Divide-and-Conquer

交通大學 資訊工程系

Divide-and-Conquer Paradigm

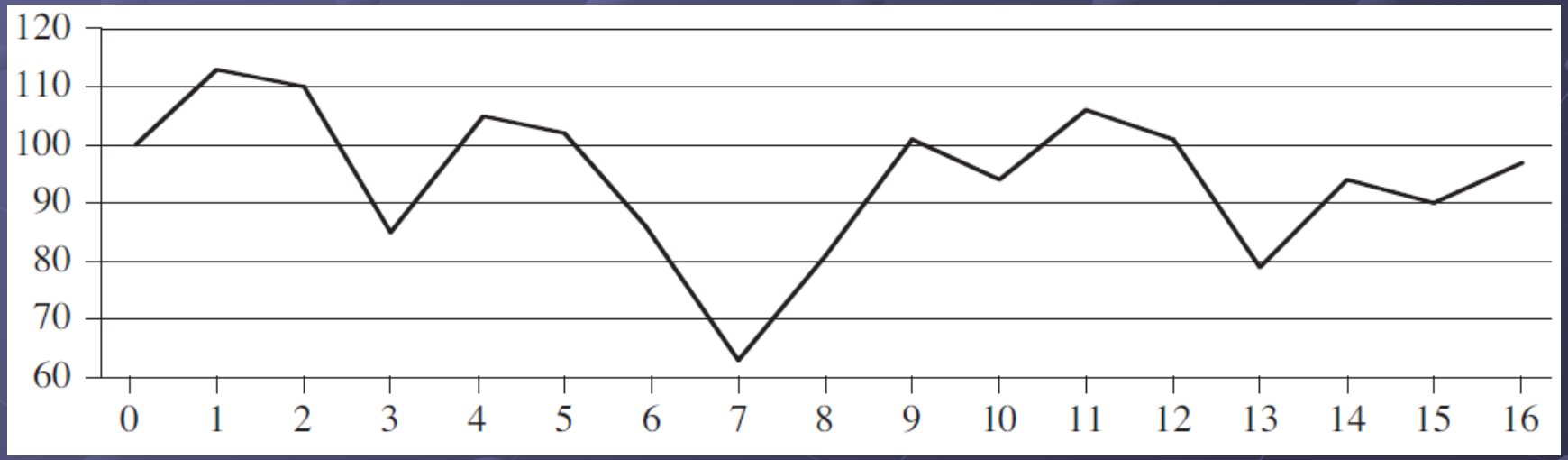
- Applying three steps at each level of the recursion:
  - *Divide* the problem into a number of subproblems.
    - Each subproblem is an instance of the same problem.
  - *Conquer* the subproblems by solving them recursively.
    - If the subproblem sizes are small enough, just solve the subproblems in a straightforward manner.
  - *Combine* the solutions to the subproblems into the solution for the original problem.
- Recursive case
- Base case

Recurrences

- A ***recurrence*** is an equation or inequality that describes a function in terms of its value on smaller inputs.
- Three methods for solving recurrences:
  - Substitution method
  - Recursion-tree method
  - Master method
- Technicalities in recurrences
  - Neglect certain technical details when we state and solve recurrences.
    - For example: Integer arguments to functions, boundary conditions is ignored, omit floors, ceilings.

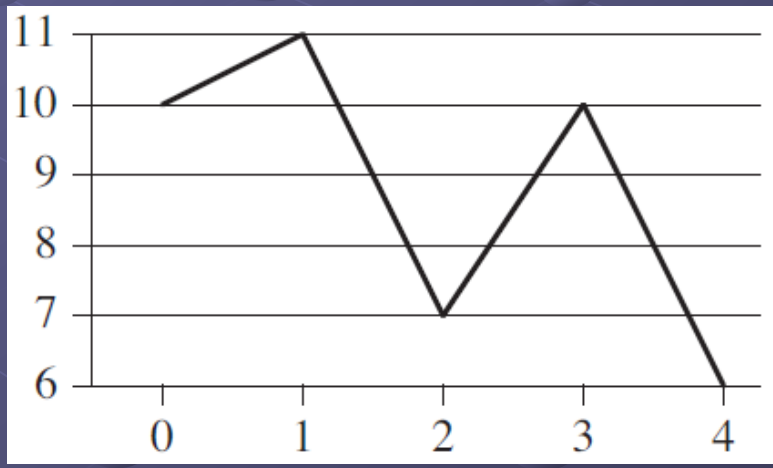
The Maximum-Subarray Problem[1]

- Example: Stock buying and selling
  - To buy one unit of stock only one time and sell it at a later time.
  - Buying and selling after the close of trading for the day.
  - Goal: To maximize your profit.



