Divide-and-Conquer (Recurrences)

Divide-and-Conquer: 4-1x

Divide: (into the same problems of smaller size)

Conquer:

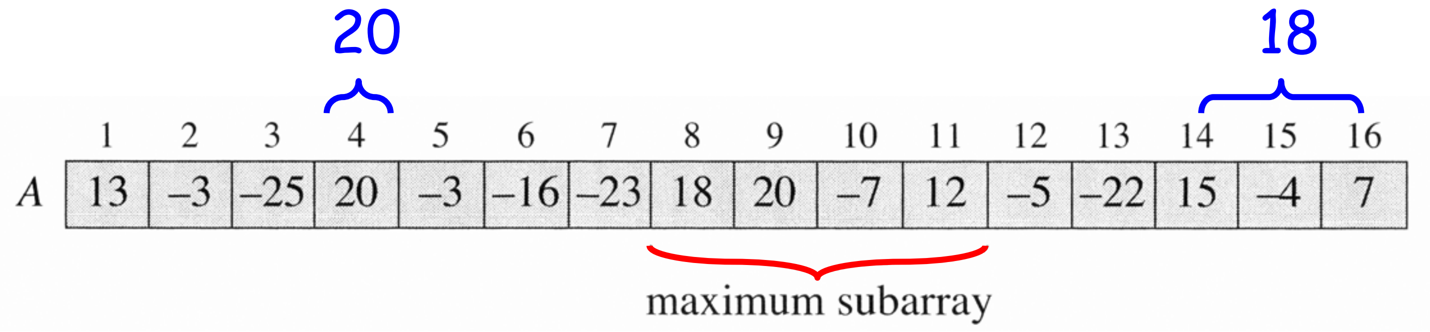
Combine:

Two examples of divide-and-conquer: 4.1, 4.2
Solving recurrences: 4.3, 4.4, 4.5

4.1 The maximum-subarray problem

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

Output: a nonempty subarray $A[i..j]$ having the largest sum $S[i, j] = a_i + a_{i+1} + \ldots + a_j$



$S[8, 11] = 43$

A brute-force solution

all pairs of i, j

4-2xy

* Examine all $\binom{n}{2}$ possible $S[i, j]$

* Two implementations

$O(j - i + 1)$
||

(1) compute each S[i, j] in $O(n)$ time $\Rightarrow$ $O(n^3)$ time

(2) compute each $S[i, j+1]$ from $S[i, j]$ in $O(1)$ time
($S[i, i] = A[i]$ and $S[i, j+1] = S[i, j] + A[j+1]$)
$\Rightarrow$ $O(n^2)$ time

(ex. S[2, 12] = S[2, 11] + A[12])

⇩ i = 2

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$A[i]$	13	-15	23	4	-13	-16	-23	18	20	-7	12	-5	-22

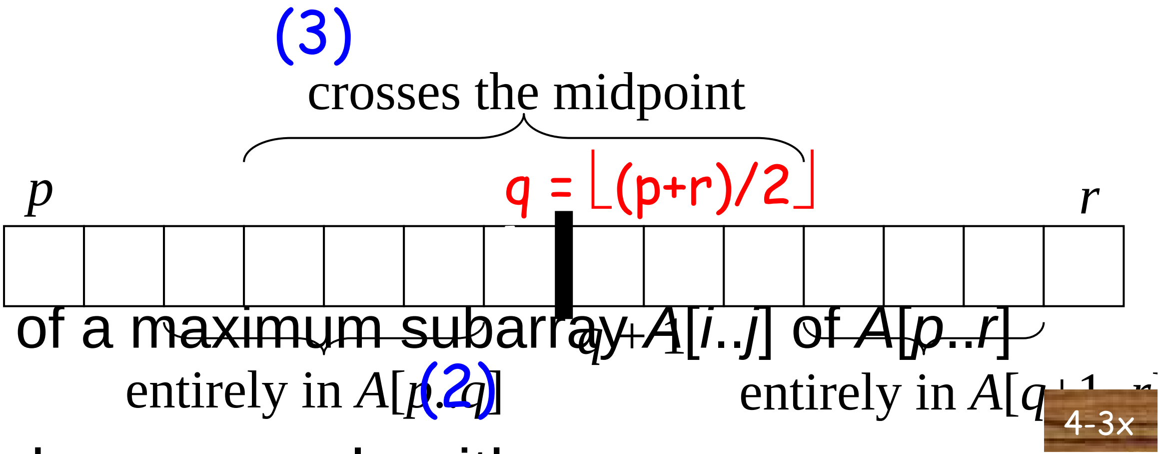
$S[2, 2]$ = -15
$S[2, 3]$ = 8
$S[2, 4]$ = 12
$S[2, 5]$ = -1

