4.Divide-and-Conquer

中國文化大學
資訊工程學系
副教授　張耀鴻
112年度第2學期

4.Divide-and-Conquer

1. Divide-and-conquer (各個擊破法)
   - ✓ Divide, Conquer, Combine (分開, 征服, 合併)
2. Recurrence Problem (遞迴問題)
   - ✓ Problem with one or more base cases, and
   - ✓ Itself, with smaller arguments
3. Maximum-subarray problem (最大子陣列問題)
   - ✓ Find-Max-Crossing-Subarray(…)
   - ✓ Find-Maximum-Subarray(…)
4. Strassen’s matrix multiplication (Strassen 矩陣相乘問題)
   - ✓ Rec-Mat-Mult(…)
   - ✓ Strassen’s algorithm
5. Substitution method (替代法)
6. Recursion tree method (遞迴樹法)
7. The Master Theorem (大師法)

Divide-and-Conquer Paradigm

- **Divide** the problem into a number of subproblems that are smaller instances of the same problem. 分割
- **Conquer** the subproblems by solving them recursively. 征服
  - ***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. 合併
- We look at two more algorithms based on divide-and-conquer. 最大子陣列問題, 矩陣相乘

Analyzing divide-and-conquer algorithms 各個擊破法之分析

- Use a recurrence to characterize the running time of a divide-and-conquer algorithm.
- Solving the recurrence gives us the asymptotic running time.
- A ***recurrence*** is a function is defined in terms of
  - one or more base cases, and
  - itself, with smaller arguments.

遞迴函式: 在函式中呼叫並傳遞較小的參數給自已.

各個擊破法適合以遞迴函式觀念來分析.

將欲分析的問題以遞迴式列出, 可快速得知時間複雜度.

Example 1

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

Example 2

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

Example 3

$$T(n) = \begin{cases} 1 & \text{if } n = 1 , \\ T(n/3) + T(2n/3) + n & \text{if } n > 1 . \end{cases}$$

