Chapter 4 Divide-and-Conquer

About this lecture (1)

- Recall the divide-and-conquer paradigm, which we used for merge sort:
  - Divide the problem into a number of subproblems that are smaller instances of the same problem.
  - Conquer the subproblems by solving them recursively.
    - ***Base case: If the subproblems are small enough, just solve them by brute force.***
  - Combine the subproblem solutions to give a solution to the original problem.
- We look at two more algorithms based on divide-and-conquer.

About this lecture (2)

- Analyzing divide-and-conquer algorithms
- Introduce some ways of solving recurrences
  - Substitution Method (If we know the answer)
  - Recursion Tree Method (Very useful !)
  - Master Theorem (Save our effort)

Maximum-subarray problem

- Input: an array *A*[1..*n*] of *n* numbers
  - Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative
- Output: a nonempty subarray *A*[*i*..*j*] having the largest sum $S[i, j] = A[i] + A[i+1] + \ldots + A[j]$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	**13**	**-3**	**-25**	**20**	**-3**	**-16**	**-23**	**18**	**20**	**-7**	**12**	**-5**	**-22**	**15**	**-4**	**7**

maximum subarray

How to solve this problem?

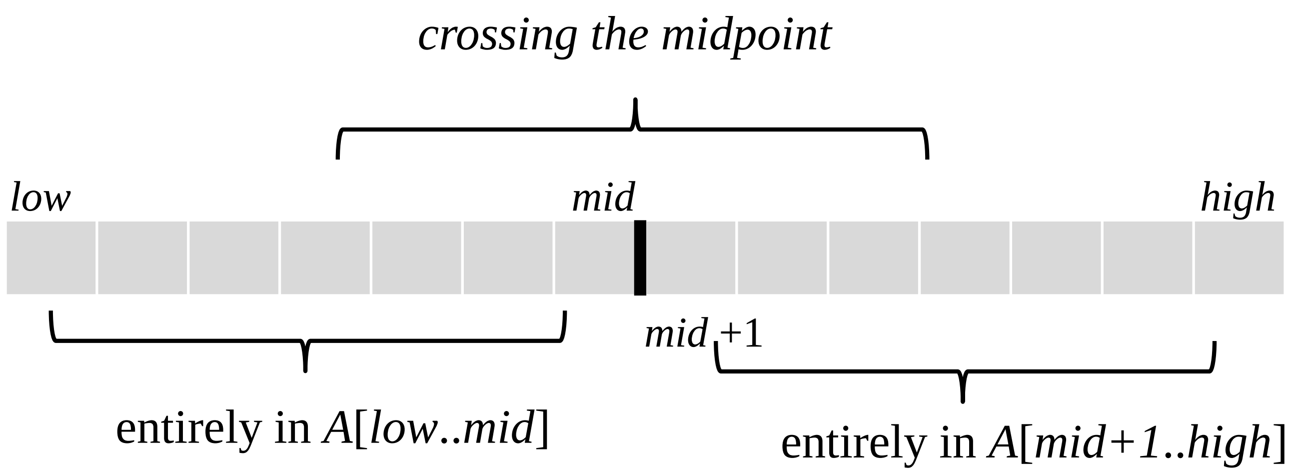
A brute-force solution

- Examine all $\binom{n}{2}$ possible $S[i, j]$
- Two implementations:
  - compute each S[i, j] in $O(n)$ time $\Rightarrow$ $O(n^3)$ time
  - 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: i	1	2	3	4	5	6
A[i]	13	-3	-25	20	-3	-16
$S[1, j]$ =	13	10	-15	5	2	-14
S[2, j] =		-3	-28	-8	-11	-27
S[3, j] =			-25	-5	-8	-34
S[4, j] =				20	17	1
S[5, j] =					-3	-19
S[6, j] =						-16