⇨ O(n) time for each i

A divide-and-conquer solution

* Possible locations of a maximum subarray $A[i..j]$ of $A[p..r]$, where $q = \lfloor (p+r)/2 \rfloor$

(1) entirely in $A[p..q]$
(2) entirely in $A[q+1..r]$
(3) crossing the midpoint ($p \le i < q < j \le r$)



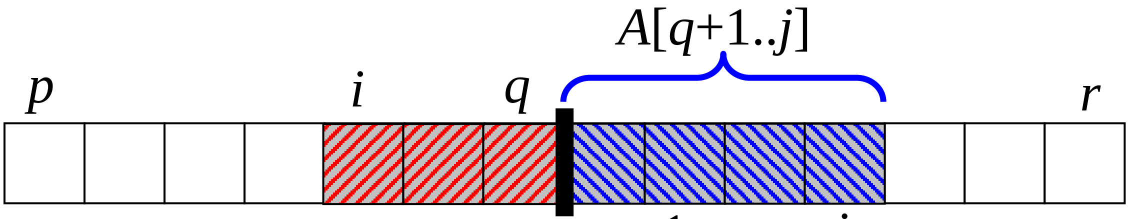
Locations of a maximum subarray $A[i..j]$ of $A[p..r]$

(1)

* A divide-and-conquer algorithm

FINDMAXSUBARRAY(A, p, r)
if $p = r$ **then return** (p, p, $A[p]$)   //base case
take it even negative
(nonempty subarray)
else
$q \leftarrow \lfloor (p + r) / 2 \rfloor$
recursive calls
(i_1, j_1, s_1) $\leftarrow$ FINDMAXSUBARRAY(A, p, q)
(i_2, j_2, s_2) $\leftarrow$ FINDMAXSUBARRAY(A, q+1, r)
(i_c, j_c, s_c) $\leftarrow$ ~~FINDMAXCROSSING(A, p, q, r~~)
if $s_1 \ge s_2$ and ~~$s_1 \ge s_c$ **then return** (i_1, j_1, s_1~~)
elseif $s_2 \ge$ ~~s_c **then return** (i_2, j_2, s_2)~~
else return (i_c, j_c, s_c)

* Find a maximum subarray crossing the midpoint



$A[i..j]$ comprises two subarrays
$A[i..q]$ and $A[q+1..j]$

FINDMAXCROSSING(A, p, q, r)

```
s1 ← −∞
sum ← 0
for i ← q downto p do
      sum ← sum + A[i]
      if sum > s1
         then s1 ← sum
              maxleft ← i
s2 ← −∞
sum ← 0
for j ← q + 1 to r do
      sum ← sum + A[j]
      if sum > s2
         then s2 ← sum
              maxright ← j
return (maxleft, maxright, s1 + s2)
```

Find maxleft, s_1
($A[i..q]$)

Find maxright, s_2
($A[q+1..j]$)

Example:

q = 6

A

1	2	3	4	5	6	7	8	9	10	11	12
-7	8	-5	20	-3	-8	-23	18	20	-7	12	-5

$s_1 = -\infty$ maxleft

S[6, 6] = -8 -8 6
S[5, 6] = -11
S[4, 6] = 9 9 4
S[3, 6] = 4
S[2, 6] =` ☆ 12 ⇐ (maxleft = 2) 12 2
S[1..6] = 5

***q* = 6**

A

1	2	3	4	5	6	7	8	9	10	11	12
-7	8	-5	20	-3	-8	-23	18	20	-7	12	-5

$s_2 = -\infty$

S[7, 7] = -23 -23
S[7, 8] = -5 -5
S[7, 9] = 15 15
S[7, 10] = 8
S[7, 11] =` (maxright = 11) ⇒ 20 ☆ 20
S[7, 12] = 15

⇒ maximum subarray crossing *q* is *A*[2, 11]
(with *S*[2, 11] = 32)

* Time complexity

(1) FINDMAXCROSSING: $\Theta(n)$, where $n = r - p + 1$

(2) FINDMAXSUBARRAY:

$$T(n) = \underline{2T(n/2) + \Theta(n)} \quad \text{(with } T(1) = \Theta(1)\text{)}$$
$$= \underline{\Theta(n\lg n)} \quad \text{(similar to merge-sort)}$$

Remark: See Ex4.1-5 for an $O(n)$-time algorithm.