The Maximum-Subarray Problem[2]

- "*buy low, sell high*"
  - To buy at the lowest price, then sell at the highest price.
  - Either buying at the lowest price or selling at the highest price.
- But the maximum profit does not always start at the lowest price or end at the highest price.

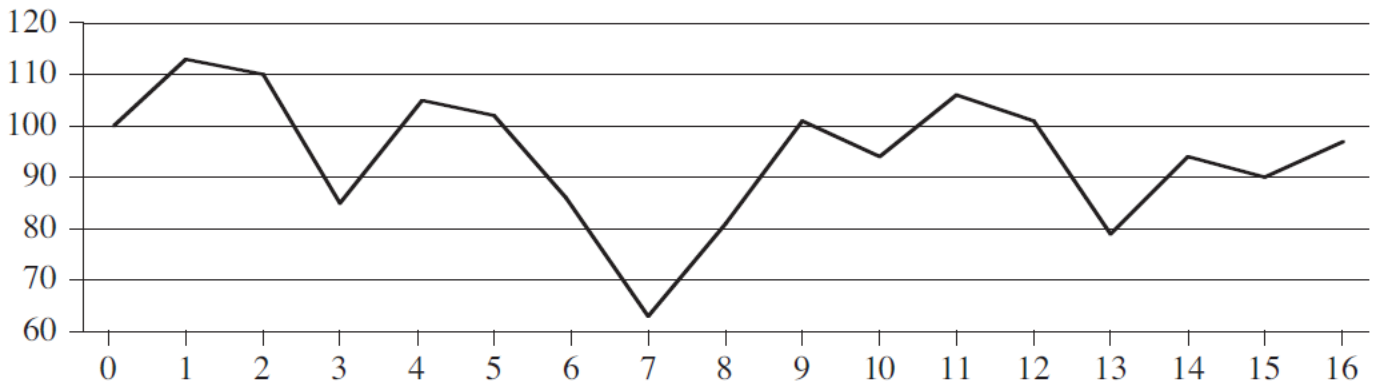


Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

The Maximum-Subarray Problem[3]

- A brute-force solution
  - Try every possible pair of buy and sell dates in which the buy date precedes the sell date.
  - A period of n days has $\binom{n}{2}$ such pairs of dates, which is $\Theta(n^2)$.
- A transformation
  - Look at the input in a slightly different way.
  - To find a sequence of days over which the net change from the first day to the last is maximum.
  - This contiguous subarray is called ***maximum subarray***.

The Maximum-Subarray Problem[4]



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

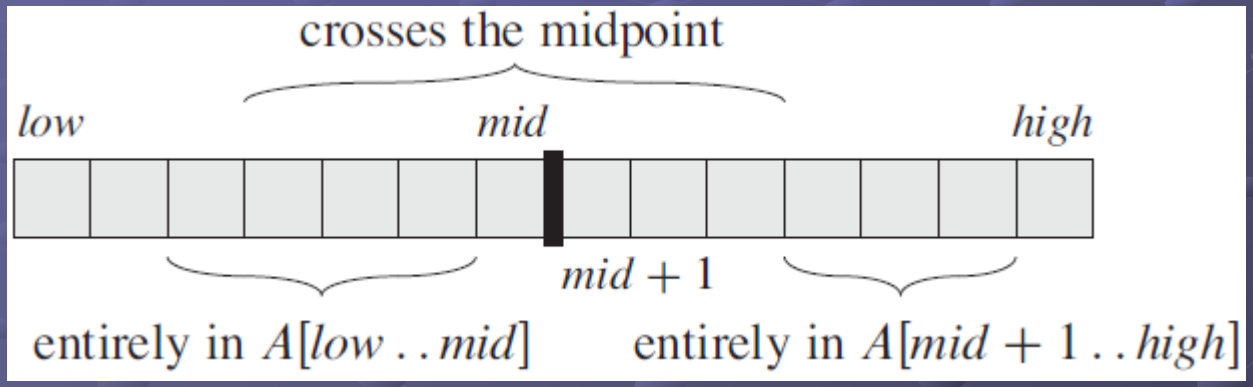
- The maximum-subarray problem is interesting only when the array contains some negative numbers.

