CS 421: Design and Analysis of Algorithms

Chapter 4: Divide and Conquer

Dr. Gaby Dagher
Department of Computer Science
Boise State University

February 8, 2022

BOISE STATE UNIVERSITY

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - Powering a number
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - The Recursion-Tree Method
  - The Master Method

Content of this Chapter

- **Divide and Conquer**
  - Maximum-subarray
  - Binary search
  - Exponentiation
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - The Recursion-Tree Method
  - The Master Method

Divide-and-conquer Design Paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

Divide-and-conquer: Merge Sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

Divide-and-conquer Cases

- Recursive case(s):
  - Subproblems are large enough to solve recursively.
- Base case(s):
  - Subproblems become small enough that we no longer recurse.
  - The recursion "bottoms out".

Recurrence

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

Merge Sort:

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

 # subproblems

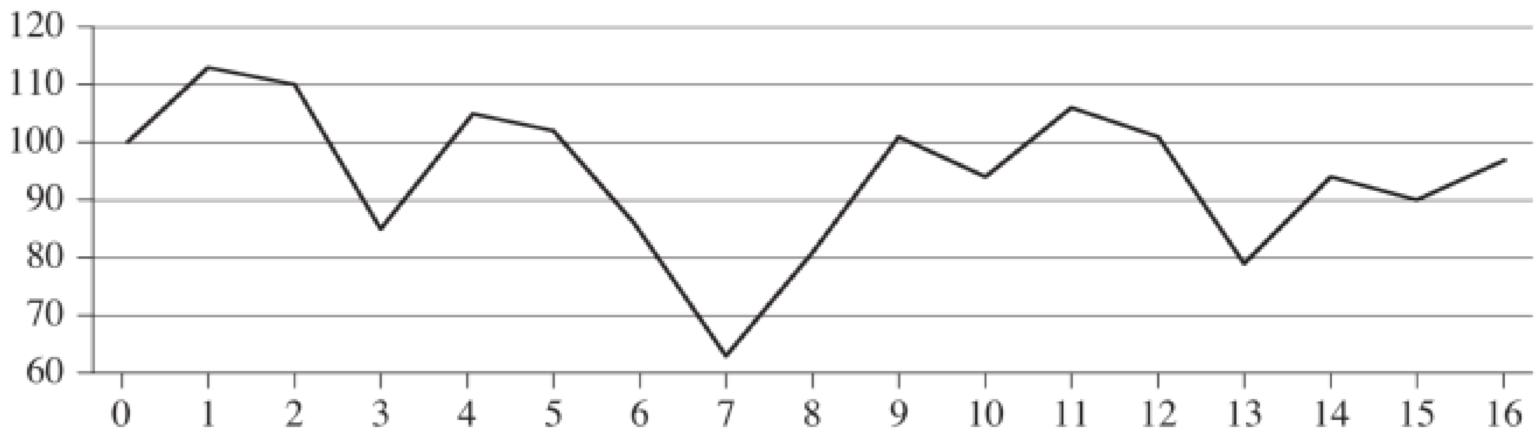
subproblem input size

Work: dividing $\Theta(1)$ + *combining* $\Theta(n)$

Content of this Chapter

- Divide and Conquer
  - **Maximum-subarray**
  - Binary search
  - Powering a number
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - The Recursion-Tree Method
  - The Master Method

Example: ~~Stock Market~~ Investment



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97

The Problem: Use the investment graph to determine the days to buy and sell that yield maximum profit (you can buy and sell only one time).

Example: ~~Stock Market~~ Investment

Question: What strategy should you follow (in terms of when to buy and sell) in order to **maximize** your profit?

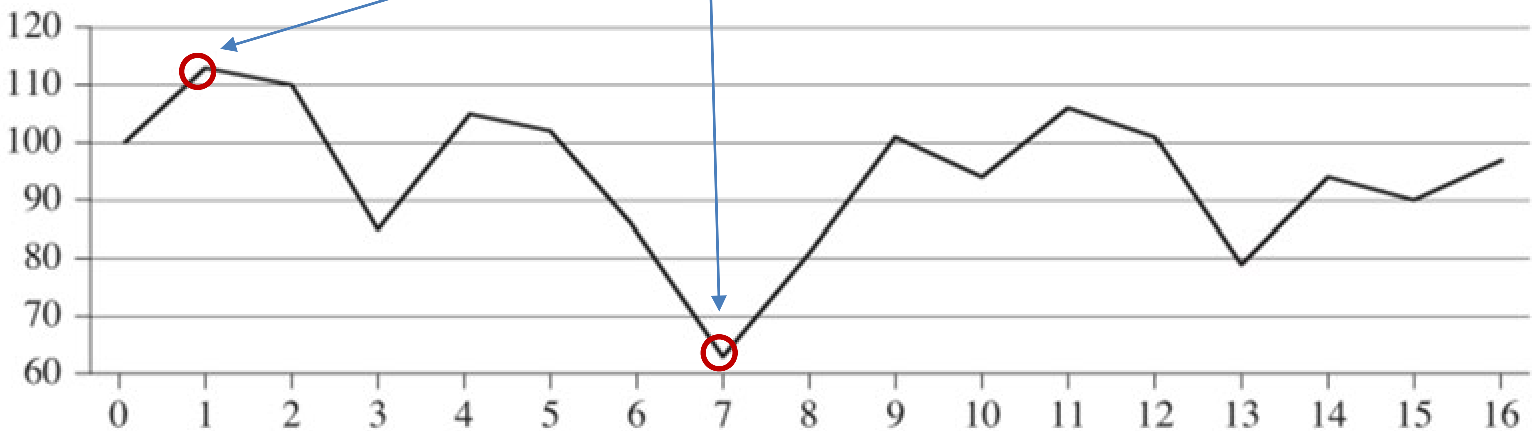
<u>Recall that you can buy and sell only one time.</u>

Example: ~~Stock Market~~ Investment

Strategy#1: To maximize profit, buy at the lowest price and sell at the highest price.

Is it correct: No!

Highest point precedes lowest point

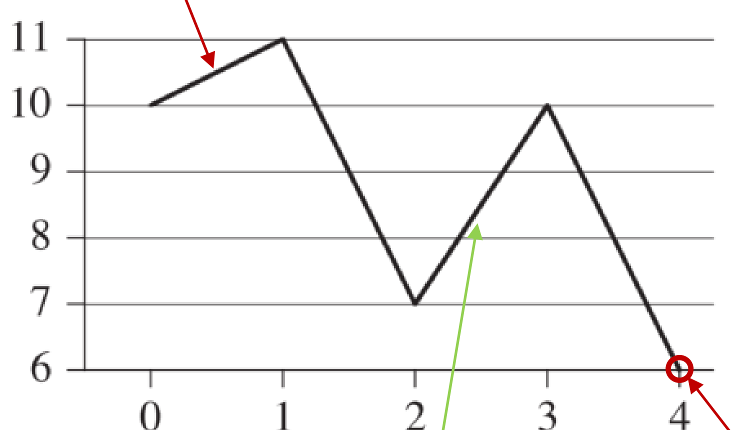


Example: ~~Stock Market~~ Investment

Strategy#2: To maximize profit, either buy at the lowest price or sell at the highest price.

Is it correct: No!

Sell at the highest:
Profit is 1



Best Option:
Profit is 3 ☺

Buy at the lowest:
Profit is undefined (not clear when you can sell and for what price)

Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

Approach#1: Brute-Force

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97

Naïve solution (brute-force): *Try every possible pair of buy and sell dates in which the buy date precedes the sell date.*

InvestmentBruteforce (P)

- $maxVal = 0$
- **for** $i \leftarrow 0$ **to** $n\text{-}1$
  - **for** $j \leftarrow 1$ **to** n
    - **if** $P[i]\text{-}P[j] > maxVal$ **then**
      - $maxVal = P[i]\text{-}P[j]$
      - $buyDay = i$
      - $sellDay = j$
- **Return** $maxVal, buyDay, sellDay$

Problem: $\Theta(n^2)$
(also $\Omega(n^2)$)

Approach#2: Net Change

Better solution: *find a sequence of days over which the net change from the first day to the last is maximum.*

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

The maximum-subarray problem

maximum-subarray: Given array *A*, find the nonempty, contiguous subarray of *A* whose values have the largest sum.

	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

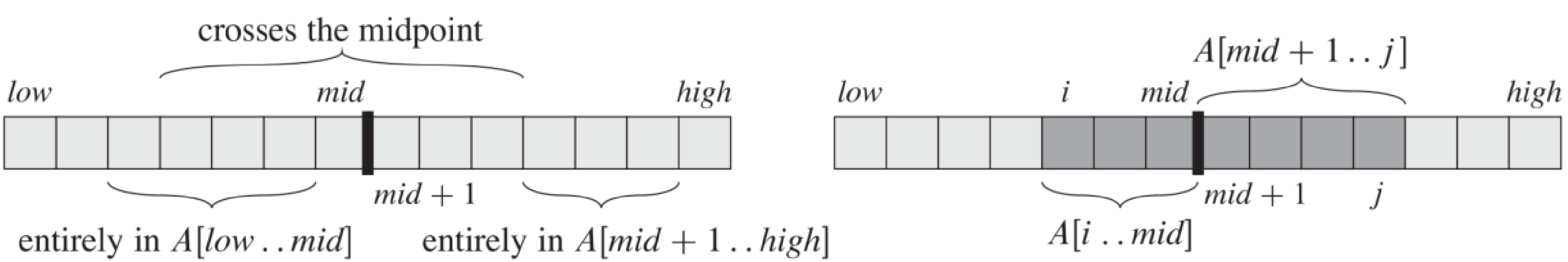
- *Goal: Design a recursive algorithm based on divide-and-conquer to find a maximum-subarray.*

Possible maximum-subarrays

Question: If the array is split into two subarrays, where could the *maximum-subarray* be located?