4.2 Strassen's algorithm for matrix multiplication

4-6a

4-6y

Input: two $n \times n$ matrices A and B

Output: $C = AB$, where $c_{i,j} = \sum_{1 \le k \le n} a_{ik} b_{kj}$

An $O(n^3)$ time naive algorithm

```
SQUARE-MATRIX-MULTIPLY(A, B)
n ← rows[A]
let C be an n × n matrix
for i ← 1 to n do
    for j ← 1 to n do
        c_ij ← 0
        for k ← 1 to n do
            c_ij ← c_ij + a_ik • b_kj
return C
```

compute c_{ij}

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

* Computing $A+B \rightarrow O(n^2)$ time

Strassen's algorithm

* Assume that n is an exact power of 2

* We divide each of A, B, and C into four $n/2 \times n/2$ sub-matrices and rewrite $C = AB$ as

$$\frac{n}{2}\times\frac{n}{2} \rightarrow \overset{C}{\begin{pmatrix} r & s \\ t & u \end{pmatrix}} = \overset{A}{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}\overset{B}{\begin{pmatrix} e & g \\ f & h \end{pmatrix}} \qquad \text{(EQ-1)}$$

* We have

$$r = \overset{1}{ae} + \overset{2}{bf} \qquad s = \overset{3}{ag} + \overset{4}{bh}$$
$$t = \underset{5}{ce} + \underset{6}{df} \qquad u = \underset{7}{cg} + \underset{8}{dh}$$

* A straightforward divide-and-conquer algorithm

$$\begin{aligned} T(n) &= 8T(n/2) + O(n^2) \\ &= O(n^3) \qquad \hookrightarrow 4\times\left(\frac{n}{2}\right)^2 \text{ for addition} \end{aligned}$$

* Let

$P_1 = \overset{1}{a}(g\text{-}h)$ (=ag-ah)

$P_2 = (a+b)\overset{2}{h}$ (=$ah+bh$)

$P_3 = (c+d)\overset{3}{e}$ (=$ce+de$)

$P_4 = \overset{4}{d}(f\text{-}e)$ (=df-de)

$P_5 = (a+d)\overset{5}{}(e+h)$ (=$ae+ah+de+dh$)

$P_6 = (b\text{-}d)\overset{6}{}(f+h)$ (=$bf+bh$-df-dh)

$P_7 = (a\text{-}c)\underset{7}{}(e+g)$ (=$ae+ag$-ce-cg) (EQ-2)

* We have

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7 \qquad \text{(EQ-3)}$$

* Strassen's divide-and-conquer algorithm

Step 1: Divide each of *A*, *B*, and *C* into four sub-matrices. (EQ-1)

Step 2: Recursively, compute $\underbrace{P_1, P_2, \ldots, P_7}_{7\ '\times',\ 10\ '+'}$. (EQ-2)

Step 3: Compute $\underbrace{r, s, t, u}_{8\ '+'}$ according to EQ-3.

* Time complexity

$$\begin{aligned} T(n) &= 7T(n/2) + O(n^2) \\ &= O(n^{\log_2 7}) \\ &= O(n^{2.81}) \end{aligned}$$

$18 \times (\frac{n}{2})^2$ (for addition)

(by Master Thm)

Discussion:

1. Strassen's method is largely of theoretical interest. (for $n \geq 45$)

$T(n) = qT(\frac{n}{2}) + O(n^2)$ "$q < 7$?"

2. Strassen's method is based on the fact that we can multiply two 2×2 matrices using only 7 multiplications (instead of 8). It was showed that it is impossible to multiply two 2×2 matrices using less than 7 multiplications.

4-9x

3. We can improve Strassen's algorithm by finding an efficient way to multiply two $k \times k$ matrices using a smaller number q of multiplications, where $k > 2$. The time is $T(n) = qT(n/k) + O(n^2)$.