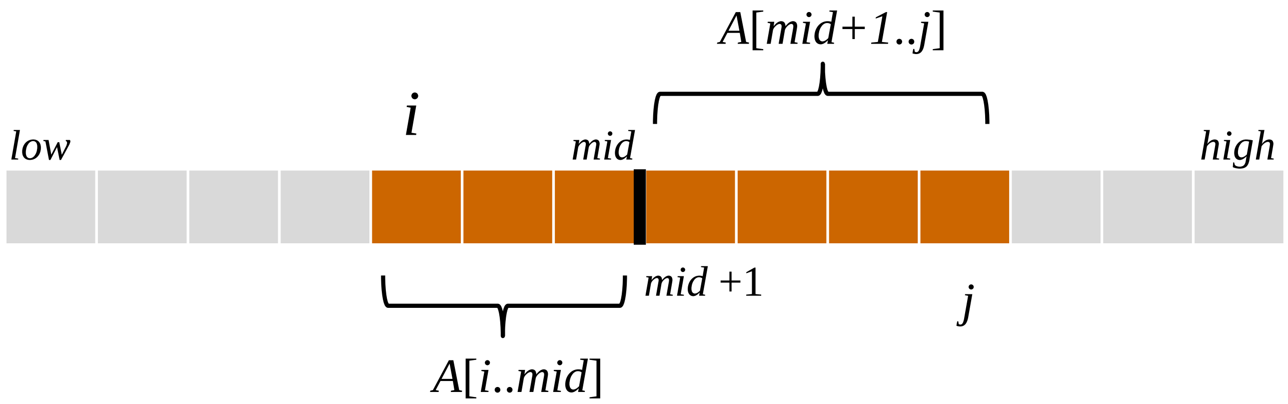
A divide-and-conquer solution

- Possible locations of a maximum subarray *A*[*i*..*j*] of *A*[*low*..*high*], where *mid* = $\lfloor$(*low*+*high*)/2$\rfloor$
  – entirely in *A*[*low*..*mid*] (*low* ≤ *i* ≤ *j* ≤ *mid*)
  – entirely in *A*[*mid*+1..*high*] (*mid* < *i* ≤ *j* ≤ *high*)
  – crossing the midpoint (***low* ≤ *i* ≤ *mid* < *j* ≤ *high***)



Possible locations of subarrays of A[low..high]

Crossing the midpoint



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

How to find the maximum subarray $A[i..j]$ crossing the midpoint?

Example:

mid =5

	1	2	3	4	5	6	7	8	9	10
A	13	-3	-25	20	-3	-16	-23	18	20	-7

$S[5 .. 5] =$ -3

$S[4 .. 5] =$ 17 $\Leftarrow$ **(max-left = 4)**

$S[3 .. 5] =$ -8

$S[2 .. 5] =$ -11

$S[1 .. 5] =$ 2

mid =5

	1	2	3	4	5	6	7	8	9	10
A	13	-3	-25	20	-3	-16	-23	18	20	-7

$S[6 .. 6] =$ -16

$S[6 .. 7] =$ -39

$S[6 .. 8] =$ -21

$S[6 .. 9] =$ **(max-right = 9)** $\Rightarrow$ -1

$S[6..10] =$ -8

$\Rightarrow$ **maximum subarray crossing *mid* is $S[4..9] = 16$**

Example:

mid =3

	1	2	3	4	5
A	13	-3	-25	20	-3

$S[3 .. 3] =$ -25
$S[2 .. 3] =$ -28
$S[1 .. 3] =$ -15 $\Leftarrow$ (max-left = 1)

mid =3

	1	2	3	4	5
A	13	-3	-25	20	-3

$S[4 .. 4] =$ (max-right = 4) $\Rightarrow$ 20
$S[4 .. 5] =$ 17

$\Rightarrow$ **maximum subarray crossing *mid* is $S[1..4] = 5$**

mid =7

	6	7	8	9	10
A	-16	-23	18	20	-7

$S[8 .. 8] =$ 18 $\Leftarrow$ (max-left = 8)
$S[7 .. 8] =$ -5
$S[6 .. 8] =$ -21

mid =7

	6	7	8	9	10
A	-16	-23	18	20	-7

$S[9 .. 9] =$ (max-right = 9) $\Rightarrow$ 20
$S[9 ..10] =$ 13

$\Rightarrow$ **maximum subarray crossing *mid* is $S[8..9] = 38$**

Example:

mid =2

1	2	3
13	-3	-25

S[2 .. 2] = -3
S[1 .. 2] = 10 (max-left = 1)
S[3 .. 3] = -25 (max-right = 3)

maximum subarray crossing *mid* is *S*[1..3] = -15

mid =4

4	5
20	-3

S[4 .. 4] = 20 (max-left = 4)
S[5 .. 5] = -3 (max-left = 5)