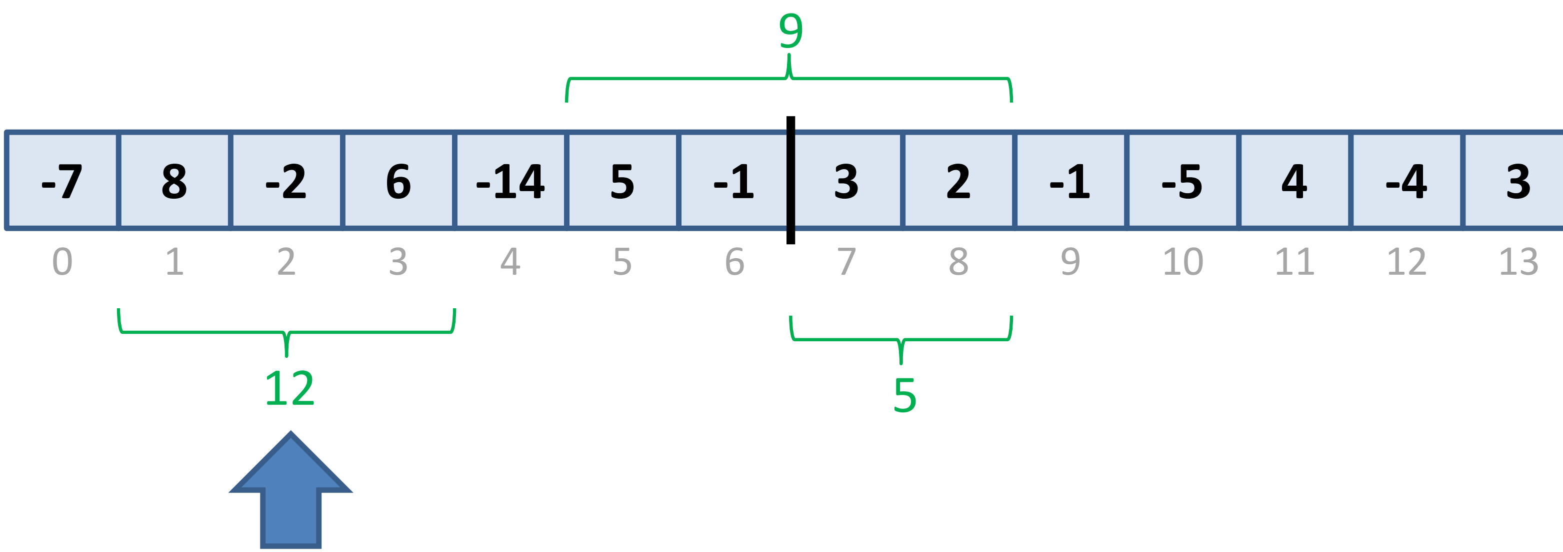
Answer: There are three possibilities:

- In the left subarray
- In the right subarray
- Across the two subarrays



Possible maximum-subarrays: Example

- Maximum-subarray in the left subarray: **12**
- Maximum-subarray in the right subarray: 5
- Maximum-subarray in across the two subarrays: 9



Maximum Subarray
(Index: 1-3**, Total:** 12**)**

FIND-MAX-CROSSING-SUBARRAY Algorithm

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

Complexity: $\Theta(n)$

FIND-MAXIMUM-SUBARRAY Algorithm

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = ⌊(low + high)/2⌋
    (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)
```

The maximum-subarray problem

Question: What does FIND-MAXIMUM-SUBARRAY return when all elements of A are positive?

Answer: The total sum of all numbers in the array.
(why?)

Question: What does FIND-MAXIMUM-SUBARRAY return when all elements of A are negative?

Answer: The largest negative number in the array.

Analyzing the maximum-subarray algorithm

Base case:

$$T(1) = \Theta(1) .$$

Recursive case:

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

Analyzing the maximum-subarray algorithm

Recurrence for FIND-MAXIMUM-SUBARRAY:

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

$\Rightarrow$ $T(n) = \Theta(n \lg n)$ (recall recursion tree for merge sort)

better that $\Theta(n^2)$ *brute-force*

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - **Binary search**
  - Exponentiation
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - The Recursion-Tree Method
  - The Master Method

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a *sorted* array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Recurrence for binary search

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

 # subproblems

subproblem input size

work dividing

(there is no combining)

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

(Master method, which CASE?)

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - **Exponentiation**
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - The Recursion-Tree Method
  - The Master Method

Exponentiation

Problem: Compute a^n, where $n \in N$.

Naive Algorithm:

$$\underbrace{a \cdot a \cdots\cdots a}_{n} \quad \Rightarrow \quad T(n) = \Theta(n).$$

Exponentiation: Divide-and-conquer

Problem: Compute a^n, where $n \in N$.

Divide-and-conquer Algorithm:

1. ***Divide:*** divide n in half.
2. ***Conquer:*** recursively solve two subproblems of $a^{n/2}$.
3. ***Combine:*** multiply the results of the two recursions.

Exponentiation: Divide-and-conquer

Problem: Compute a^n, where $n \in N$.

Recurrence:

$$a^n = \begin{cases} \Theta(1) & \text{If n = 1} \\ a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

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

(Master method, which CASE?)

Exponentiation: Divide-and-conquer

Better Idea: recursively solve <u>one</u> subproblems of $a^{n/2}$.

$$T(n) = 1T(n/2) + \Theta(1) \;\; \Rightarrow T(n) = \Theta(\lg n)$$

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - Exponentiation
  - **Matrix multiplication (Strassen algorithm)**
- Solving Recurrences
  - The Substitution Method
  - The Recursion-tree Method
  - The Master Method

Matrix multiplication

$$\left.\begin{array}{ll}\textbf{Input:} & A = [a_{ij}],\ B = [b_{ij}]. \\ \textbf{Output:} & C = [c_{ij}] = A \cdot B.\end{array}\right\} \quad i, j = 1, 2, \dots, \boldsymbol{n}$$

$$\overset{C}{\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}} = \overset{A}{\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}} \cdot \overset{B}{\begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

First Attempt: Standard algorithm

- Each element in the output matrix C can be computed as follows:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

- We can compute C using the following algorithm:

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Complexity?

$\Theta(n^3)$

Second Attempt: Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix $=$ 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$
\begin{array}{ccccc}
C & & A & & B \\
\left(\begin{array}{cc|cc}
c_{11} & c_{12} & \cdots & c_{1n} \\
 r & & s & \\
c_{21} & c_{22} & \cdots & c_{2n} \\
\hline
\cdots & \cdots & \cdots & \cdots \\
 t & & u & \\
c_{n1} & c_{n2} & \cdots & c_{nn}
\end{array}\right)
& = &
\left(\begin{array}{cc|cc}
a_{11} & a_{12} & \cdots & a_{1n} \\
 a & & b & \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\hline
\cdots & \cdots & \cdots & \cdots \\
 c & & d & \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{array}\right)
& \cdot &
\left(\begin{array}{cc|cc}
b_{11} & b_{12} & \cdots & b_{1n} \\
 e & & f & \\
b_{21} & b_{22} & \cdots & b_{2n} \\
\hline
\cdots & \cdots & \cdots & \cdots \\
 g & & h & \\
b_{n1} & b_{n2} & \cdots & b_{nn}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{ccccc}
\begin{pmatrix} r & s \\ t & u \end{pmatrix} & = & \begin{pmatrix} a & b \\ c & d \end{pmatrix} & \cdot & \begin{pmatrix} e & f \\ g & h \end{pmatrix} \\
C & = & A & \cdot & B
\end{array}
$$

Second Attempt: Divide-and-conquer algorithm

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

$$C \quad = \quad A \quad . \quad B$$

Q: How to compute submatrices *r*, *s*, *t*, *u*?

A: $r = ae + bg;\ \ s = af + bh;\ t = ce + dg;\ u = cf + dh$

Example:

$$\begin{pmatrix} \boxed{\begin{matrix} 96 & 68 \\ 24 & 56 \end{matrix}}_{\,r} & \begin{matrix} 69 & 69 \\ 18 & 52 \end{matrix} \\ \begin{matrix} 58 & 95 \\ 90 & 107 \end{matrix} & \begin{matrix} 71 & 92 \\ 81 & 142 \end{matrix} \end{pmatrix} = \begin{pmatrix} \boxed{\begin{matrix} 5 & 2 \\ 0 & 6 \end{matrix}}_{\,a} & \boxed{\begin{matrix} 6 & 1 \\ 2 & 0 \end{matrix}}_{\,b} \\ \begin{matrix} 3 & 8 \\ 1 & 8 \end{matrix} & \begin{matrix} 1 & 4 \\ 5 & 6 \end{matrix} \end{pmatrix} \times \begin{pmatrix} \boxed{\begin{matrix} 7 & 5 \\ 1 & 8 \end{matrix}}_{\,e} & \begin{matrix} 8 & 0 \\ 2 & 6 \end{matrix} \\ \boxed{\begin{matrix} 9 & 4 \\ 5 & 3 \end{matrix}}_{\,g} & \begin{matrix} 3 & 8 \\ 7 & 9 \end{matrix} \end{pmatrix}$$

Second Attempt: Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

- 8 recursive multiplications of $(n/2) \times (n/2)$ submatrices
- 4 additions of $(n/2) \times (n/2)$ submatrices

Second Attempt: Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \qquad \textbf{(4.9)}$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

SQUARE-MATRIX-MULTIPLY-RECURSIVE algorithm

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

```
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Complexity?

Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; T(n) = \Theta(n^3)$ (Master method, which CASE?)

No better than the ordinary algorithm.

Third Attempt: Strassen's algorithm

- **Recall that:**

$$C \quad = \quad A \quad . \quad B$$

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

Third Attempt: Strassen's algorithm

- **IDEA:** Multiply 2×2 matrices with only 7 recursive mults.

$$P1 = a \times (f - h)$$
$$P2 = (a + b) \times h$$
$$P3 = (c + d) \times e$$
$$P4 = d \times (g - e)$$
$$P_5 = (a + d) \times (e + h)$$
$$P_6 = (b - d) \times (g + h)$$
$$P_7 = (a - c) \times (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 recursive mults, 18 adds/subs.

Note: No reliance on commutativity of mult!

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f\,)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$

Strassen's algorithm

1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

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

Analysis of Strassen algorithm

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

$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow T(n) = \Theta(n^{\lg 7})$ (Master method, which CASE?)

- The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant.

- In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method.
- The divide-and-conquer strategy often leads to efficient algorithms.

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - Exponentiation
  - Matrix multiplication (Strassen algorithm)
- **Solving Recurrences**
  - The Substitution Method
  - The Recursion-Tree Method
  - The Master Method

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - Exponentiation
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - **The Substitution Method**
  - The Recursion-Tree Method
  - The Master Method

The substitution method

The ***substitution method*** for solving recurrences comprises of two steps:

1. Guess the form of the solution (leads to *inductive hypothesis*).
2. Use *mathematical induction* to find the constants and show that the solution works.

- Mathematical induction usually includes:
  - **Basis**
  - **Inductive step**

Example#1:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$

$T(n) = ?$

1. Guess: $n \lg n + n$

2. Inductive Hypothesis:

   $\boldsymbol{T(n) = n \lg n + n}$ for $n \geq n_0$

Basis: (we need to prove our hypothesis in the basis case)

When $n = 1$: $T(n) \stackrel{?}{=} n \lg n + n$

$n = 1 \Rightarrow$ from the recurrence: $T(1) = 1$

$n = 1 \Rightarrow 1 \lg 1 + 1 = 1$

$\rightarrow \quad T(n) \stackrel{\checkmark}{=} n \lg n + n$ when $n = 1$

Inductive Step:

Assuming that our inductive hypothesis

T*(*m*) = *m lg m* + *m is true for <u>all</u> values m where $0<m<n$, we now need to show that our hypothesis is also true when $m = n$.

1. Compute $T(n/2)$ from hypothesis:

   Let $m = n/2 \rightarrow T(n/2) = n/2 \lg n/2 + n/2$

1. Substitute $T(n/2)$ from above in the given recurrence:

$$
\begin{aligned}
T(n) &= 2\,T(n/2) + n \qquad \text{(the given recurrence)} \\
&= 2\,(n/2 \lg n/2 + n/2) + n \\
&= n \lg n/2 + n + n \\
&= n(\lg n - \lg 2) + n + n \\
&= n \lg n - n + n + n \\
&= n \lg n + n \quad \boxed{\text{Done!}}
\end{aligned}
$$

Example#2:

$$T(n)=\begin{cases}1 & n=1\\ 2T\left(\left\lfloor \frac{n}{2}\right\rfloor\right)+n & n>1\end{cases}$$

Upper bound?

1. Guess: $O(n\ lg\ n)$
2. Inductive Hypothesis:

   $T(n) \le cn \lg n$ for $c>0$ and $n \ge n_0$

Basis: $n_0 = 1 \Rightarrow T(1) \le c.1.\lg 1 = 0$

There is no constant c that makes the above inequality true!

Example#2:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\left\lfloor \dfrac{n}{2} \right\rfloor\right) + n & n > 1 \end{cases}$$

Upper bound?

- Recall that:

  $T(n) \leq cn \lg n$ *for* $c>0$ and $n \geq n_0$

- We have:

  $n = 1$ invalid

  $n = 2$?

  $n = 3$?

  $n > 3$ ok (the *recurrence* does not depend directly on $T(1)$)

Example#2:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & n > 1 \end{cases}$$

Upper bound?

- Derive $T(2)$ and $T(3)$ from $T(1)$

  $n = 2 \Rightarrow T(2) = 2\,T(1) + 2 = 2 + 2 = 4$

  $n = 3 \Rightarrow T(3) = 2\,T(1) + 3 = 2 + 3 = 5$

- Use $T(2)$ and $T(3)$ as the base cases in the inductive proof.

Example#2:

$$T(n) = \begin{cases} 1 & n = 1 \\ 4 & n = 2 \\ 5 & n = 3 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & n > 3 \end{cases}$$

Upper bound?

Basis:

- We choose: $n_0 = 2$
- Can we find c such that:

$n = 2$: $T(2)=4 \leq c.2.\lg 2$ and

$n = 3$: $T(3)=5 \leq c.3.\lg 3$

Yes. The above inequalities hold for any $c \geq 2$

Example#2:

$$T(n) = \begin{cases} 1 & n=1 \\ 4 & n=2 \\ 5 & n=3 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & n>3 \end{cases}$$

Upper bound?

Inductive Step: Given the inductive hypothesis:

$T(m) \le cm \lg m$ <u>for all m</u>: $0 < m < n$.

Now, we need to show that the hypothesis is also true for $m = n$.

1. Let: $m = \left\lfloor \frac{n}{2} \right\rfloor \Rightarrow T(\left\lfloor \frac{n}{2} \right\rfloor) \le \underbrace{c\left\lfloor \frac{n}{2} \right\rfloor \lg(\left\lfloor \frac{n}{2} \right\rfloor)}$

2. We substitute into the recurrence:

Example#2:

$$T(n) = \begin{cases} 1 & n = 1 \\ 4 & n = 2 \\ 5 & n = 3 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & n > 3 \end{cases}$$

Upper bound?

Inductive step:

$$T(n) \le 2(c\left\lfloor \frac{n}{2} \right\rfloor \lg(\left\lfloor \frac{n}{2} \right\rfloor)) + n$$

$$\Rightarrow T(n) \le cn\lg(\frac{n}{2}) + n$$

$$= cn\lg n - cn\lg 2 + n$$

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

$$\overset{?}{\le} cn\lg n$$

Yes. This inequality holds for any $c \ge 1$

Making a good guess

- Option#1: Based on ***similarity***.

  Example:
$$\boxed{T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + n & n > 1 \end{cases}}$$

  Argument: when n is large, the difference between $\left\lfloor \frac{n}{2} \right\rfloor$ and $\left\lfloor \frac{n}{2} \right\rfloor + 17$ is not that large.

  → Guess: $O(n\ lg\ n)$

- Option#2: Use ***recursion trees***

Subtleties

- If the guess does not work, you could ***subtract (or add) a lower order term*** and try again!
  Example:

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

Guess: $O(n)$

  - First try: Show that ***T(n) ≤ cn*** *(hypothesis)*

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

Does not guarantee:
T(n) ≤ cn

  - Second try: Show that ***T(n) ≤ cn - d*** where ***d* ≥ 0** is a constant

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

This inequality holds for any c and for any $d \geq 1$

Avoiding Pitfalls

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\left\lfloor \dfrac{n}{2} \right\rfloor\right) + n & n > 1 \end{cases}$$

Upper bound?

1. Guess: $\mathrm{O}(n)$

2. Induction: We need to proof that:

   T(n) ≤ cn **for** ***c>0*** **and** ***n*** **≥** $\boldsymbol{n_0}$

$$\begin{aligned} T(n) &\le 2(c\left\lfloor \frac{n}{2} \right\rfloor) + n \\ &\le cn + n \\ &= \mathrm{O}(n) \end{aligned}$$

false, because we must prove the *exact* form of the inductive hypothesis: ***T(n) ≤ cn***

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - Exponentiation
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - **The Recursion-Tree Method**
  - The Master Method

Recursion-Tree Method

- A recursion tree is best used to generate a *good guess*, which you can then verify by the substitution method.