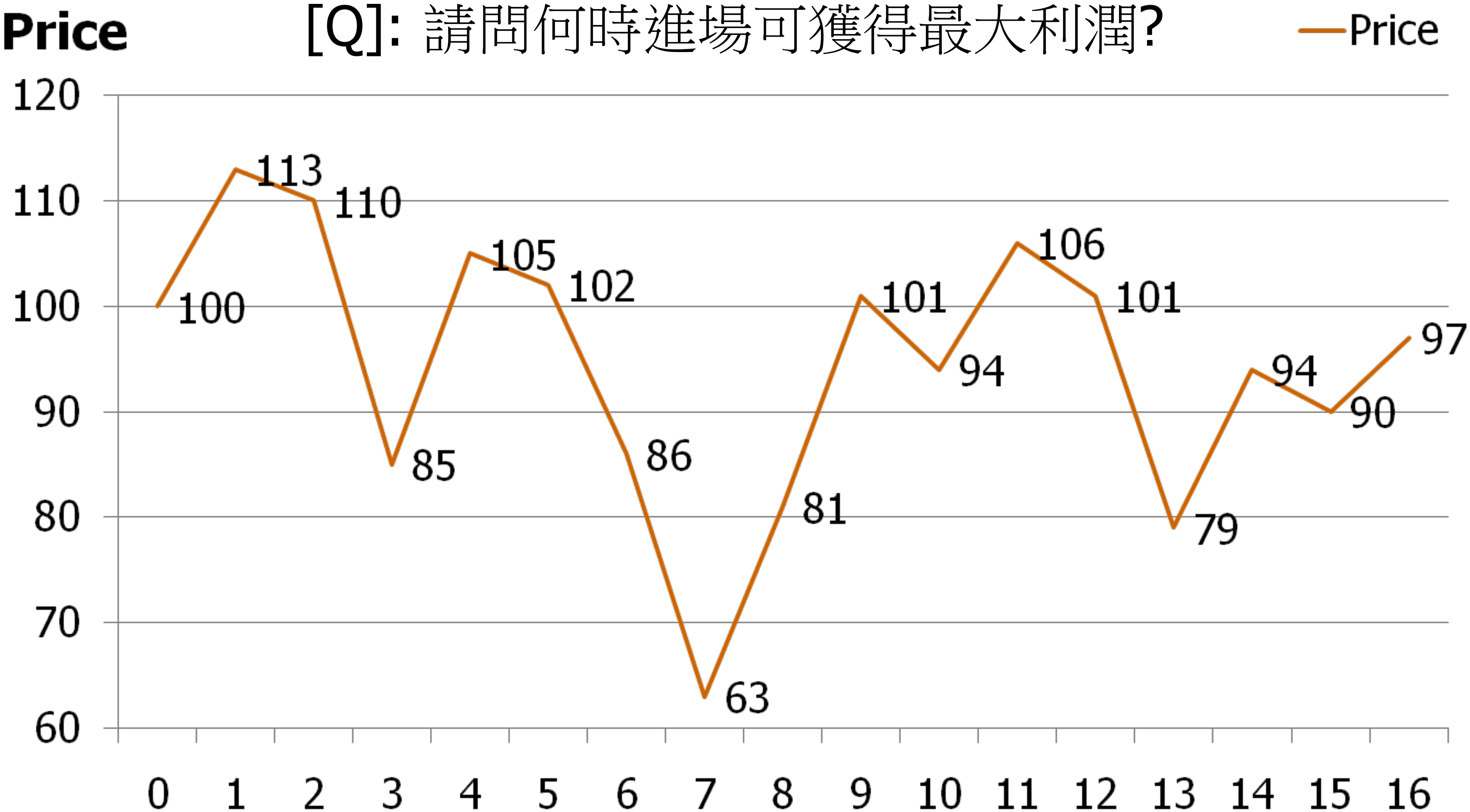
遞迴式
Recurrences -- $T(n) = aT(n/b) + f(n)$

- *Substitution method* 替身法
- *Recursion-tree method* 遞迴樹法
- *Master method* 大師法

最大子陣列問題

4.1 Maximum-subarray problem

[Q]: 請問何時進場可獲得最大利潤?



Scenario

- Allowed to buy one unit of stock at one time
- Sell it at later date
- Buy/Sell after the close of trading day
- **Allowed to learn the price in the future**
- **Goal: Maximize your profit**
  - Strategy: Buy low, sell high
  - Problem: May not be able to implement strategy
    - Highest: Day 1
    - Lowest: Day 7