	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

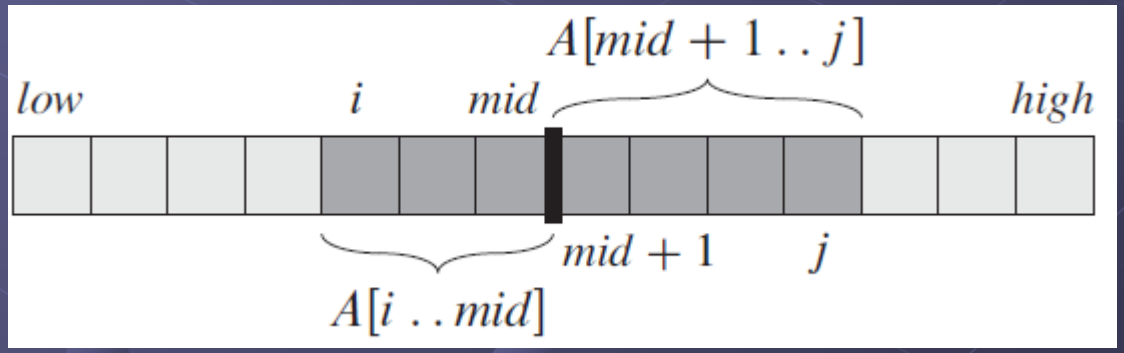
maximum subarray

The Maximum-Subarray Problem[5]

- A solution using divide-and-conquer



- Any subarray of $A[low\,..\,high]$ crossing the midpoint



The Maximum-Subarray Problem[6]

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

The Maximum-Subarray Problem[7]

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

The Maximum-Subarray Problem[8]

- **Analyzing the divide-and-conquer algorithm**

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

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

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

Strassen's Algorithm for Matrix Multiplication[1]

- **Square matrix multiplication**
  - **Let $A = (a_{ij})$ and $B = (b_{ij})$ be $n \times n$ matrices, then the product $C = A \cdot B = (c_{ij})$, we have**

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

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

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

Strassen's Algorithm for Matrix Multiplication[2]

- **A simple divide-and-conquer algorithm**

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

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

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

Strassen's Algorithm for Matrix Multiplication^3

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

Strassen's Algorithm for Matrix Multiplication[4]

- **Analysis**

$$T(n) = \Theta(1) + 8T(n/2) + \Theta(n^2)$$

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

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

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

Strassen's Algorithm for Matrix Multiplication[5]

- Strassen's method
  - Instead of performing eight recursive multiplications of $n/2 \times n/2$ matrices, it performs only seven.
  - Replace one matrix multiplication by several new additions of $n/2 \times n/2$ matrices.
  - It consists of four steps:
    - Divide matrices A, B and C into $n/2 \times n/2$ submatrices. This step takes $\Theta(1)$ time by index calculation.

Strassen's Algorithm for Matrix Multiplication[6]

- **Create 10 matrices S_1, S_2, ⋯, S_{10}, each of which is $n/2 \times n/2$ and is sum or difference of two matrices created in step 1.**

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}\,, \\
S_2 &= A_{11} + A_{12}\,, \\
S_3 &= A_{21} + A_{22}\,, \\
S_4 &= B_{21} - B_{11}\,, \\
S_5 &= A_{11} + A_{22}\,, \\
S_6 &= B_{11} + B_{22}\,, \\
S_7 &= A_{12} - A_{22}\,, \\
S_8 &= B_{21} + B_{22}\,, \\
S_9 &= A_{11} - A_{21}\,, \\
S_{10} &= B_{11} + B_{12}\,.
\end{aligned}
$$

Strassen's Algorithm for Matrix Multiplication[7]

- **Recursively compute seven matrix products P_1, P_2, ⋯, P_7. Each matrix P_i is $n/2 \times n/2$.**

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22}\,, \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22}\,, \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11}\,, \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11}\,, \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}\,, \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}\,, \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}\,.
\end{aligned}
$$

Strassen's Algorithm for Matrix Multiplication[8]

- Compute the desired submatrices C_{11}, C_{12}, C_{21}, and C_{22}.