- In a recursion tree, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

Cost of Recursion-Tree

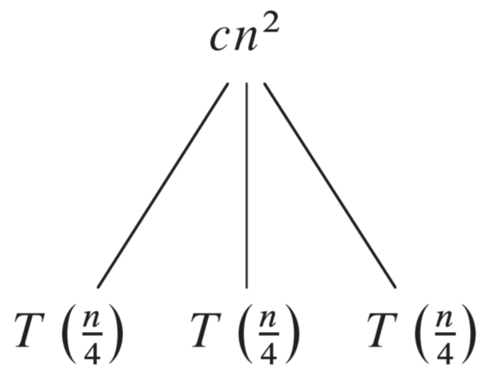
- To determine the cost of a recursion-tree:

    1. Determine the height of the tree.
    2. Sum the costs within each level of the tree to obtain a set of per-level costs.
    3. Sum all the per-level costs to determine the total cost of all levels of the recursion.

$$\boxed{T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)}$$

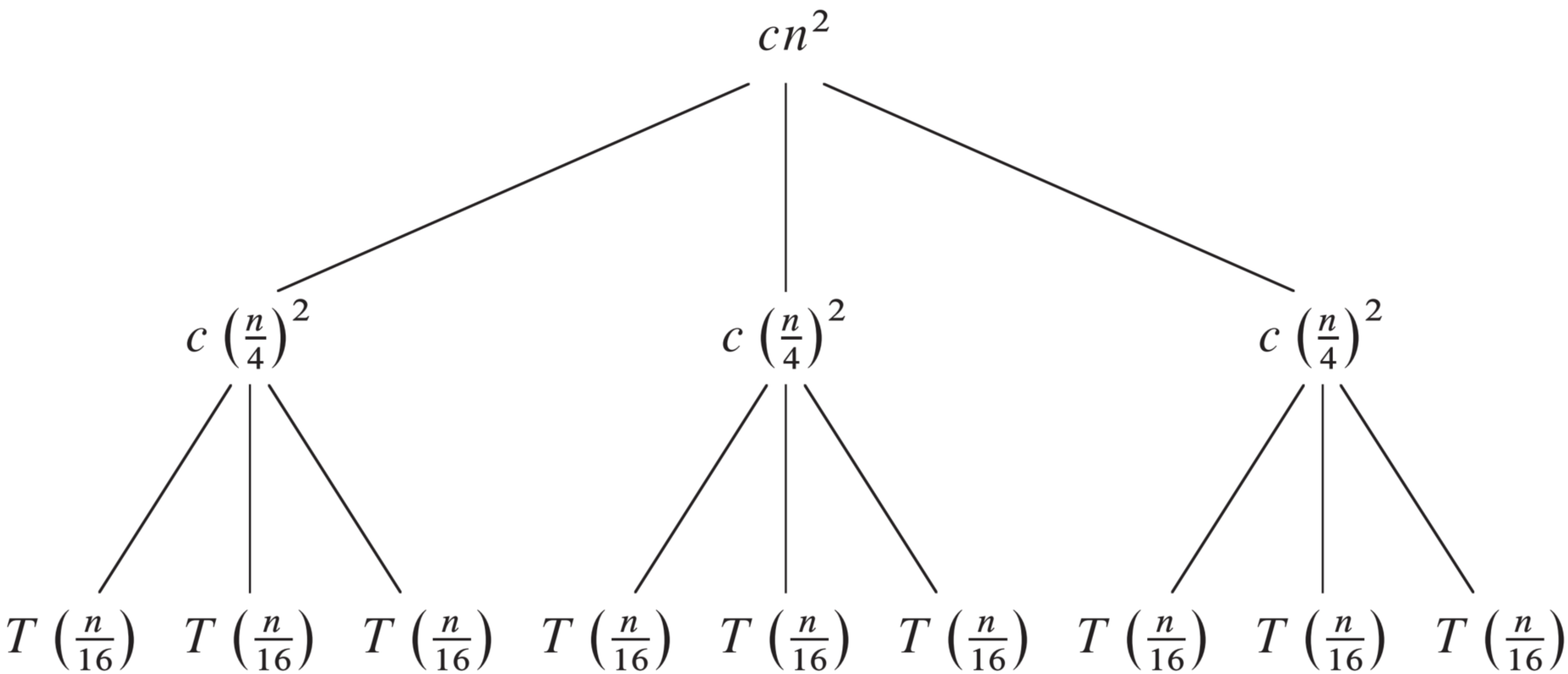
→ We create a recursive tree for: $\boldsymbol{T(n) = 3T(n/4) + cn^2}$

(Remove *floor* function!)



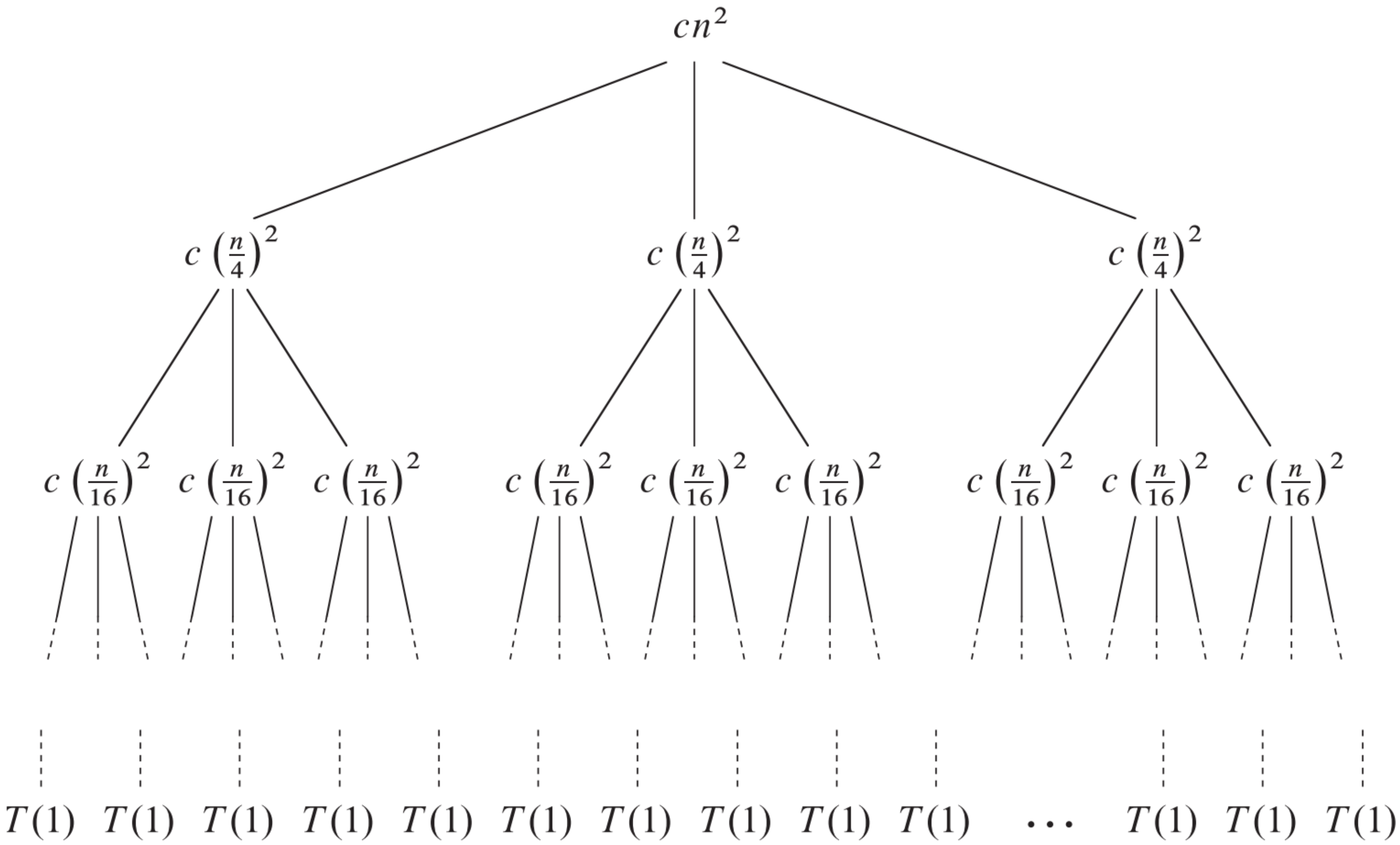
$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $T(n) = 3T(n/4) + cn^2$