$q < k^3$

4. The current best upper bound is $O(n^{2.376})$. *1990

*2010: 2.374; 2011: 2.3728642; 2014: 2.3728639

4.3 The substitution method

The substitution method: (i) Guess an answer and then (ii) prove it by induction. (for both upper and lower bounds)

Example: Find an upper bound for

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \qquad \text{(with } T(1) = 1\text{)}$$

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

(ii) Try to prove there exist constants c and n_0 such that $T(n) \le cn \lg n$ for all $n \ge n_0$.

Basis: ($n = n_0$)

For $n = 1$, no constant c satisfies $T(1) \le cn \lg n$=0.
For $n \ge 2$, any constant $c \ge T(n)/(n \lg n)$ satisfies $T(n) \le cn \lg n$. That is, we can choose

(1) $n_0 \ge 2$ and $c \ge T(n_0)/(n_0 \lg n_0)$.

Induction: ($n > n_0$)

Assume: $T(x) \le cx \lg x$ for $x = n_0 \sim n$-1

Assume that it holds for all n between n_0 and n–1. We have

將 $T(x) \le cx \lg x$ 代 入

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

$$
\begin{aligned}
T(n) &\le 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n \qquad \textbf{(Substitution)} \\
&\le cn \lg (n/2) + n \\
&= cn \lg n - cn \lg 2 + n \\
&= cn \lg n - cn + n \\
&\le cn \lg n, \quad \text{goal}
\end{aligned}
$$

where the last step holds for

(2) $\quad c \ge 1.$

* to make substitution holds, we also need $n_0 \le \lfloor n/2 \rfloor < n-1$
=> $n_0 = 2, 3$, $n \ge 4$

From (1) and (2), we can choose $n_0 = 2, 3$ and $c = \max\{1, T(2)/(2 \lg 2), T(3)/(3 \lg 3)\} = 2$ to make both the *basis* and the *induction* steps holds.

Substitution Method

Step 1. Guess $T(n) = O(g(n))$

Step 2. Prove the guess by induction

Prove $T(n) = O(g(n))$

$\Rightarrow$ Prove that there are c and n_0 such that
$T(n) \le cg(n)$ for all $n \ge n_0$ ------------(1)

$\Rightarrow$ If c and n_0 are known, we can prove (1) by induction

(a) Basis step: (1) holds for $n = n_0$
(b) Induction step: (1) holds for $n > n_0$

$\Rightarrow$ How to find c and n_0 satisfying the induction proof?

(i) find the condition of c and n_0 for which the basis step holds
(ii) find the condition of c and n_0 for which the induction step holds
(iii) Combine conditions (i) and (ii)

/ˈsʌtltɪ/ 微妙之處(細微的差別)

***Subtleties*:** 4-12a

(Revise a guess by subtracting a lower-order term.) $\Rightarrow$ induction proof does not always work unless the exact form is given

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

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

$\exists\ c, n_0$ s.t.

Try to prove $T(n) \leq cn$. (for all $n \geq n_0$)

Basis: ok!

Assume: $T(x) \le cx$ for $x = n_0 \sim n-1$

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

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

$= cn + 1$

We can not prove that $T(n) \le cn$!!!!

goal

Try to prove $T(n) \le cn - b$. (for $n \ge n_0$)

Induction: $T(n) \le (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$

$= cn - 2b + 1$

$\le cn - b$, goal

where the last step holds for any constant $b \ge 1$.

Basis: ($n_0 = 1$) $1 \le c - b$

$\Rightarrow c \ge b + 1$

陷 阱

Avoiding pitfalls

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

Guess $T(n) = O(n)$. Try to prove $T(n) \le cn$.

Induction: $T(n) \le 2c\lfloor n/2 \rfloor + n$

$\le cn + n$

$= O(n)$ $\Leftarrow$===== ***wrong !!***

4-13x

$\le cn$ (goal)

Assume: $T(x) \le cx$ for $x = n_0 \sim n-1$

Changing variable:

$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$

For simplicity, assume $n = 2^m$. Then

$T(2^m) = 2T(2^{m/2}) + m$

Let $S(m) = T(2^m)$. We have

$S(m) = 2S(m/2) + m$ (Renaming $m = \lg n$)

$T(n) = S(m) = S(\lg n) = \lg n \lg\lg n$

Since $O(m \lg m)$ is the solution to $S(m)$, we know that $O(\lg n \lg\lg n)$ is the solution to $T(n)$.