maximum subarray crossing *mid* is *S*[4..5] = 17

mid =7

	6	7	8
A	-16	-23	18

S[7 .. 7] = -23 (max-left = 7)
S[6 .. 7] = -39
S[8 .. 8] = 18 (max-right = 8)

maximum subarray crossing *mid* is *S*[7..8] = -5

mid =9

	9	10
A	20	-7

S[9 .. 9] = 20 (max-left = 9)
S[10 ..10] = -7 (max-right = 10)

maximum subarray crossing *mid* is *S*[9..10] = 13

FIND-MAX-CROSSING-SUBARRAY(A, *low*, *mid*, *high*)

```
left-sum = -∞   // Find a maximum subarray of the form A[i..mid]
 sum = 0
for i = mid downto low
    sum = sum + A[i ]
    if sum > left-sum
       left-sum = sum
       max-left = i
right-sum = - ∞ // Find a maximum subarray of the form A[mid + 1 .. j ]
sum =0
for j = mid +1 to high
    sum = sum + A[j]
    if sum > right-sum
       right-sum = sum
       max-right = j
// Return the indices and the sum of the two subarrays
Return (max-left, max-right, left-sum + right-sum)
```

FIND-MAXIMUM-SUBARRAY (A, *low*, *high*)

if *high* == *low*
Return (*low*, *high*, *A*[*low*]) // ***base case: only one element***
else *mid* = $\lfloor (low + high)/2 \rfloor$
(*left-low*, *left-high*, *left-sum*) =
FIND-MAXIMUM-SUBARRAY(*A*, *low*, *mid*)
(*right-low*, *right-high*, *right-sum*) **=**
FIND-MAXIMUM-SUBARRAY(*A*, *mid* + 1, *high*)
(*cross-low*, *cross-high*, *cross-sum*) =
FIND-MAX-CROSSING-SUBARRAY(*A*, *low*, *mid*, *high*)
if *left-sum* ≧ *right-sum* ***and*** *left-sum* ≧ *cross-sum*
return (*left-low*, *left-high*, *left-sum*)
elseif ***right-sum*** ≧ ***left-sum and right-sum*** ≧ ***cross-sum***
return (*right-low*, *right-high*, *right-sum*)
else return (*cross-low*, *cross-high*, *cross-sum*)

Initial call: FIND-MAXIMUM-SUBARRAY (A, 1, n)

Analyzing time complexity

- FIND-MAX-CROSSING-SUBARRAY : $\Theta(n)$,

  where $n = high - low + 1$
- FIND-MAXIMUM-SUBARRAY

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

  $= \Theta(n\lg n)$ (similar to merge-sort)

Matrix multiplication

- Input: two $n \times n$ matrices A and B
- Output: $C = AB$, $$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

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

SQUARE-MATRIX-MULTIPLY(A, B)
$n \leftarrow$ A.*rows*
let C be an $n \times n$ matrix
for $i \leftarrow 1$ **to** n
  for $j \leftarrow 1$ **to** n
    $c_{ij} \leftarrow 0$
    for $k \leftarrow 1$ **to** n
      $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$
return C

Divide-and-Conquer Algorithm

- Assume that *n* is an exact power of 2

$$A = \begin{pmatrix} A_{11} A_{12} \\ A_{21} A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} B_{12} \\ B_{21} B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} C_{12} \\ C_{21} C_{22} \end{pmatrix}$$

$$\begin{pmatrix} C_{11} C_{12} \\ C_{21} C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} A_{12} \\ A_{21} A_{22} \end{pmatrix} \bullet \begin{pmatrix} B_{11} B_{12} \\ B_{21} B_{22} \end{pmatrix} \tag{4.1}$$

Divide-and-Conquer Algorithm

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

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

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

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

- A straightforward divide-and-conquer algorithm

$T(n) = 8T(n/2) + \Theta(n^2)$ (Computing $A+B \rightarrow O(n^2)$)
$= \Theta(n^3)$ (why?)