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

$$\begin{aligned} &A_{11}\cdot B_{11} + A_{11}\cdot B_{22} + A_{22}\cdot B_{11} + A_{22}\cdot B_{22} \\ &\quad - A_{22}\cdot B_{11} + A_{22}\cdot B_{21} \\ &\quad - A_{11}\cdot B_{22} - A_{12}\cdot B_{22} \\ &\quad - A_{22}\cdot B_{22} - A_{22}\cdot B_{21} + A_{12}\cdot B_{22} + A_{12}\cdot B_{21} \\ \hline &A_{11}\cdot B_{11} + A_{12}\cdot B_{21}\,, \end{aligned}$$

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

$$\begin{aligned} &A_{11}\cdot B_{12} - A_{11}\cdot B_{22} \\ &\quad + A_{11}\cdot B_{22} + A_{12}\cdot B_{22} \\ \hline &A_{11}\cdot B_{12} + A_{12}\cdot B_{22}\,, \end{aligned}$$

Strassen's Algorithm for Matrix Multiplication[9]

$$C_{21} = P_3 + P_4$$

$$\begin{array}{l} A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ \qquad - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\ \hline A_{21} \cdot B_{11} \qquad\qquad + A_{22} \cdot B_{21} \ , \end{array}$$

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

$$\begin{array}{l} A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ \qquad - A_{11} \cdot B_{22} \qquad\qquad + A_{11} \cdot B_{12} \\ \qquad\qquad - A_{22} \cdot B_{11} \qquad\qquad - A_{21} \cdot B_{11} \\ - A_{11} \cdot B_{11} \qquad\qquad - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} \\ \hline \qquad\qquad A_{22} \cdot B_{22} \qquad\qquad + A_{21} \cdot B_{12} \ , \end{array}$$

Strassen's Algorithm for Matrix Multiplication[10]

- **Analysis of Strassen's algorithm**
  - **Step 3 requires to perform seven multiplications of $n/2 \times n/2$ matrices.**

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

$$T(n) = \Theta\left(n^{\lg 7}\right)$$

Substitution Method for Solving Recurrences

- For cases which is easy to guess the form of the answer.
- Two steps:
  - Guess the form of the solution.
  - Use mathematical induction to find the constants and show that the solution works. (i.e. Find the constant $c>0$, s.t. $T(n) \le c\, n \lg n$.

$$\begin{cases} T(n) = 2T(\lfloor n/2 \rfloor) + n \\ T(1) = 1 \end{cases}$$

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

Assume $T(\lfloor n/2 \rfloor) \le c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$

Substitution Method

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

Initial $T(1) \le c \quad 1\ lg\ 1 = 0$ (Contraction)
However $4 = T(2) \le c\ 2\ lg\ 2$, if $c \ge 2$.

It is straightforward to extend boundary conditions to make the inductive assumption work for small *n*.

Substitution Method

- **Making a good guess**
  - **There is no general way.**
  - **If a recurrence is similar to one you have seen before, then guessing a similar solution is reasonable.**

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$$

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

  - **To prove loose upper and lower bounds on the recurrence and then reduce the range of uncertainty.**

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \begin{cases} \text{lower bound} & T(n) = \Omega(n) \\ \text{upper bound} & T(n) = O(n^2) \end{cases}$$

Substitution Method

- **Subtleties**
  - **After making a guess, but the math doesn't seem to work out in the induction.**

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

Guess O(n), and try to show T(n) $\le cn$.

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

$$= cn + 1 \quad \textit{does not imply} \quad T(n) \le cn$$

  - **By subtracting a lower-order term often permits the math to go through.**

New Guess T(n) $\le cn - d$, where $d \ge 0$ is a constant.

Substitution Method

$$T(n) \le (c\lceil n/2 \rceil - d) + (c\lceil n/2 \rceil - d) + 1$$
$$= cn - 2d + 1$$
$$\le cn - d$$

- We can prove something stronger for a given value by assuming something stronger for smaller values.

☞ **Avoiding pitfalls**

- It is easy to err in the use of asymptotic notation.

$$T(n) \le 2(c\lfloor n/2 \rfloor) + n$$
$$\le cn + n$$
$$= O(n) \quad \Leftarrow \textit{wrong!!}$$

Substitution Method

- The error is that we haven't proved the exact form of the inductive hypothesis, that is, $T(n) \le cn$.