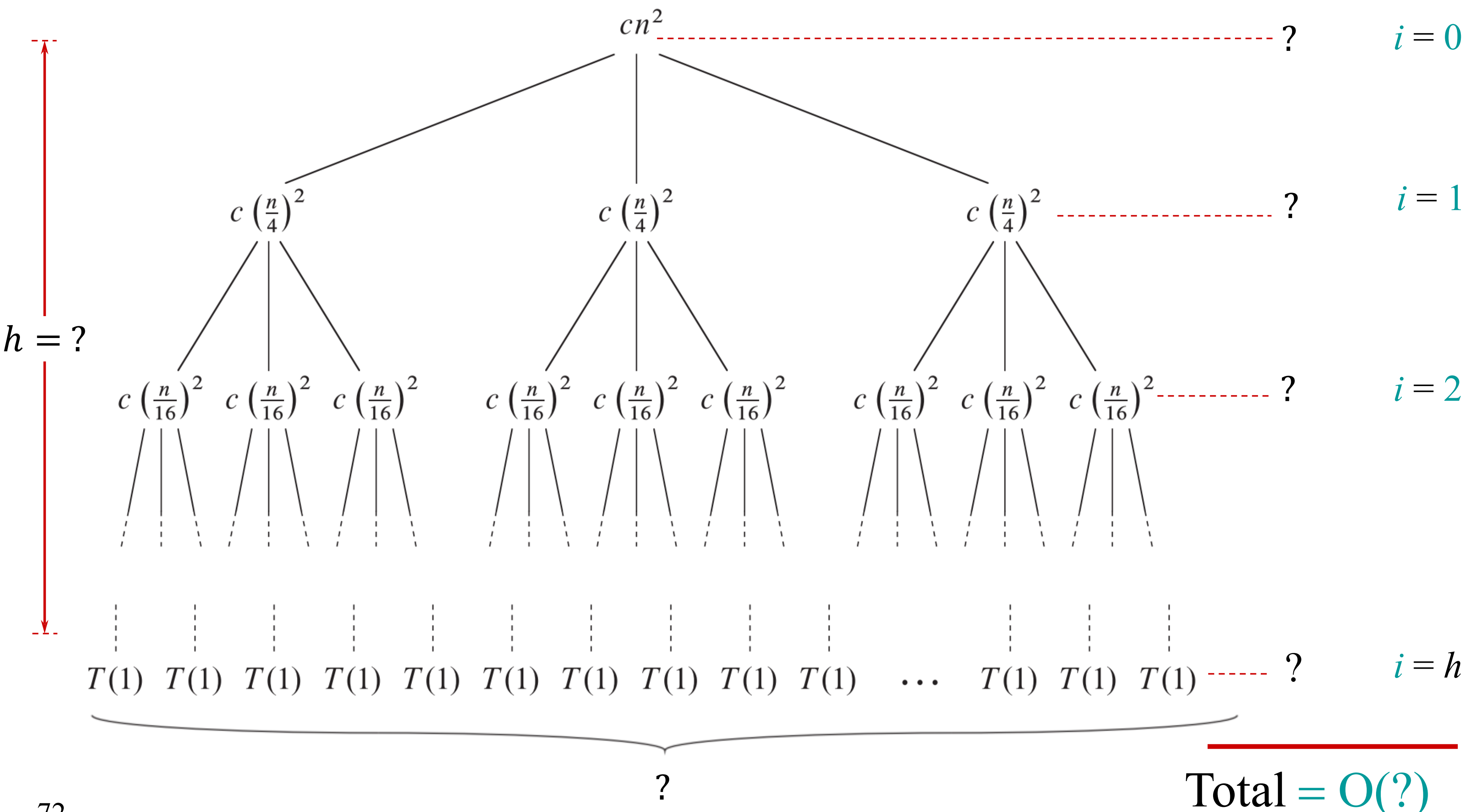
$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $T(n) = 3T(n/4) + cn^2$



$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $\boldsymbol{T(n) = 3T(n/4) + cn^2}$

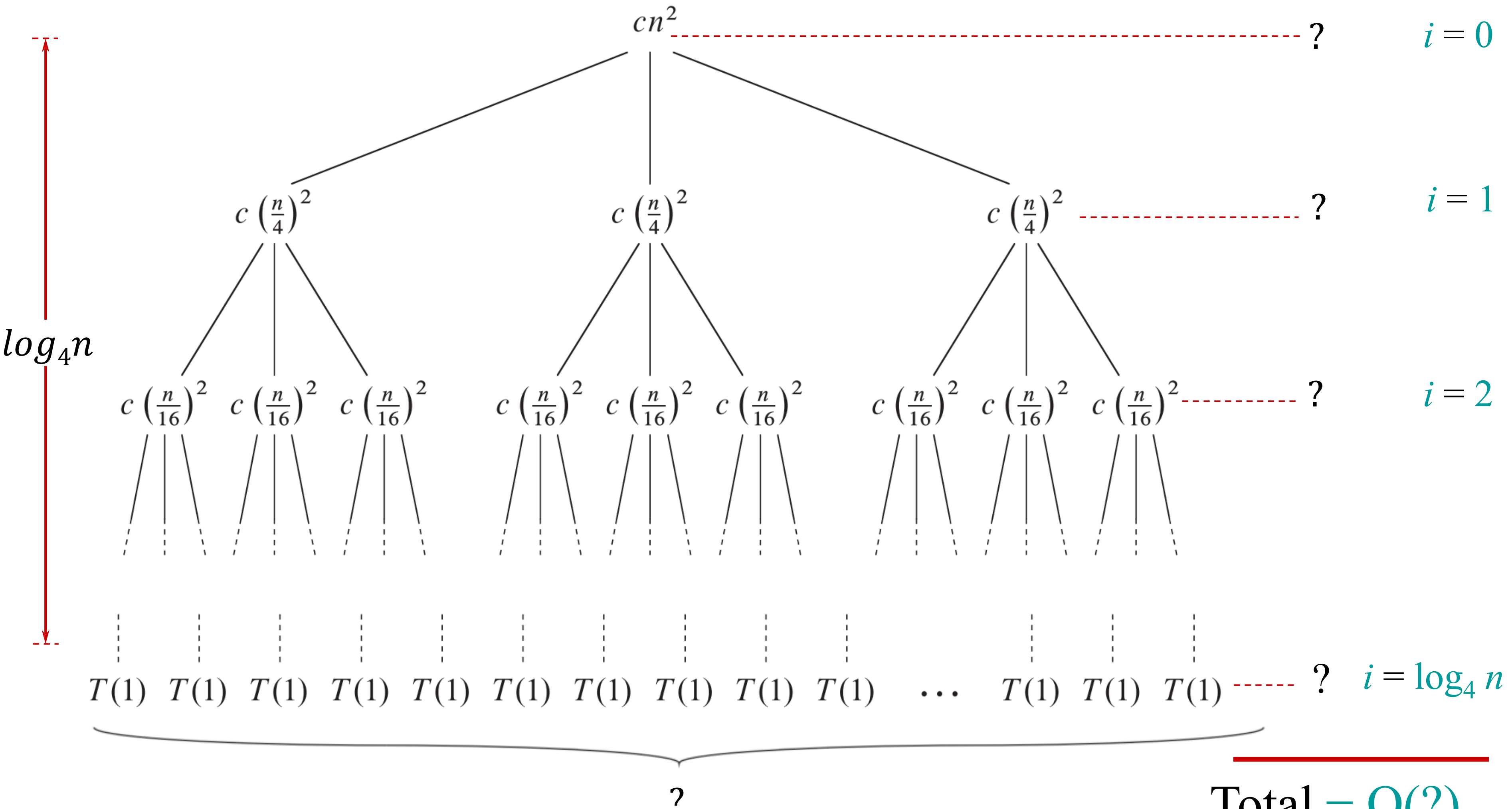


Determining the height of the tree

- The subproblem size for a node at level (depth) i is $n/4^i$
- The subproblem size becomes 1 when: $n/4^i = 1$
- $n = 4^i \rightarrow \log_4 n = \log_4 4^i \rightarrow i = \log_4 n$

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $\boldsymbol{T(n) = 3T(n/4) + cn^2}$

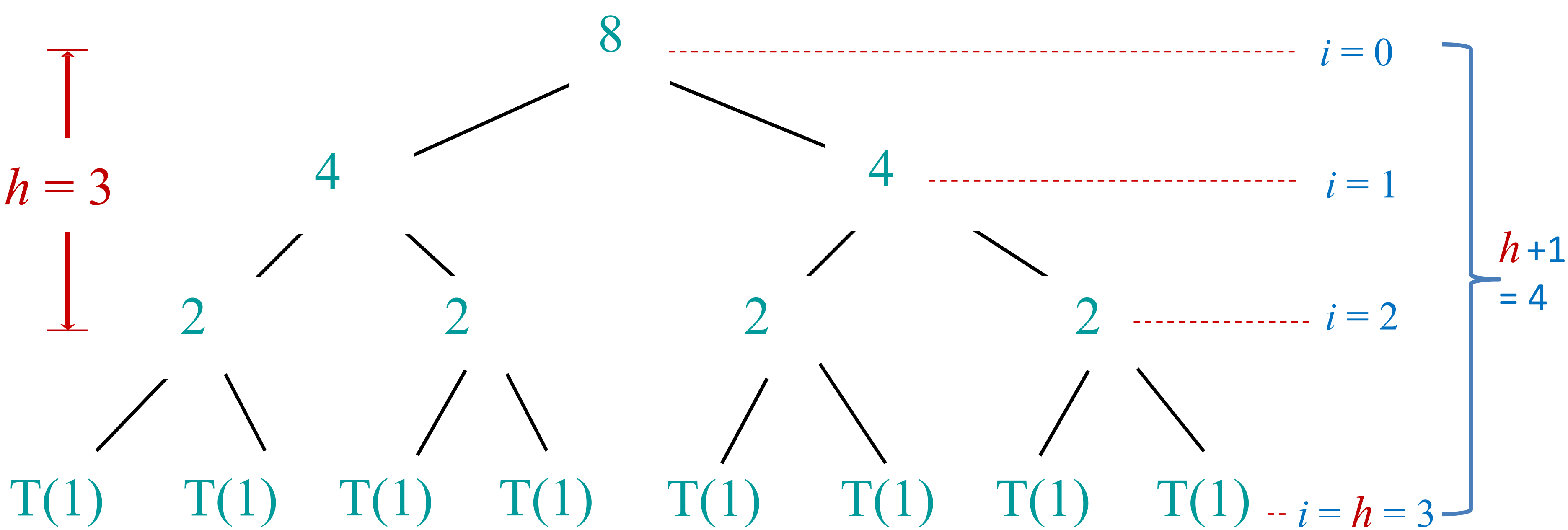


Total = O(?)