Strassen's method (1)

$$S_1 = B_{12} - B_{22},\ S_2 = A_{11} + A_{12},\ S_3 = A_{21} + A_{22}$$

$$S_4 = B_{21} - B_{11},\ S_5 = A_{11} + A_{22},\ S_6 = B_{11} + B_{22}$$

$$S_7 = A_{12} - A_{22},\ S_8 = B_{21} + B_{22},\ S_9 = A_{11} - A_{21}$$

$$S_{10} = B_{11} + B_{12} \qquad (4.2)$$

Strassen's method (2)

$$P_1 = A_{11}S_1$$

$$P_2 = S_2 B_{22}$$

$$P_3 = S_3 B_{11}$$

$$P_4 = A_{22}S_4$$

$$P_5 = S_5 S_6 \qquad (4.3)$$

$$P_6 = S_7 S_8$$

$$P_7 = S_9 S_{10}$$

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

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4 \qquad (4.4)$$

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

Strassen's divide-and-conquer algorithm

- **Step 1**: Divide each of A, B, and C into four sub-matrices as in (4.1)
- **Step 2**: Create 10 matrices S_1, S_2, …, S_{10} as in (4.2)
- **Steep 3**: Recursively, compute P_1, P_2, …, P_7 as in (4.3)
- **Step 4**: Compute $C_{11}, C_{12}, C_{21}, C_{22}$ according to (4.4)

Time complexity

$$T(n) = 7T(n/2) + \Theta(n^2)$$
$$= \Theta(n^{\lg 7})$$ (why?)
$$= \Theta(n^{2.81})$$

Discussion

- Strassen's method is largely of theoretical interest for $n \geq 45$
- 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 shown that it is impossible to multiply two 2×2 matrices using less than 7 multiplications.

Discussion

- 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 $T(n) = qT(n/k) + \theta(n^2)$.
- A trivial lower bound for matrix multiplication is $\Omega(n^2)$. The current best upper bound known is $O(n^{2.376})$.
- Open problems:
  - Can the upper bound $O(n^{2.376})$ be improved?
  - Can the lower bound $\Omega(n^2)$ be improved?

Practice at home

- Exercise: 4.1-4, 4.1-5, 4.2-3, 4.2-7

Substitution Method (1)

(if we know the answer)

How to solve this?

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

1. Make a guess

   e.g., $T(n) = O(n \lg n)$

2. Show it by induction

   - e.g., to show upper bound, we find constants c and n_0 such that $T(n) \le c\, g(n)$ for $n \ge n_0$

Substitution Method (2)

(if we know the answer)

How to solve this?

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

1. Make a guess

   e.g., $T(n) = O(n \lg n)$

2. Show it by induction

   - Firstly, $T(2) = 4$, $T(3) = 5$.

     ➔ We want to have $T(n) \leq cn \lg n$

     ➔ Let $c = 2$ ➔ $T(2)$ and $T(3)$ okay

   - Other Cases ?

Substitution Method (3)

(if we know the answer)

- Base case: n_0 = 2 hold.
- Induction Case:

Assume the guess is true for all n = 2, 3,..., k

For n = k+1, we have:

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

$$\leq 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n$$

$$\leq cn \lg n/2 + n$$

$$= cn \lg n - cn + n \leq cn \log n$$

Induction case is true

Substitution Method (4)

(if we know the answer)

Q. How did we know the value of c and n_0 ?

A. If induction works, the induction case must

be correct ➔ $c \geq 1$

Then, we find that by setting $c = 2$, our guess is correct as soon as $n_0 = 2$

Alternatively, we can also use $c = 1.5$ Then, we just need a larger $n_0 = 4$

(What will be the new base case? Why?)

Substitution Method (5)

(New Challenge)

How to solve this?

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

1. Make a guess ($T(n) = O(n)$), and
2. Show $T(n) \le cn$ by induction
   - What will happen in induction case?

Substitution Method (6)

(New Challenge)

- Assume guess is true for some base cases
- Induction Case:

Assume the guess is true for all n = 2, 3,..., k

For n = k+1, we have:

$$
\begin{aligned}
\mathrm{T}(n) &= \mathrm{T}(\lfloor n/2 \rfloor) + \mathrm{T}(\lceil n/2 \rceil) + 1 \\
&\le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\
&= cn + 1
\end{aligned}
$$

This term is not what we want …

Substitution Method (7)
(New Challenge)

- The 1st attempt was not working because our guess for T(n) was a bit "loose"
- Recall: Induction may become easier if we prove a "stronger" statement
- 2nd Attempt: Refine our statement

  Try to show T(n) $\leq cn - b$ instead

Substitution Method (8)

(New Challenge)

Induction Case:

$$
\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\
&\le c\lfloor n/2 \rfloor - b + c\lceil n/2 \rceil - b + 1 \\
&\le cn - b
\end{aligned}
$$

We get the desired term (when b ≥ 1)

It remains to find c and n_0, and prove the base case(s), which is relatively easy.