☞ **Changing variables**

- A little algebraic manipulation can make an unknown recurrence similar to one you have seen before.

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

Renaming $m = \lg n$ yields

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

Rename $S(m) = T(2^m)$

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

The Recursion-Tree Method[1]

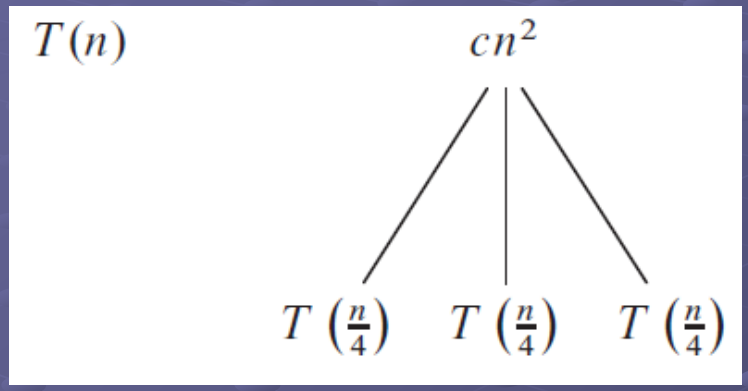
- It is sometimes difficult to come up with a good guess.
  - A recursion tree is best used to generate a good guess.
  - When using recursion tree, you can often tolerate a small amount of "*sloppiness*".
- Example:

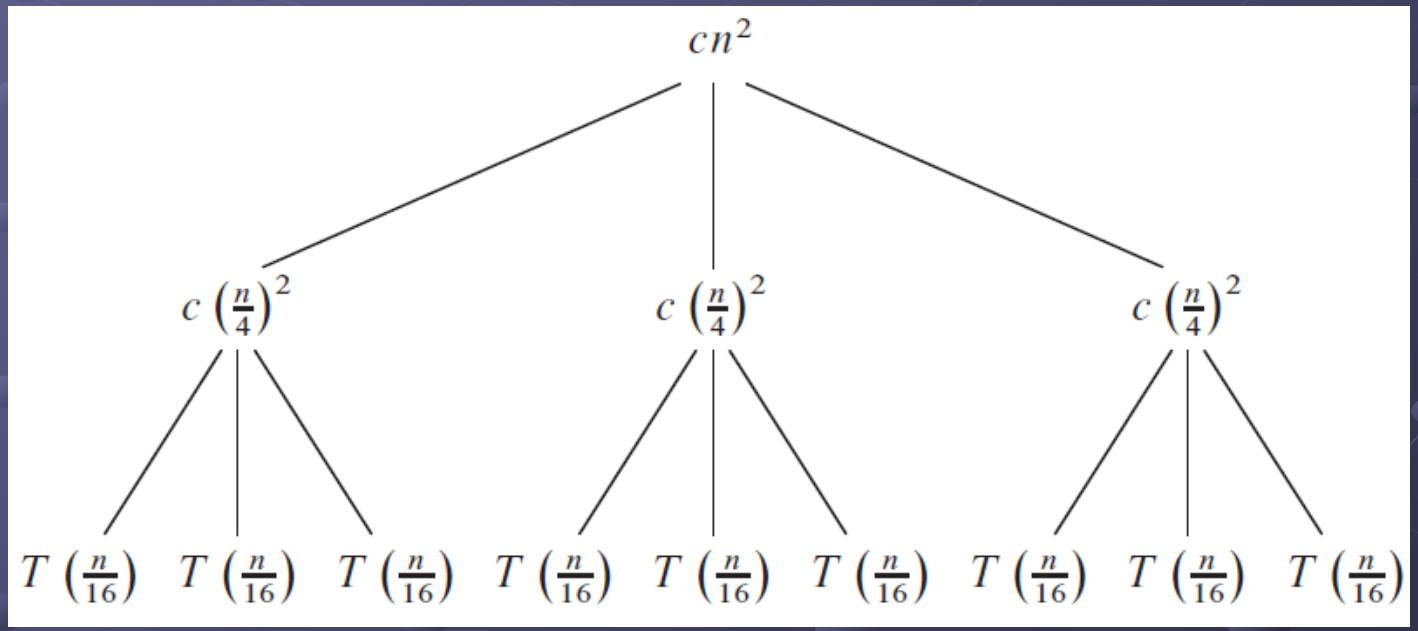
$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

  - Assuming n is an exact power of 4.