4.4 The iteration (recursion-tree) method

4-14x

Example: $T(n) = 3T(\lfloor n/4 \rfloor) + n$ with $\begin{cases} T(0) = c = \Theta(1) \\ T(1) = c = \Theta(1) \end{cases}$

$T(x) = x + 3T(\lfloor x/4 \rfloor)$

4-14y

$$\begin{aligned} T(n) &= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) \\ &= n + 3\lfloor n/4 \rfloor + 9(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor)) \\ &\;. \\ &\;. \\ &\le n + 3n/4 + 9n/16 + 27n/64 + \ldots + 3^{\log_4 n}\underbrace{\Theta(1)}_{T(0),T(1)} \\ &\le n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i + \Theta(n^{\log_4 3}) = 4n + o(n) = O(n) \end{aligned}$$

$3^k T(\lfloor \frac{n}{4^k} \rfloor) \Rightarrow \frac{n}{4^k} \le 1 \Rightarrow k \ge \lg_4 n$

(note that $n/(4^{\log_4 n}) \le 1$)

$* \; a^{\lg_c b} = b^{\lg_c a}$

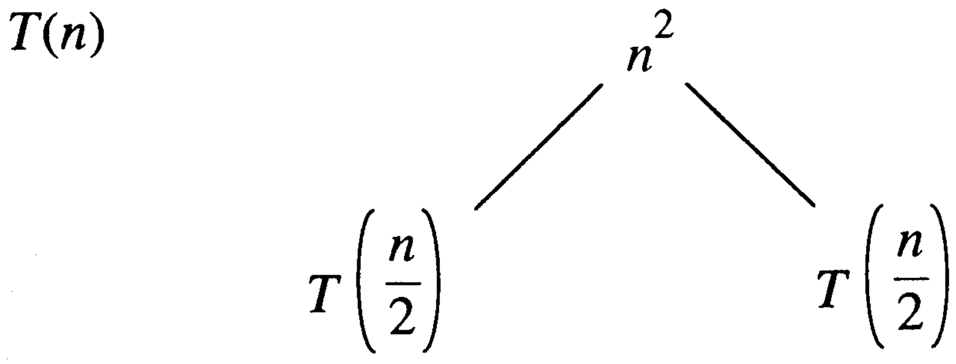
$* \; \lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor$ (similar for ceiling)

Recursion trees: (for visualizing the iteration)