If c = 2, n_0 = ? If c = 1.5, n_0 = ?

Avoiding Pitfalls

- For T(n) = 2T($\lfloor n/2 \rfloor$) + n, we can falsely prove T(n) = O(n) by guessing T(n) $\leq cn$ and then arguing

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

Wrong!!

What's wrong with it?

Your friend, after this lecture, has tried to prove

$1+2+\ldots+n = O(n)$

- His proof is by induction:
- First, $1 = O(n)$ {n=1}
- Assume $1+2+\ldots+k = O(n)$ {n=k}
- Then, $1+2+\ldots+k+(k+1) = O(n) + (k+1)$ {n=k+1}

$= O(n) + O(n) = O(2n) = O(n)$

So, $1+2+\ldots+n = O(n)$ [where is the bug??]

Substitution Method

(New Challenge 2)

How to solve this?

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

Hint: Change variable: Set m = lg n

Substitution Method

(New Challenge 2)

Set $m = \lg n$, we get

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

Next, rename $S(m) = T(2^m) = T(n)$

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

We solve $S(m) = O(m \lg m)$

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

Recursion Tree Method (1)

(Nothing Special… Very Useful !)

How to solve this?

$$T(n) = 2T(n/2) + n^2, \quad \text{with } T(1) = 1$$

Recursion Tree Method (2)

(Nothing Special… Very Useful !)

Expanding the terms, we get:

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

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

$$= n^2 + n^2/2 + n^2/4 + 8T(n/8)$$

$$= \ldots$$

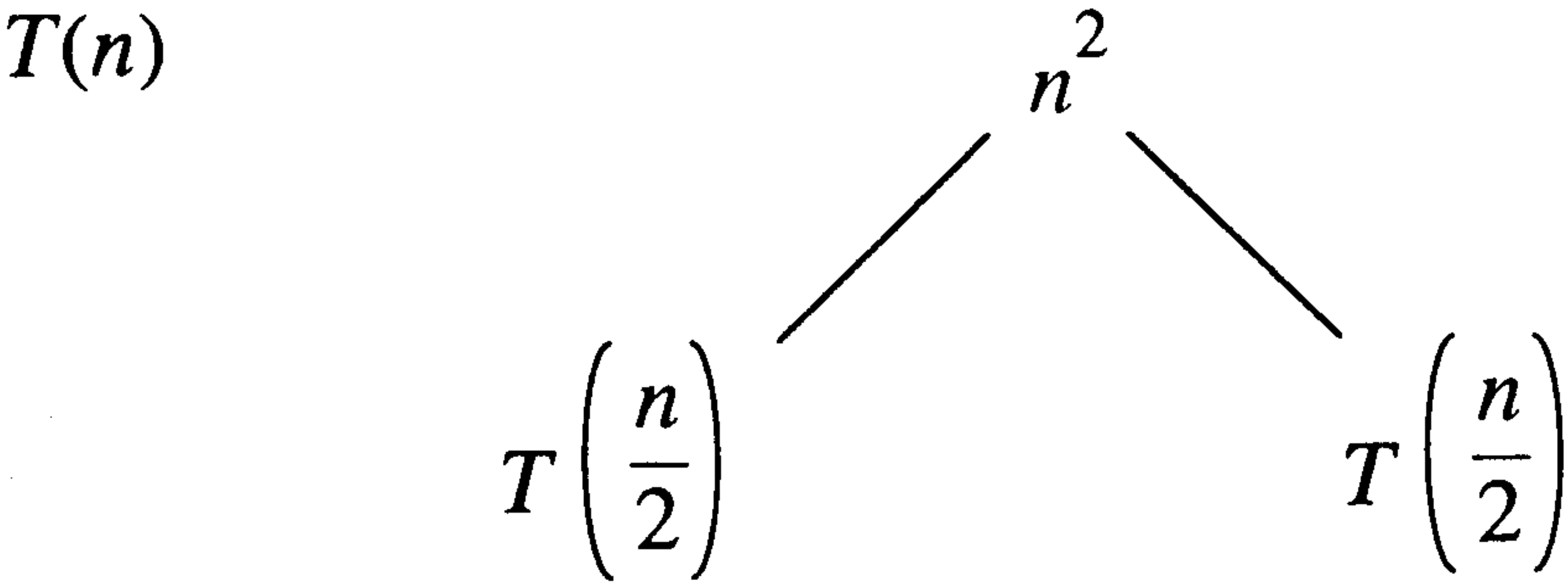
$$= \sum_{k=0}^{\lg n - 1} (1/2)^k n^2 + 2^{\lg n} T(1)$$