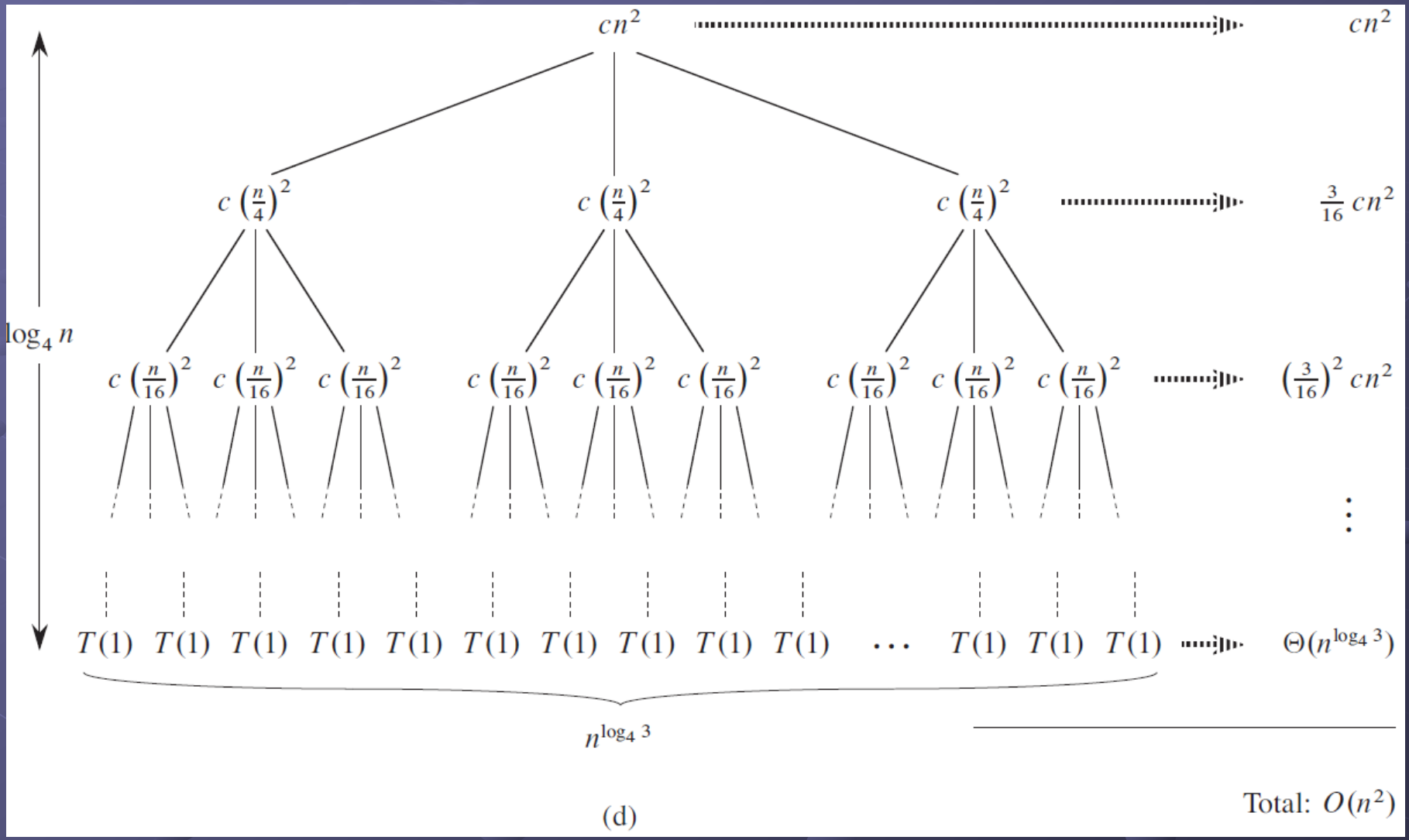
$$T(n) = 3T(n/4) + cn^2$$

The Recursion-Tree Method[2]





The Recursion-Tree Method[3]



The Recursion-Tree Method[4]

- The subproblem size for a node at depth i is $n/4^i$.
  - The subproblem size hits $n = 1$ when $n/4^i = 1$ or $i = log_4 n$.
  - The tree has $log_4 n + 1$ levels.
- Each level has three times more nodes than the level above.
  - The number of nodes at depth i is 3^i.
  - Each node at depth i, for $i = 0, 1, 2, \cdots, log_4 n - 1$, has a cost of $c(n/4^i)^2$.
  - The total cost over all nodes at depth i, for $i = 0, 1, 2, \cdots, log_4 n - 1$, is $3^i c(n/4^i)^2 = (3/16)^i cn^2$.