1. 一次只能買一張股票
2. 只能作多，不能作空
3. 只能在收盤價作買賣
4. 假設未來股價可預測

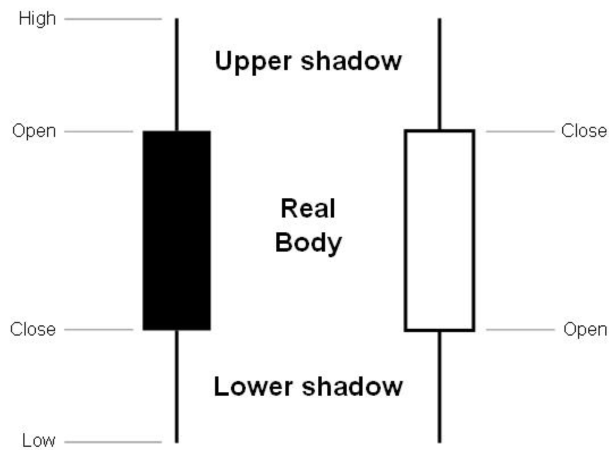
目標: 求得最大利潤

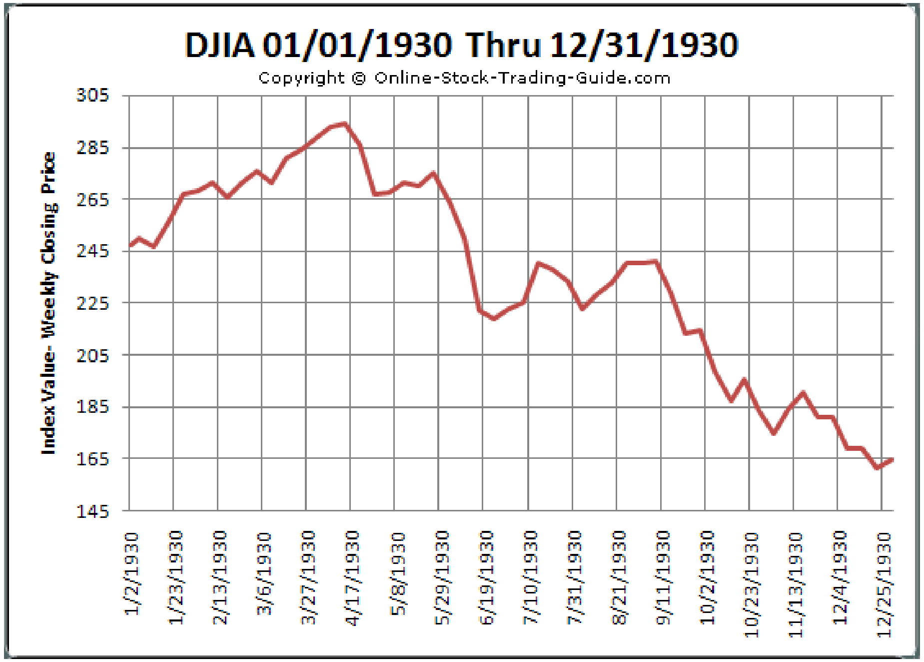
Brute-force solution 暴力破解法

- Check every possible pair of buy and sell dates 比較所有可能的買賣時間點

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \Theta(n^2)$$

- Can we do better?



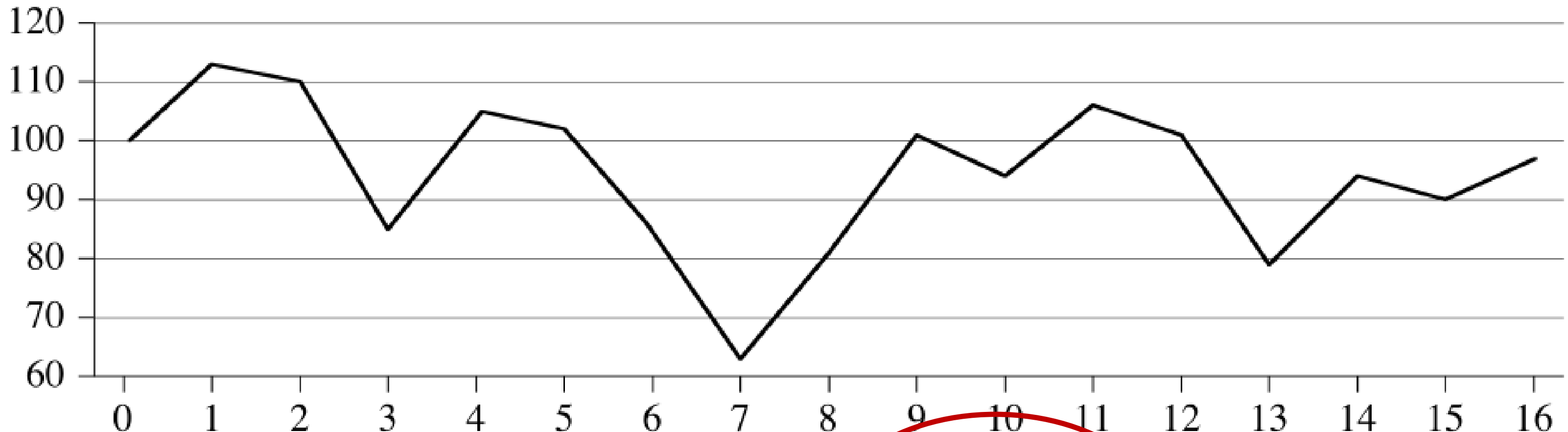


Convert to a maximum-subarray problem

今天收盤價 – 昨天收盤價 (每天虧損金額)