Tree Hight and Levels **Example**

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

If $n = 8$, then:



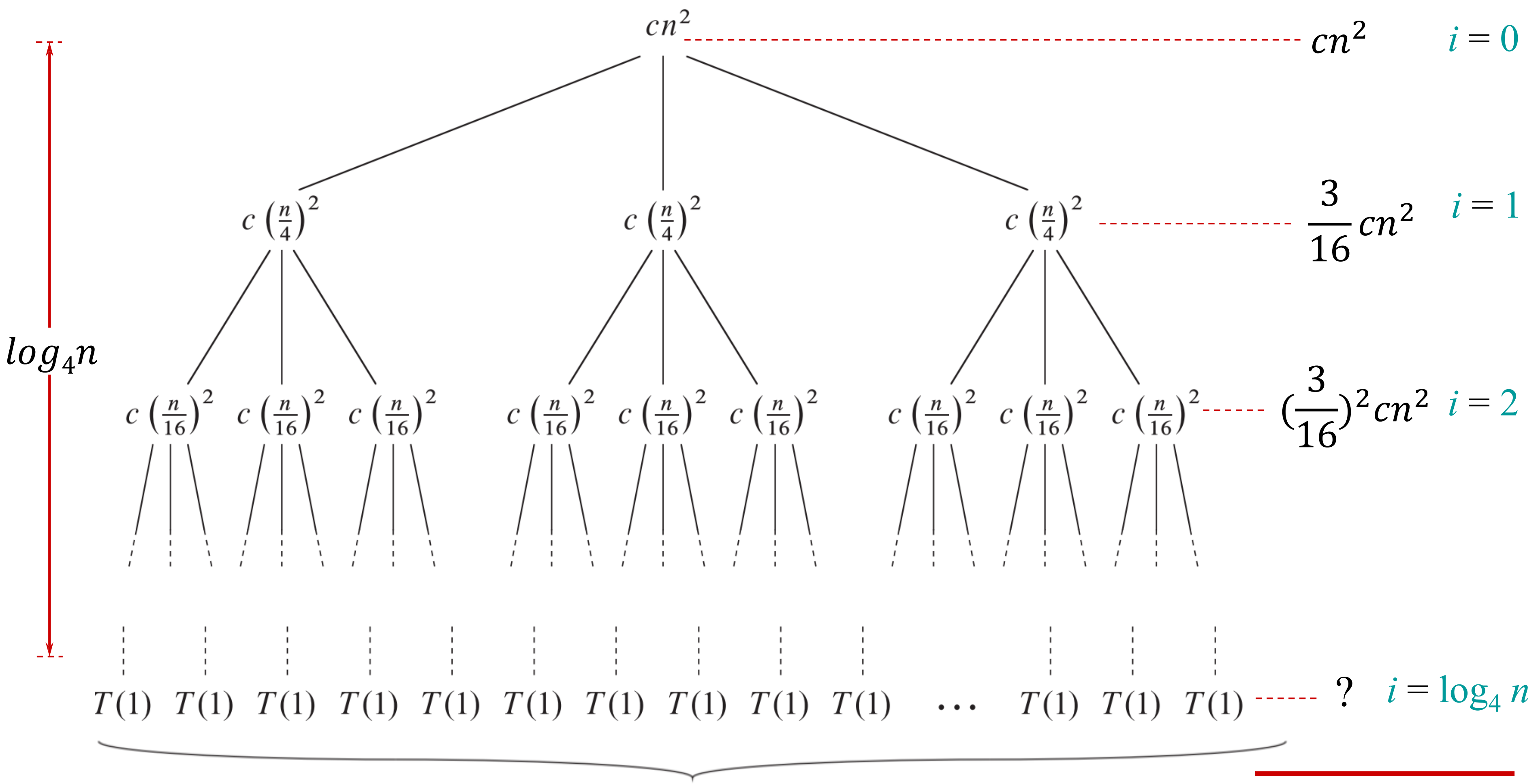
Determining the cost of each level

(except bottom level)

- Each level has three times more nodes than the level above, and so the number of nodes at depth i is 3^i
- Because subproblem sizes reduce by a factor of 4 for each level we go down from the root, each node at depth i, for $i = 0, 1, 2, \ldots, (\log_4 n) - 1$ has a cost of $c(n/4^i)^2$
- Therefore, the total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \log_4 n - 1$ is: $3^i . c(n/4^i)^2 = (3/16)^i . cn^2$

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $\boldsymbol{T(n) = 3T(n/4) + cn^2}$



?

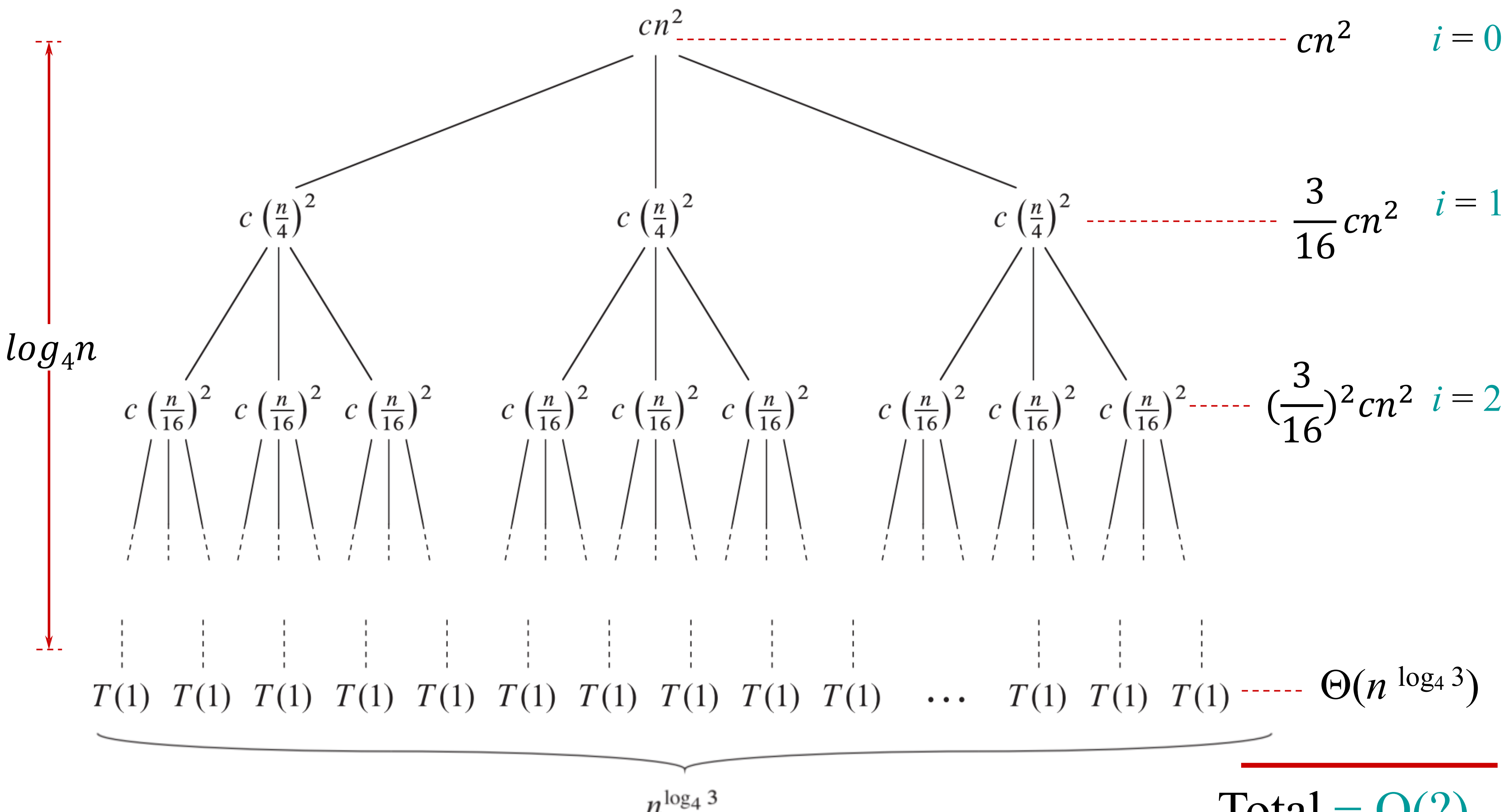
Total = O(?)

Determining the cost of bottom level

- Level: $i = \log_4 n$
- #nodes: $3^{\log_4 n} = n^{\log_4 3}$ (recall that the number of nodes at depth i is 3^i)
- Each node costs $T(1) = k$ (constant)
- → Total cost of bottom level: $k.n^{\log_4 3} = \Theta(n^{\log_4 3})$

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $\boldsymbol{T(n) = 3T(n/4) + cn^2}$



Total = O(?)