The Recursion-Tree Method[5]

- The bottom level, at depth $log_4 n$, has $3^{log_4 n} = n^{log_4 3}$ nodes, each contribute cost $T(1)$.
  - For a total cost of $n^{log_4 3} T(1)$, which is $\Theta(n^{log_4 3})$.

$$T(n) =$$

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

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

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

The Recursion-Tree Method[6]

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

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

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

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

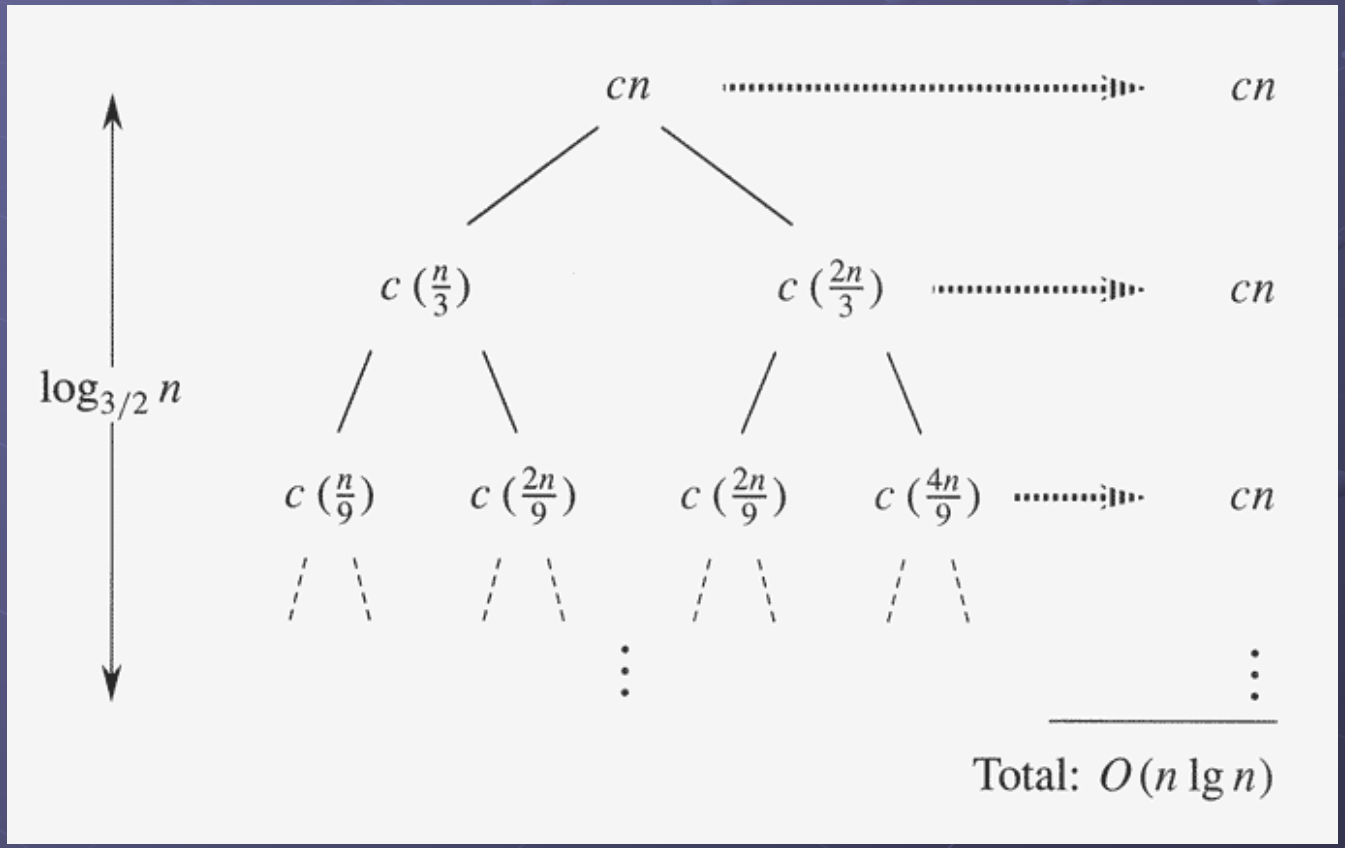
The Recursion-Tree Method[7]

