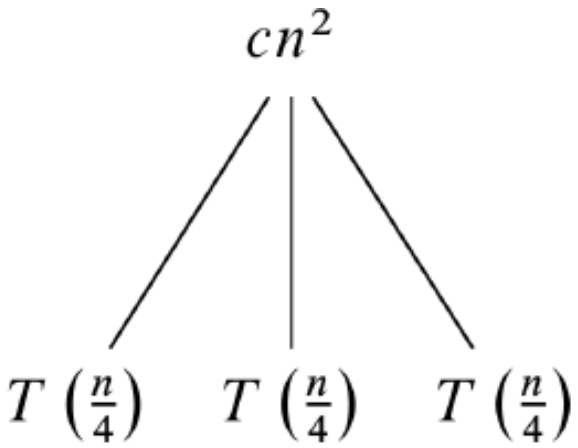
  Then, verify by substitution method

- Example

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

  Recursion tree

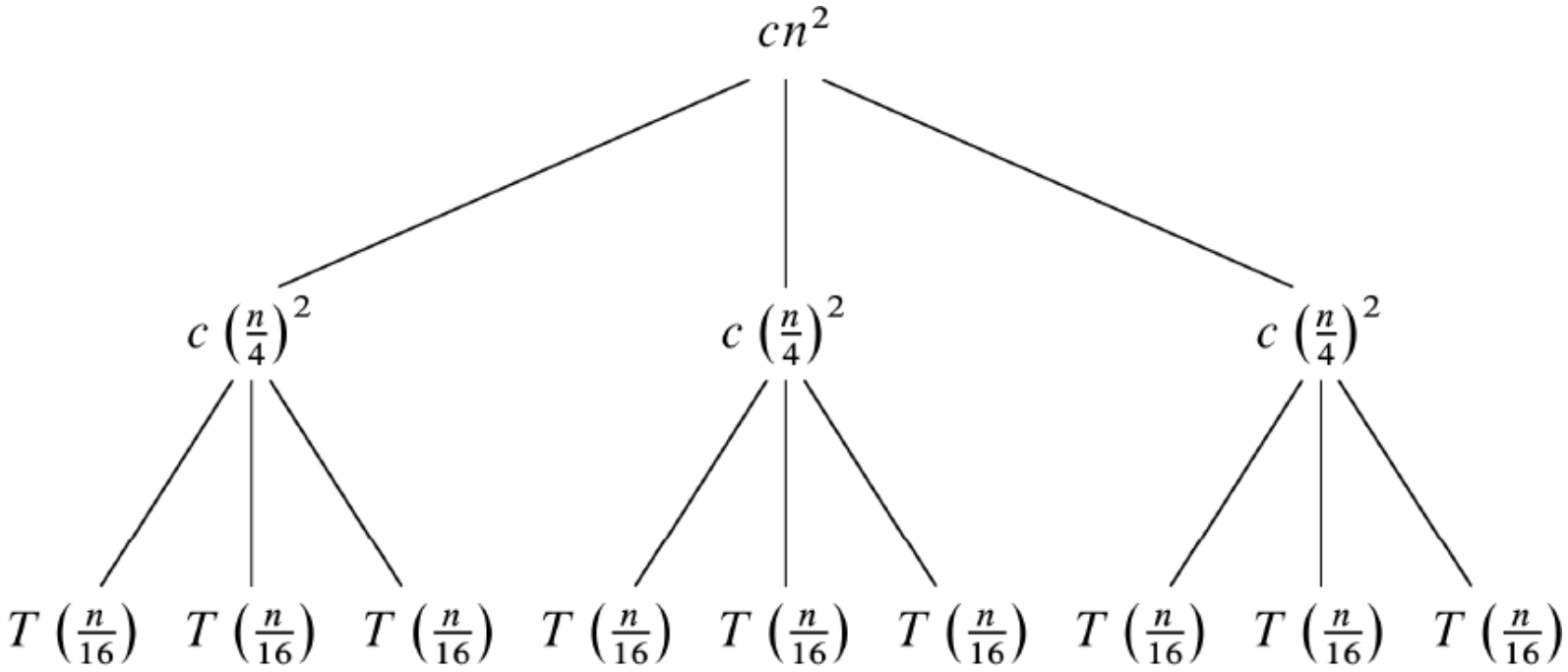
$T(n) \Rightarrow$



$\Rightarrow$ (*next page*)