4-14z

with $T(1) = 1$ or $\Theta(1)$

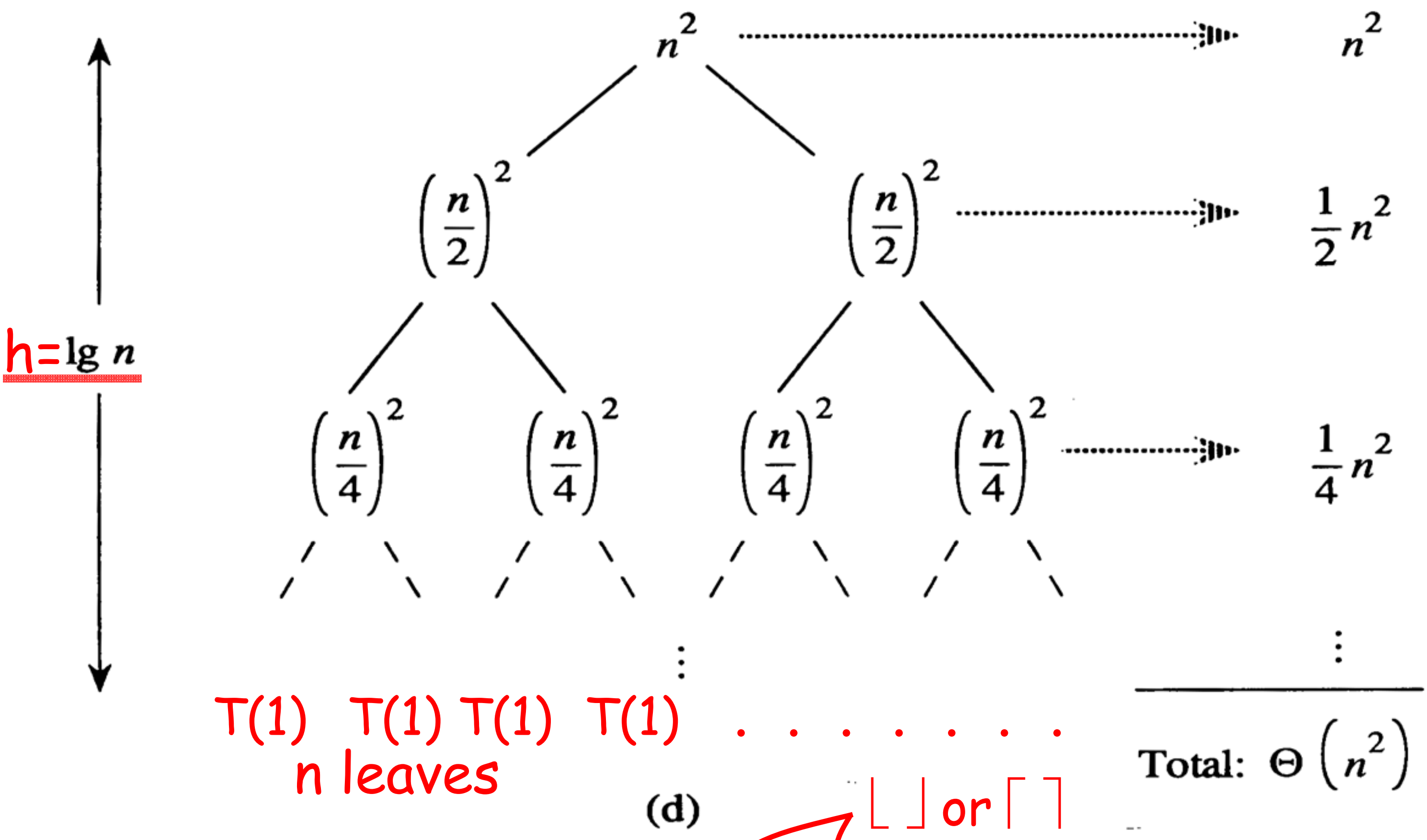
$T(n) = 2T(n/2) + n^2$ (Assume that $n = 2^h$.)



(a) (b)

Cost (Internal nodes) + Cost (leaves)