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

Recursion Tree Method (3)

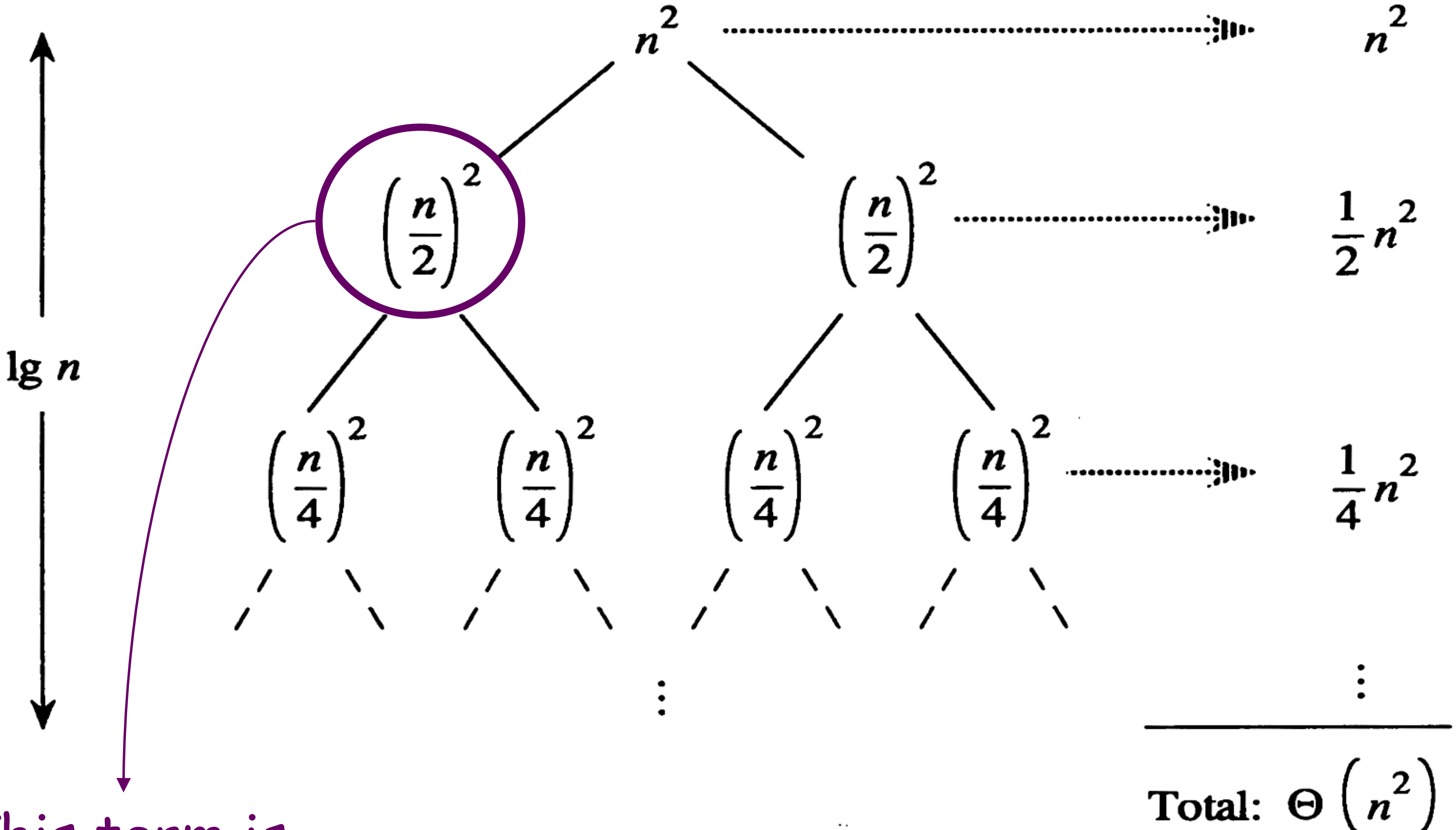
(Recursion Tree View)