Geometric Series

For real $x \neq 1$, the summation

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$$

is a ***geometric*** or ***exponential series*** and has the value

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} . \qquad \text{(A.5)}$$

When the summation is infinite and $|x| < 1$, we have the infinite decreasing geometric series

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} . \qquad \text{(A.6)}$$

Check the course homepage for more useful summation formulas

Total Cost

$(\log_4 n) - 1$

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

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

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

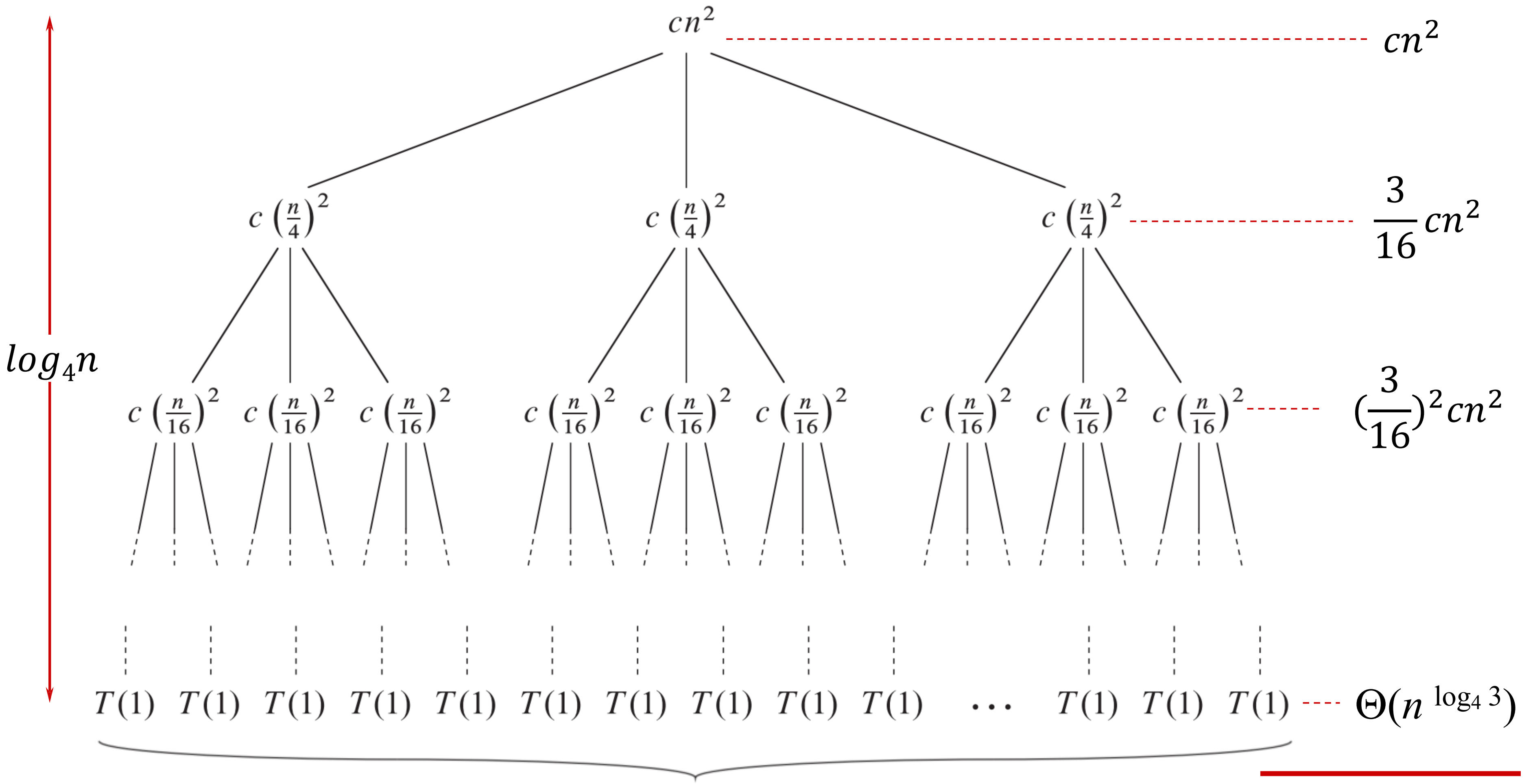
Using A.6

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

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) = O(n^2) .$$ **(our guess)**

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

→ We create a recursive tree for: $T(n) = 3T(n/4) + cn^2$



Total $= O(n^2)$

Verify Using the Substitution Method

$$T(n) = \begin{cases} \Theta(1) & : n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & : n > 1 \end{cases}$$

- Guess: $O(n^2)$
- Hypothesis: $T(n) \leq dn^2$ for some constant $d > 0$

Basis:

- We choose: $n_0 = 1$
- Can we find d such that:

$$n = 1: \quad \underbrace{T(1) = \Theta(1) = k}_{\text{from the recurrence}} \overset{?}{\leq} \underbrace{d.1^2 = d}_{\text{from the hypothesis}}$$

Yes. The inequality holds for any $d \geq k$

Verify Using the Substitution Method

Inductive step:

$$T(n) \le 3T(\left\lfloor \frac{n}{4} \right\rfloor) + cn^2$$

Why? The original recurrence is: $T(n) = 3T(\left\lfloor \frac{n}{4} \right\rfloor) + \Theta(n^2)$

$$\le 3d\left\lfloor \frac{n}{4} \right\rfloor^2 + cn^2$$

Why? substitution

$$\le 3d(\frac{n}{4})^2 + cn^2$$

$$= \frac{3}{16}dn^2 + cn^2$$

$$\stackrel{?}{\le} dn^2$$

This inequality holds for any $d \ge (16/13)c$

$$\boxed{T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + O(n)}$$

$$\Downarrow$$

$$\boxed{T(n) = T(n/3) + T(2n/3) + cn}$$

$$T(n)$$

$$\boxed{T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + O(n)}$$

$$\Downarrow$$

$$\boxed{T(n) = T(n/3) + T(2n/3) + cn}$$

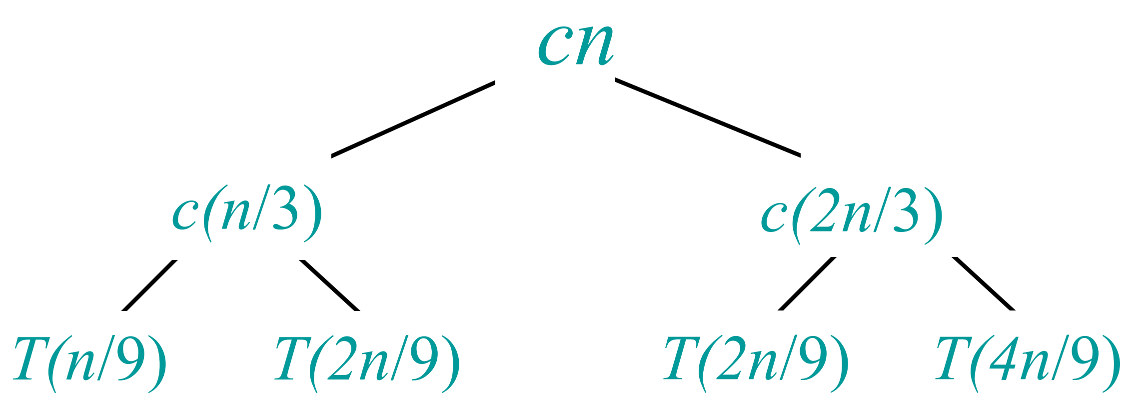
cn

$T(n/3)$ $T(2n/3)$

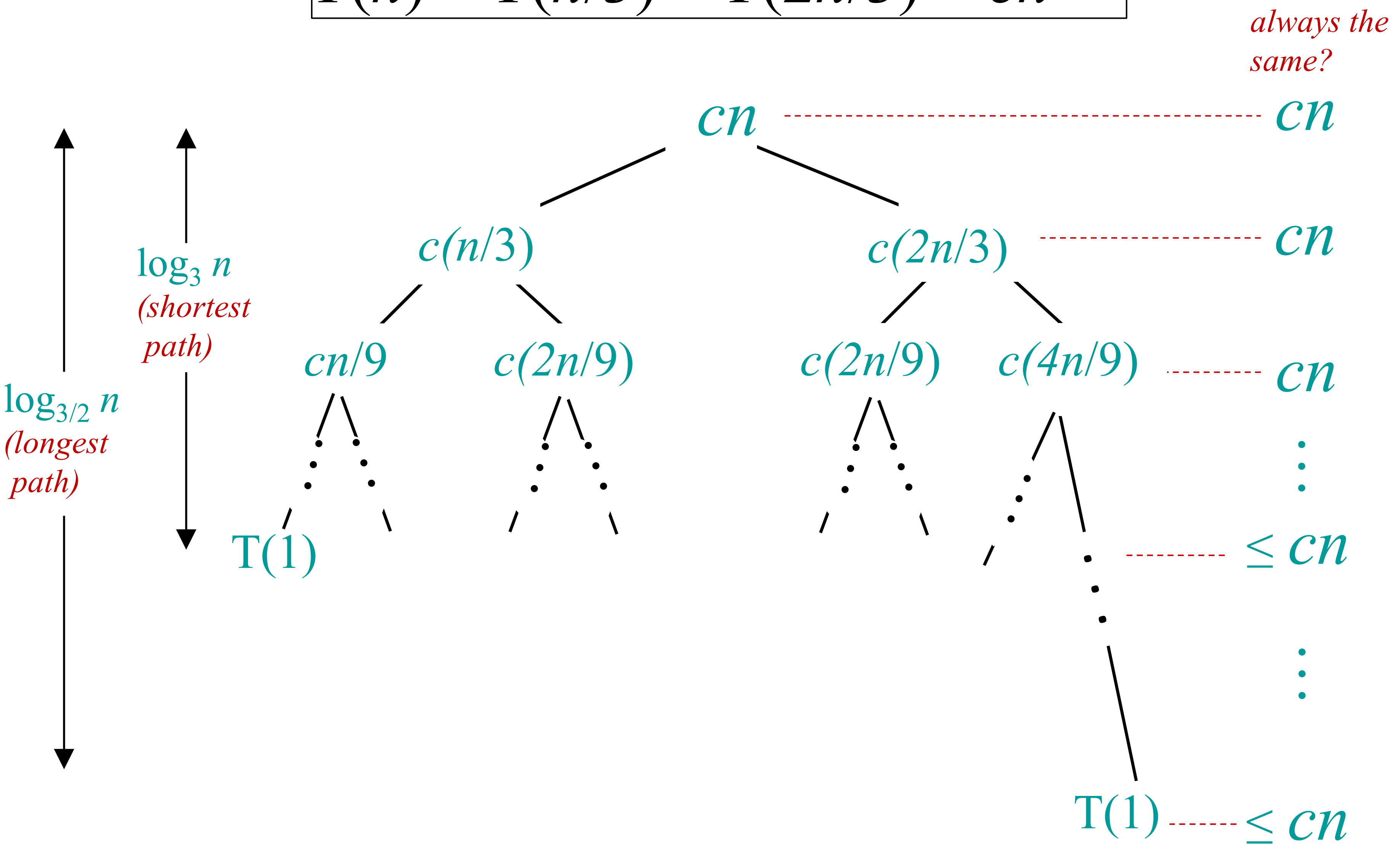
$$T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + O(n)$$