4.4 The recursion-tree method

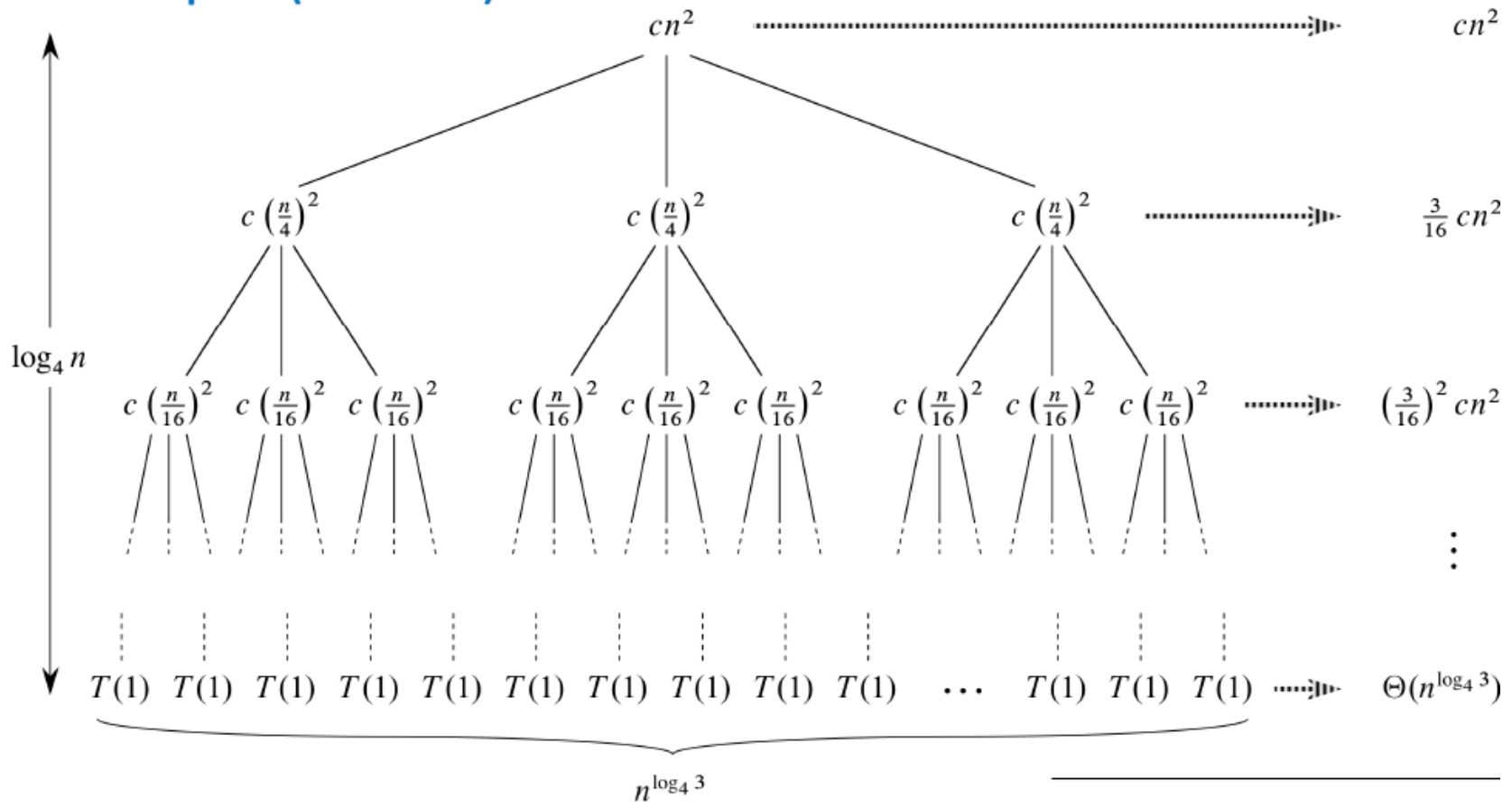
- Example (Cont'd)



$\Rightarrow$ (*next page*)