We can express the previous recurrence by:



(a) (b)

Further expressing gives us:



This term is from T(n/2)

(d)

Recursion Tree Method

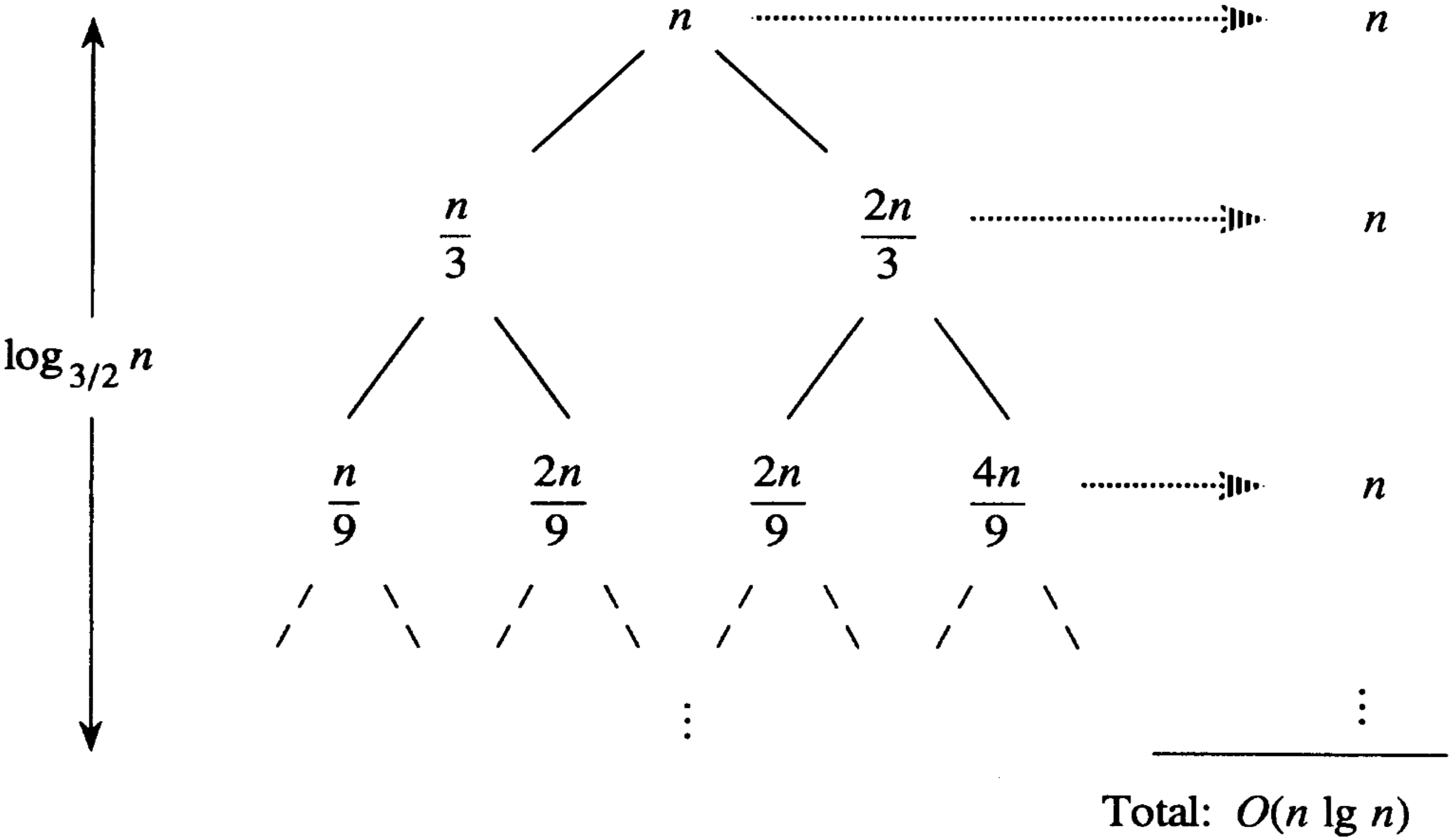
(New Challenge)

How to solve this?

$$T(n) = T(n/3) + T(2n/3) + n, \quad \text{with } T(1) = 1$$

What will be the recursion tree view?