$$A[i] = (\text{price after day } i) - (\text{price after day } (i-1))$$

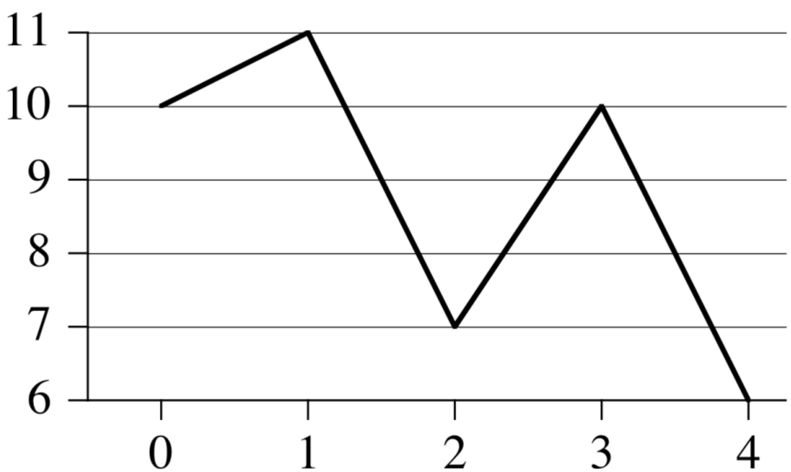
- Input: Array A[1..n] of numbers 輸入陣列A
- Output: Indices i, j and sum of A[i..j], s.t. A[i..j] has the greatest sum 輸出陣列索引值i,j, 使得A[i..j]總合最大



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

Example

- 最大利潤不見得一定是買在最低，賣在最高。

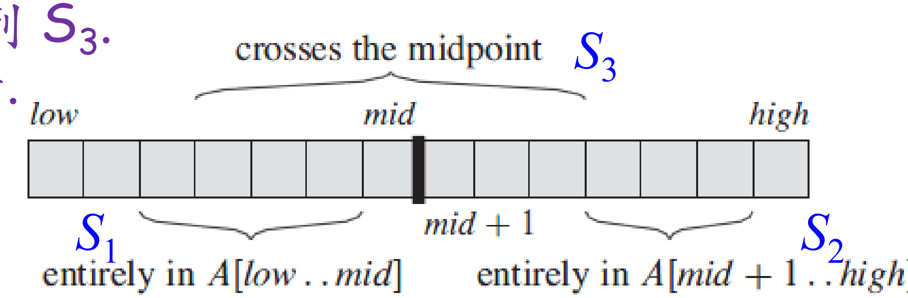


Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

Solving by Divide-and-Conquer

- Subproblem: Find a **max subarray** of A[low..high]
  - Initially: low = 1, high = n
    子問題: 找出最大子陣列A[low..high], 一開始為整個陣列

- **Divide** the subarray into two subarrays of as equal size as possible. 分開: 將陣列一分為二: *A[low..mid]* and *A[mid+1..high]*
  - Find the midpoint *mid* of the subarrays, and consider the subarrays *A[low..mid]* and *A[mid+1..high]*
- **Conquer** by finding a maximum subarrays of *A[low ..mid]* and *A[mid+1..high]* 征服: 分別找出每一段的最大子陣列, S_1, S_2 .
- **Combine** by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three
  組合: 再找出跨越中點的最大子陣列 S_3.
  最後選 S_1, S_2 , S_3 中最大者.