4.4 The recursion-tree method

- Example (Cont'd)



$$3^{\log_4 n}T(1) = n^{\log_4 3}\Theta(1) = \Theta\left(n^{\log_4 3}\right)$$

Total: $O(n^2)$

4.4 The recursion-tree method

- Example (Cont'd)

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

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

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

Next, verify by substitution method that $T(n) \leq dn^2$

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

$$\leq 3d(n/4)^2 + O(n^2)$$

$$= dn^2 - \left(\frac{13}{16}dn^2 - O(n^2)\right)$$

$$\leq dn^2$$

as long as d is large enough to make $\frac{13}{16}dn^2$ dominate $O(n^2)$

4.4 The recursion-tree method

- Example (Cont'd)

  On the other hand,

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

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

- Example

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

  For upper bound, rewrite it as

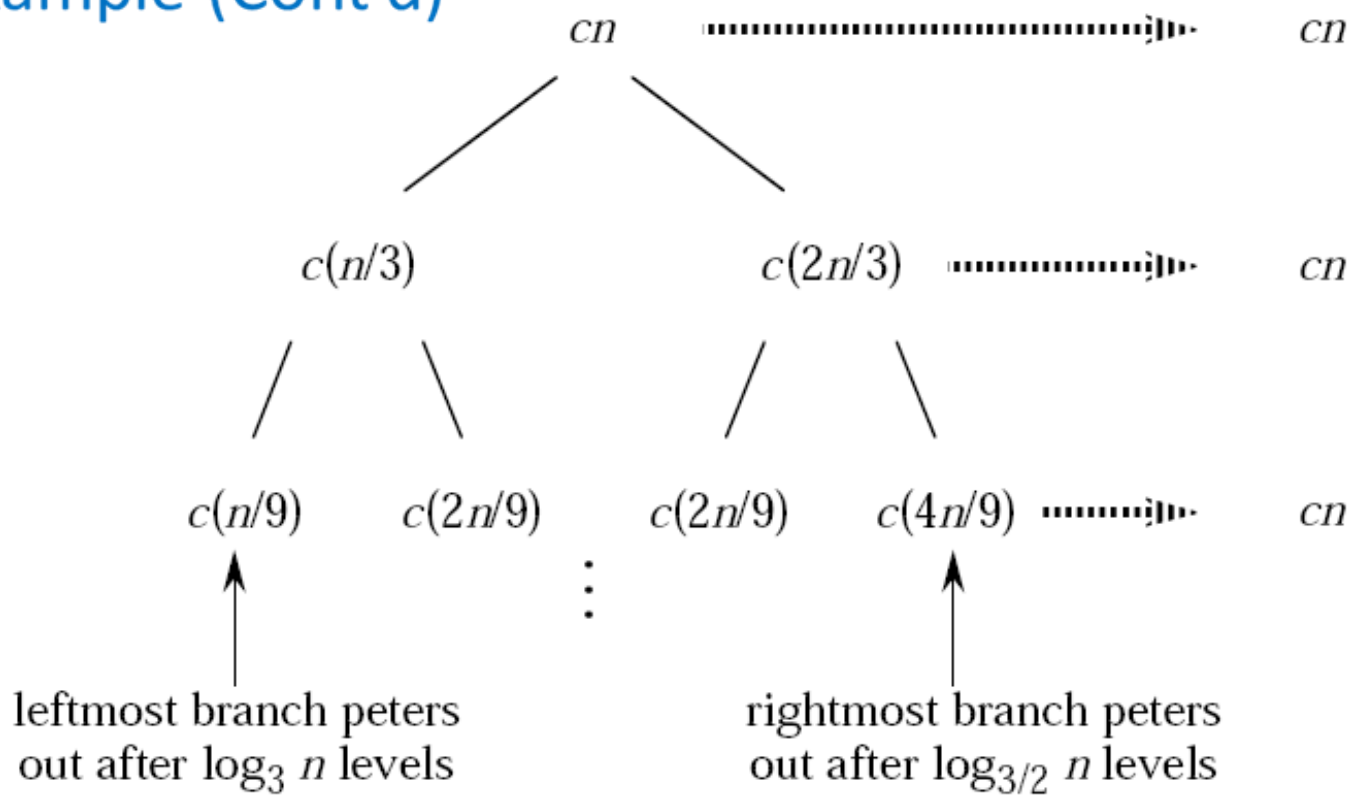
  $T(n) \leq T(n/3) + T(2n/3) + cn$

  For lower bound, rewrite it as

  $T(n) \geq T(n/3) + T(2n/3) + cn$

4.4 The recursion-tree method

- Example (Cont'd)