- **Use substitution method to verify the guess.**
  - **Thus, we have derived a guess of T(n) = O(n^2).**

$$
\begin{aligned}
T(n) &\le 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\
&\le 3d(n/4)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\le dn^2, \quad \text{as long as} \quad d \ge (16/13)c.
\end{aligned}
$$

The Recursion-Tree Method[8]

☞ Example: T(n) = T(n/3) + T(2n/3) + O(n).



The Recursion-Tree Method[9]

- Guess O(*n lg n*) as an upper bound.

$$
\begin{aligned}
T(n) &\le T(n/3) + T(2n/3) + cn \\
&\le d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) + (d(2n/3)\lg n \\
&\quad - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2 + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\le dn\lg n, \quad \text{as long as} \quad d \ge c/(\lg 3 - (2/3))
\end{aligned}
$$

The Master Method[1]

- A "cookbook" method for solving recurrences of the following form:

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

  - $a \geq 1$ and $b > 1$ are constants.
  - $f(n)$ is an asymptotically function.
- For example: The Strassen's algorithm
  - $a = 7$, $b = 2$, and $f(n) = \Theta(n^2)$.
- The master method depends on the *Master Theorem*

The Master Method[2]

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 $T(n)$ has the following asymptotic bounds:

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

The Master Method[3]

- Intuitively, comparing the function $f(n)$ with the function $n^{\log_b a}$, the solution to recurrence is determined by the larger of the two functions.
  - In case 1, the function $n^{\log_b a}$ is the larger, then the solution is $T(n) = \Theta(n^{\log_b a})$.
    - *Polynomially smaller*: $f(n)$ also must be asymptotically smaller than $n^{\log_b a}$ by a factor n^{ε}, for $\varepsilon > 0$.
  - In case 3, the function $f(n)$ is the larger, then the solution is $T(n) = \Theta(f(n))$.
    - *Polynomially larger*

The Master Method[4]

- In case 2, the two functions are the same size, we multiply by a logarithmic factor, the solution is $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$.

☞ The three cases do not cover all the possibilities for $f(n)$.

- Gap between cases 1 and 2
  - $f(n)$ is smaller than $n^{\log_b a}$ but not polynomially smaller.
- Gap between cases 2 and 3
  - $f(n)$ is larger than $n^{\log_b a}$ but not polynomially larger.

The Master Method[5]

- The master method cannot be used to solve the recurrence, if
  - $f(n)$ falls into one of the above gaps.
  - The regularity condition in case 3 fails to hold.
    - Regularity condition:

$$af(n/b) \leq cf(n)$$

The Master Method[6]

- **Examples:**
  - $T(n) = 9T(n/3) + n$.

$$a = 9, \quad b = 3, \quad f(n) = n \quad \Rightarrow \quad n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$$

$$f(n) = O(n^{\log_3 9 - \varepsilon}), \text{ where } \varepsilon = 1 \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

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

$$a = 1, b = 3/2, \ f(n) = 1 \quad \Rightarrow \quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

$$f(n) = \Theta(n^{\log_b a}) = \Theta(1) \quad \Rightarrow \quad T(n) = \Theta(\lg n).$$

The Master Method[7]

- $T(n) = 3T(n/4) + n \lg n$.

$$a = 3, b = 4, f(n) = n \lg n \quad \Rightarrow \quad n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

$$f(n) = \Omega(n^{\log_4 3 + \varepsilon}), \text{ where } \varepsilon \approx 0.2.$$

Check

$$af(n/b) = 3(n/4)\lg(n/4) \le (3/4)n \lg n = cf(n)$$

$$\text{for} \quad c = 3/4, \text{ and sufficiently large } n$$

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

The Master Method[8]

- The master method does not apply to the recurrence $T(n) = 2T(n/2) + n \lg n$.

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

  - It might seem that case 3 should apply, since $f(n) = n \lg n$ is asymptotic larger than $n^{\log_b a} = n$.
  - It is not polynomially larger.