$$\Downarrow$$

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



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



Total $= O(n \lg n)$

$$T(n) = \begin{cases} 1 & : n = 1 \\ T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + cn & : n > 1 \end{cases}$$

1. Guess: $O(n \lg n)$
2. Inductive Hypothesis:

   $T(n) \le dn \lg n$ for $d>0$ and $n \ge n_0$

Basis:

- We choose: $n_0 = 1$
- Can we find d such that:

  $n = 1$: $T(1) = 1 \overset{?}{\le} d.1.\lg 1 = 0$

There is no constant d that makes the above inequality true!

$$T(n) = \begin{cases} 1 & : n = 1 \\ T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + cn & : n > 1 \end{cases}$$

- Recall that:

  $T(n) \le dn \lg n$ *for* $d>0$ and $n \ge n_0$

- We have:

  $n = 1$ invalid

  $n = 2$?

  $n = 3$?

  $n = 4$?

  $n = 5$?

  $n > 5$ ok (the *recurrence* does not depend directly on $T(1)$)

$$T(n) = \begin{cases} 1 & : n = 1 \\ T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + cn & : n > 1 \end{cases}$$

- Derive $T(2)$, $T(3)$, $T(4)$, $T(5)$ from $T(1)$

$$\begin{aligned}
n = 2 \Rightarrow T(2) &= T(0) + T(1) + c.2 = 1 + 2c \\
n = 3 \Rightarrow T(3) &= T(1) + T(2) + c.3 = 2 + 5c \\
n = 4 \Rightarrow T(4) &= T(1) + T(2) + c.4 = 2 + 6c \\
n = 5 \Rightarrow T(5) &= T(1) + T(3) + c.5 = 3 + 10c
\end{aligned}$$

- Use $T(2)$, $T(3)$, $T(4)$, $T(5)$ as the base cases in the inductive proof.

$$T(n) = \begin{cases} 1 & : n = 1 \\ 1 + 2c & : n = 2 \\ 2 + 5c & : n = 3 \\ 2 + 6c & : n = 4 \\ 3 + 10c & : n = 5 \\ T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + cn & : n > 5 \end{cases}$$

Basis:

- We choose: $n_0 = 2$
- Can we find *d* such that:

$$n = 2: \quad T(2) = 1 + 2c \leq d.2.\lg 2 = 2d$$

and $n = 3: \quad T(3) = 2 + 5c \leq d.3.\lg 3 = 3d.\lg 3$

and $n = 4: \quad T(4) = 2 + 6c \leq d.4.\lg 4 = 8d$

and $n = 5: \quad T(5) = 3 + 10c \leq d.5.\lg 5 = 5d.\lg 5$

$$T(n) = \begin{cases} 1 & : n = 1 \\ 1 + 2c & : n = 2 \\ 2 + 5c & : n = 3 \\ 2 + 6c & : n = 4 \\ 3 + 10c & : n = 5 \\ T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + cn & : n > 5 \end{cases}$$

- $n = 2$: $\rightarrow$ $\mathbf{0.5} + c \leq d$
- $n = 3$: $\rightarrow$ $0.42 + \mathbf{1.05}\boldsymbol{c} \leq d$
- $n = 4$: $\rightarrow$ $0.25 + 0.75c \leq d$
- $n = 5$: $\rightarrow$ $0.26 + 0.86c \leq d$

Yes. The above inequalities hold for any $d \geq 1.05c + 0.5$

Verify Using the Substitution Method

Inductive step:

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

This inequality holds for any:
$d \geq c/(\lg 3 - (2/3))$

Content of this Chapter

- Divide and Conquer
  - Maximum-subarray
  - Binary search
  - Exponentiation
  - Matrix multiplication (Strassen algorithm)
- Solving Recurrences
  - The Substitution Method
  - The Recursion-tree Method
  - **The Master Method**

The Master Method

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

Recall that:

- The recurrence above describes the running time of an algorithm that divides a problem of size n into a subproblems, each of size n/b, where a and b are positive constants.
- The subproblems are solved recursively, each in time $T(n/b)$.
- The function $f(n)$ encompasses the cost of dividing the problem and combining the results of the subproblems.

The Master Method

- If $T(n) = aT(n/b) + f(n) : a \geq 1,\ b > 1,\ f(n)$ *is positive* then:

$$T(n) = \left\{ \begin{array}{ll} & \textit{solution} \\ Case1 : \Theta\left(n^{\log_b a}\right) & f(n) \overset{?}{=} O\left(n^{\log_b a - \varepsilon}\right) \\ Case2 : \Theta\left(n^{\log_b a} \lg n\right) & f(n) \overset{?}{=} \Theta\left(n^{\log_b a}\right) \\ Case3 : \Theta\left(f(n)\right) & f(n) \overset{?}{=} \Omega\left(n^{\log_b a + \varepsilon}\right) \text{AND} \\ & \underbrace{af(n/b) \overset{?}{\leq} cf(n) \ \text{ for large } n}_{\textit{regularity condition}} \end{array} \right\} \begin{array}{l} \varepsilon > 0 \\ c < 1 \end{array}$$

Gaps in the Master method

- The three cases do not cover all the possibilities for $f(n)$.
- There is a gap between Case1 and Case2 when $f(n)$ is smaller than $n^{\log_b a}$ but not *polynomially smaller*.
- There is a gap between Case2 and Case3 when $f(n)$ is larger than $n^{\log_b a}$ but not *polynomially larger*.
- If $f(n)$ falls into one of these gaps, or if the regularity condition in Case3 fails to hold, the master method cannot be used to solve the recurrence.

$$\boxed{T(n) = 9T(n/3) + n}$$

- $a=9,\ b=3,\ f(n)=n$
- $n^{\log_b a} = n^{\log_3 9} = n^2 = \mathrm{O}(n^2)$
- Case1: $f(n) = n = \mathrm{O}(n^{2-\varepsilon})$ is satisfied because:
  If $\varepsilon = 1 \Rightarrow n \le cn$ for $1 = c$

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

$$\boxed{T(n) = T(2n/3) + 1}$$

- *a=1, b=3/2, f(n)=1*
- $n^{\log_b a} = n^{\log_{3/2} 1} = 1 = \Theta(1)$
- Case2: $f(n) = n^{\log_{3/2} 1} = \Theta(1)$

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

$$\boxed{T(n) = 3T(n/4) + n \lg n}$$

- $a=3,\ b=4,\ f(n)=n \lg n$
- $n^{\log_b a} = n^{\log_4 3} \approx n^{0.8}$
- Case3: $f(n) = n \lg n = \Omega(n^{\log_4 3+\varepsilon})$ is satisfied when $\varepsilon \approx 0.2$
- We also need to show that: $af(n/b) \le cf(n)$

$$3\frac{n}{4}\lg\frac{n}{4} \le cn \lg n$$

$$\frac{3}{4}\lg\frac{n}{4} \le c \lg n$$

This inequality holds for: $c = 3/4$

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

$$\boxed{T(n) = 2T(n/2) + n \lg n}$$

- $a=2,\ b=2,\ f(n)=n \lg n$
- $n^{\log_b a} = n^{\log_2 2} = n$
- Case3: $f(n) = n \lg n = \Omega(n^{1+\varepsilon})$?
- For any positive constant $\varepsilon, n^{\varepsilon}$ is asymptotically bigger than $\lg n$

→ We cannot use Case3 to solve the recurrence.

This recurrence falls between Case2 and Case3

Conclusion

- The *substitution method* is a powerful tool to solve recurrences, but requires experience and creativity to guess a solution.

- The *recursion-tree method* can be used to derive a guess for the substitution method.

- The *master method* provides ready-to-use solutions, but does not cover all cases.