Each level $\geq cn$

Lower bound $\geq cn \log_3 n$

Each level $\leq cn$

Upper bound $\leq cn \log_{3/2} n$

4.4 The recursion-tree method

- Example (Cont'd)

  Upper bound

  Guess $T(n) = O(n \lg n)$ and assume that $T(n) \le dn \lg n$

$$
\begin{aligned}
T(n) &\le T(n/3) + T(2n/3) + cn \\
&\le d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= d(n/3)\lg(n/3) + 2d(n/3)(1 + \lg(n/3)) + cn \\
&= dn\lg(n/3) + 2d(n/3) + cn \\
&= dn\lg n - (dn(\lg 3 - 2/3) - cn) \\
&\le dn\lg n
\end{aligned}
$$

  as long as $dn(\lg 3 - 2/3) - cn \ge 0 \Rightarrow d \ge \dfrac{c}{\lg 3 - 2/3}$

4.4 The recursion-tree method

- Example (Cont'd)

  Lower bound

  Guess $T(n) = \Omega(n \lg n)$

  Assume that $T(n) \geq dn \lg n$

  The proof is the same as for the upper bound, but replacing $\leq$ by $\geq$.

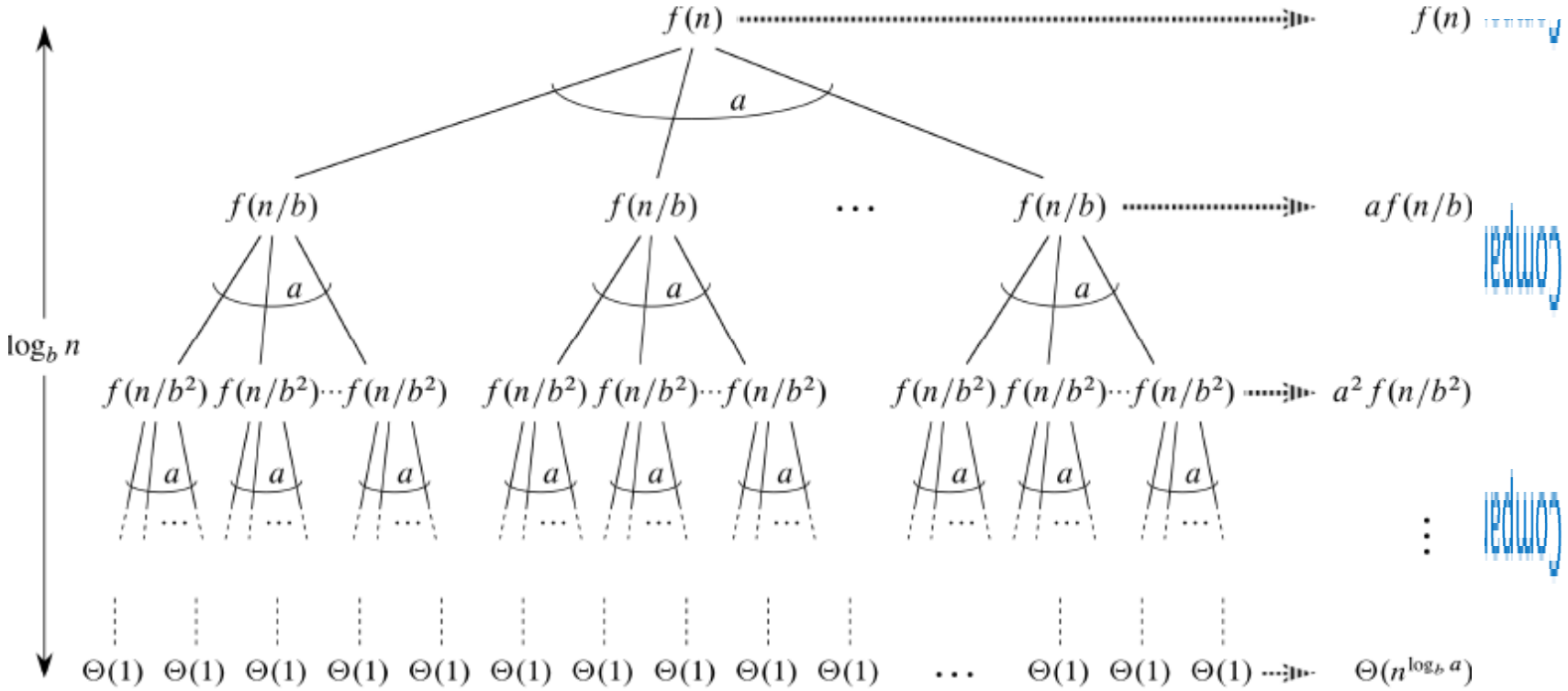
  End up needing

  $$d \leq \frac{c}{\lg 3 - 2/3}$$

4.5 The master method

- The master theorem

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1, \text{and } f(n) > 0$$



Total: $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

4.5 The master method

- Case 1

  $f(n) = O(n^{\log_b a - \varepsilon})$, for some constant $\varepsilon > 0$

  $f(n)$ is polynomially smaller than $n^{\log_b a}$.

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

  Intuitively, cost is dominated by leaves.

  Example

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

  $n^0 = O(n^{\log_2 2 - \varepsilon})$ for any $\varepsilon \leq 1 \Rightarrow T(n) = \Theta(n)$

  n^0 is polynomially smaller than $n^{\log_2 2} = n$.

4.5 The master method

- Case 1

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

$\lg n = O(n^{\log_2 2-\varepsilon})$ for any $\varepsilon < 1 \Rightarrow T(n) = \Theta(n)$

$\lg n$ is polynomially smaller than $n^{\log_2 2} = n$.

$T(n) = 5T(n/2) + \Theta(n^2)$

$n^2 = O(n^{\log_2 5-\varepsilon})$ for any $\varepsilon \leq \log_2 5 - 2 \Rightarrow T(n) = \Theta(n^{\lg 5})$

n^2 is polynomially smaller than $n^{\log_2 5}$.

$T(n) = 9T(n/3) + O(n)$