The corresponding recursion tree view is:



The depth of the tree is $\log_{3/2} n$. Why?

Master Method

(Save our effort)

- When the recurrence is in a special form, we can apply the Master Theorem to solve the recurrence immediately.

- Let $T(n) = aT(n/b) + f(n)$

with $a \geq 1$ and $b > 1$ are constants, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

- The Master Theorem has 3 cases …

Master Theorem (1)

Let $T(n) = aT(n/b) + f(n)$

Theorem 1: (*Case* 1)

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

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

For example: $T(n) = 2\,T(n/2) + 1$

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

Master Theorem (2)

1. Solve T(n) = 9T(n/3) + n , T(1) = 1

We have a = 9 b= 3, f(n) =n

Since $n^{\log_b a} = n^{\log_3 9} = n^2$, f(n)= n = $O(n^{2-\varepsilon})$

We have T(n) = $\Theta(n^2)$, where ε = 1.

2. T(n) = 8T(n/2) + n^2 , T(1) = 1

- ✓ a = 8, b = 2, and f(n) = $\Theta(n^2)$, we can apply case 1, and T(n) = $\Theta(n^3)$.
- ✓ How about T(n) = 8T(n/2) + n ?

Master Theorem (3)

3. $T(n) = 7T(n/2) + n^2$

- ✓ $a = 7$, $b = 2$, $n^{\log_b a} = n^{\lg 7} \approx n^{2.81}$, and $f(n) = \Theta(n^2)$, we can apply case 1 and
- ✓ $T(n) = \Theta(n^{2.81})$

How about $T(n) = 7T(n/2) + 1$?

Master Theorem (4)

- Let $T(n) = aT(n/b) + f(n)$
- Theorem 2: (Case 2)

If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$

1. Solve $T(n) = T(2n/3) + 1$

Here, $a = 1$, $b = 3/2$, $f(n) = 1$, and $n^{\log_b a} = n^{\log_{3/2} 1} = 1$. Case 2 applied, $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$. Thus $T(n) = \Theta(\lg n)$

2. How about $T(n) = 4T(n/2) + n^2$?

3. How about $T(n) = T(n/2) + 1$

Master Theorem (5)

- Let $T(n) = aT(n/b) + f(n)$

- Theorem 3: (*Case* 3)

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

Master Theorem (6)

1. Solve $T(n) = 3T(n/4) + n \lg n$ (case 3)

- ✓ $a = 3$, $b = 4$, $f(n) = n \lg n$, and $n^{\log_4 3} = O(n^{0.793})$.
- ✓ $f(n) = \Omega(n^{0.793 + \varepsilon})$, where $\varepsilon \approx 0.2$, $af(n/b) = 3f(n/4) = 3(n/4) \lg (n/4) \leq (3/4)\, n \lg n = c\, f(n)$, for $c = 3/4$
- ✓ $T(n) = \Theta(n \lg n)$

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

Master Theorem (7)

- ✓ Note that, there is a gap between case 1 and case 2 when f(n) is smaller than $n^{\log_b a}$, but not polynomial smaller.
  - For example: T(n) = 2T(n/2) + n/lg n
- ✓ Similarly, there is a gap between cases 2 and 3 when f(n) is larger than $n^{\log_b a}$ but not polynomial larger.
- ✓ In the above cases, you cannot apply master theorem.

Master Theorem (8)

- For example: $T(n) = 2T(n/2) + n \lg n$

$a = 2$, $b = 2$, $f(n) = n \lg n$ and $n^{\log_b a} = n$.

You cannot apply case 3. Why?

$n \lg n / n = \lg n$ is smaller than n^{ε} for any positive constant ε.

Practice at Home

- Exercise: 4.3-2, 4.3-6, 4.3-8, 4.3-9, 4.4-5, 4.4-6, 4.4-8, 4.5-1, 4.5-4
- Problem: 4.1(b, d, f), 4.3 (b, e, h)

- 4.3-8: $T(n) = 4T(n/2) + n$