$\rightsquigarrow O(n^2)$ $\rightsquigarrow L\times\theta(1) = O(n)$

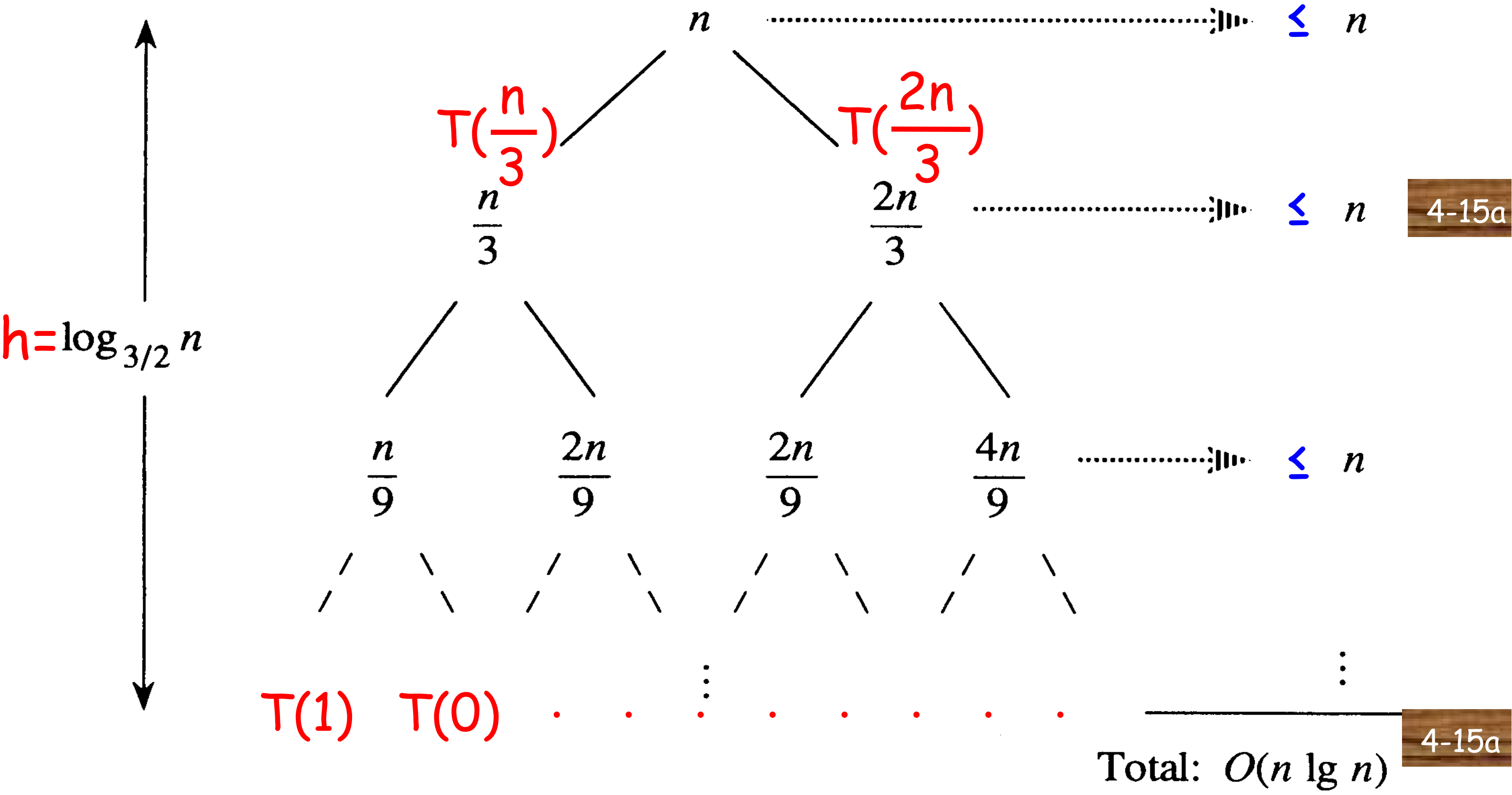


⌊ ⌋ or ⌈ ⌉

但不可以同時取⌈ ⌉

Example: $T(n) = T(n/3) + T(2n/3) + n$

$(T(0)=T(1)=T(2)=1 \text{ or } \Theta(1))$



* Using recursive trees to make a good guess for S.M.

4-15y

4.5 The master method

***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, $T(n)$ can be bounded as follows.

4-16a

1. If $f(n) = O(n^{(\log_b a)-\varepsilon})$ for some constant $\varepsilon > 0$, 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. ① If $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ for some constant $\varepsilon > 0$, and if ② $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

$a=9$ $b=3$ * $\log_b a = 2$

Example: $T(n) = 9T(n/3) + \underbrace{n}_{f(n)}$

By applying case 1, we have $T(n) = \Theta(n^2)$.

$a=1$ $b=3/2$ * $\log_b a = 0$

Example: $T(n) = T(2n/3) + \underbrace{1}_{f(n)}$

By applying case 2, we have $T(n) = \Theta(\lg n)$.

$a=3$ $b=4$ * $\log_4 3 \approx 0.793$

Example: $T(n) = 3T(n/4) + n \lg n$ * $c = 3/4$

By applying case 3, we have $T(n) = \Theta(n \lg n)$.

Note: The three cases do not cover all the possibilities for $f(n)$. There are gaps between cases 1 and 2, and between cases 2 and 3. 4-16a

Example: $T(n) = 2T(n/2) + \underline{n \lg n}$ $O(n \lg^2 n)$ (recursion tree)

In this example, both cases 2 and 3 cannot be applied.

*Case 1. $O(n^{\log_b a - \varepsilon}) = o(n^{\log_b a})$???

4-16a

Homework: Ex. 4.1-5, 4.2-1, 4.2-4, 4.2-5, 4.2-7, 4.3-5 (using substitution method), 4.4-6, 4.4-9, 4.5-2, and Pro. 4-5bc (using substitution method), 4-6de.

$$T(n) = 2T\left(\frac{n}{2}\right) + \begin{cases} n \\ n^2 \\ n \lg n \end{cases} \Rightarrow \begin{matrix} n \lg n & \text{(recur. tree, MS)} \\ \underline{n^2} & \text{(recur. tree, MS)} \\ n \lg^2 n & \text{(recur. tree)} \end{matrix}$$