$n = O(n^{\log_3 9-\varepsilon})$ for any $\varepsilon \leq 1 \Rightarrow T(n) = O(n^2)$

n is polynomially smaller than $n^{\log_3 9} = n^2$.

4.5 The master method

- Case 2 (Exercise 4.6-2)

  $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k \geq 0$

  $f(n)$ is within a polylog factor of $n^{\log_b a}$, but not smaller.

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

  Intuitively, cost is $n^{\log_b a} \lg^k n$ at each level, and there are $\Theta(\lg n)$ levels.

  Simple case

  $k = 0 \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$

4.5 The master method

- Case 2

  Example

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

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

  n is within a polylog factor of $n^{\log_2 2} = n$, but not smaller.

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

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

  $n \lg n$ is within a polylog factor of $n^{\log_2 2} = n$, but not smaller.

4.5 The master method

- Case 3

  $f(n) = \Omega(n^{\log_b a+\varepsilon})$, for some constant $\varepsilon > 0$

  and satisfies the regularity condition:

  $af(n/b) \leq cf(n)$ for some $c < 1$ and sufficiently large n

  $f(n)$ is polynomially greater than $n^{\log_b a}$.

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

  Intuitively, cost is dominated by root.

4.5 The master method

- Case 3

  LEMMA (Exercise 4.6-3)

  The regularity condition implies $f(n) = \Omega(n^{\log_b a+\varepsilon}), \varepsilon > 0$

  Proof

  First show by induction that

  $a^k f\left(n/b^k\right) \leq c^k f(n)\ \forall k \geq 1$, where $c < 1$

  Next, let $k = \log_b n$

  $a^{\log_b n} f(n/b^{\log_b n}) \leq c^{\log_b n} f(n)$

  $\Rightarrow n^{\log_b a} f(1) \leq n^{\log_b c} f(n)$

  $\Rightarrow n^{\log_b a - \log_b c} f(1) \leq f(n)$

  $\Rightarrow f(n) = \Omega(n^{\log_b a+\varepsilon})$, where $\varepsilon = -\log_b c > 0, \because c < 1$

4.5 The master method

- Case 3

  LEMMA

  $f(n) = n^i \lg^j n$ and $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some $\varepsilon > 0$

  $\Rightarrow af(n/b) \le cf(n)$ for some $c < 1$

  Proof

$$
\begin{aligned}
af(n/b) &= a(n/b)^i \lg^j (n/b) \\
&= (a/b^i) n^i (\lg n - \lg b)^j \\
&= (a/b^i) n^i (\lg^j n - \Theta(\lg^{j-1} n)) \\
&\le (a/b^i) n^i \lg^j n \\
&= cf(n), \text{ where } c = a/b^i
\end{aligned}
$$

4.5 The master method

- Case 3

  We have to show that $c = a/b^i < 1$.

  $f(n) = n^i \lg^j n$ and $f(n) = \Omega(n^{\log_b a+\varepsilon})$

  $\Rightarrow i \geq \log_b a + \varepsilon$

  $\Rightarrow i > \log_b a$

  $\Rightarrow b^i > b^{\log_b a} = a \Rightarrow a/b^i < 1$

  Example

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

  $n^2 = \Omega(n^{\log_2 2+\varepsilon})$ for any $\varepsilon \leq 1 \Rightarrow T(n) = \Theta(n^2)$

  n^2 is polynomially greater than $n^{\log_2 2} = n$.

  N.B. There is no need to check the regularity condition

4.5 The master method

- Case 3

  Example

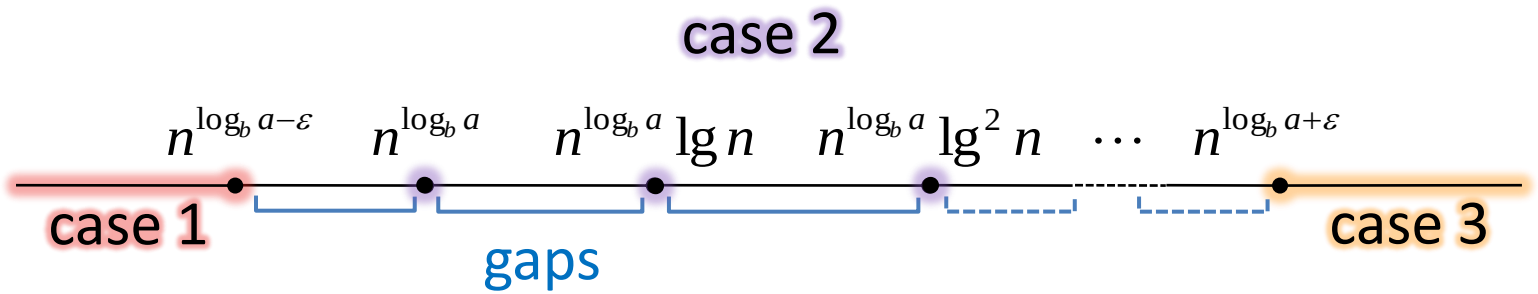
  $T(n) = 5T(n/2) + \Theta(n^3)$

  $n^3 = \Omega(n^{\log_2 5+\varepsilon})$ for any $\varepsilon \le 3 - \log_2 5 \Rightarrow T(n) = \Theta(n^3)$

  n^3 is polynomially greater than $n^{\log_2 5}$.

- Gaps

  There are gaps that can't be solved by the master theorem.



4.5 The master method

- Gaps

  Example

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

  $n/\lg n$ is smaller than $n^{\log_2 2} = n$, but not polynomially smaller.

  In fact, $n/\lg n = \omega(n^{1-\varepsilon})$, since $n^{1-\varepsilon} = n/n^{\varepsilon}$

  Since

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

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

  it is reasonable to guess that $T(n) = \Theta(n \lg \lg n)$.

  And, indeed it is.

4.5 The master method

- Changing variables

  Example

  $T(n) = T(\sqrt{n}) + \lg n$

  Let $m = \lg n$, then $n = 2^m$

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

  Let $S(m) = T(2^m)$, then

  $S(m) = S(m/2) + m$

  $m = \Omega\left(m^{\log_2 1+\varepsilon}\right)$ for any $\varepsilon \leq 1$

  $\Rightarrow S(m) = \Theta(m)$ by case 3

  $\Rightarrow T(n) = T(2^m) = S(m) = \Theta(m) = \Theta(\lg n)$