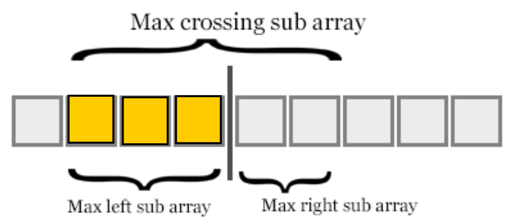
Max subarray that crosses the midpoint 找出跨越中點的最大子陣列

Running time: $\Theta(n)$

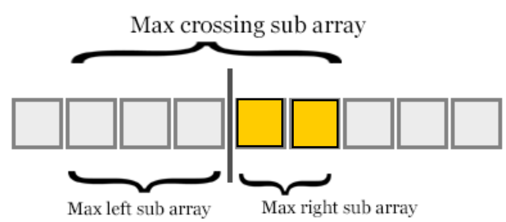
FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$

```
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)
```

從中點開始往左找出含中點之前的最大子陣列



從中點開始往右找出含中點之後的最大子陣列



兩者相加後回傳

Divide-and-conquer for the maximum-subarray problem

```
T(n)      FIND-MAXIMUM-SUBARRAY(A, low, high)
              if high == low
                  return (low, high, A[low])        // base case: only one element
              else mid = ⌊(low + high)/2⌋
T(n/2)            (left-low, left-high, left-sum) =
                      FIND-MAXIMUM-SUBARRAY(A, low, mid)
T(n/2)            (right-low, right-high, right-sum) =
                      FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
O(n)              (cross-low, cross-high, cross-sum) =
                      FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
O(1)              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)
```

Analyzing FIND-MAXIMUM-SUBARRAY

- Simplified assumption: Problem size is power of 2 為簡單起見, 假設輸入大小為2的次方
- Let T(n) denotes the running time on n elements T(n) 表示輸入大小為n個數的執行時間.
- **Base case**: high = low → n = 1 → $T(n) = \Theta(1)$

Analyzing FIND-MAXIMUM-SUBARRAY

- **Recursive case**: n > 1
  - **Dividing** takes Θ(1) time.
  - **Conquering** solves two subproblems, each on a subarray of n/2 elements. 問題一分為2, 每個大小n/2
    - ➔Takes T(n/2) time for each subproblem 2T(n/2) time for conquering.
  - **Combining** consists of calling FIND-MAX-CROSSING-SUBARRAY, which takes Θ(n) time, and a constant number of constant-time tests Find-Max-Crossing-Subarray時間複雜度為線性, 比大小為常數
    - ➔Θ(n) + Θ(1) time for combining
  - ➔

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n) \qquad \text{(absorb } \Theta(1) \text{ terms into } \Theta(n)\text{)} \end{aligned}$$

Recurrence for FIND-MAXIMUM-SUBARRAY

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

Same as Merge-Sort.

➔ $T(n) = \Theta(n \lg n)$

Can we do better?

➔ Homework #ex4.1-5 (p.75)

4.2 Strassen's matrix multiplication

$$\mathbf{C} = \mathbf{A}\mathbf{B} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

Input: Two n x n matrices

Output: n x n matrix C, where C = A · B, i.e.,

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

$$\text{for } i, j = 1, 2, \ldots, n.$$

Trivial method

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

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

$O(n)$ for $i = 1$ to n

$O(n)$ for $j = 1$ to n

$c_{ij} = 0$

$O(n)$ for $k = 1$ to n

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

return C

$$C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

Analysis: Three nested loops, each iterates n times ➔ $\Theta(n^3)$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

Simple divide-and-conquer algo

$T(n)$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

$O(1)$

```
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)
```

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

```
    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
```

Analysis of Rec-Mat-Mult

- Base case: n = 1 當矩陣大小為1時，只需做1次純量乘法
  - ✓ Perform one scalar multiplication ➔ $\Theta(1)$.
- Recursive case: n > 1 當矩陣大小大於1時，
  - ✓ Dividing takes Θ(1) time, using index calc
  - ✓ Conquering makes 8 recursive calls ➔ 8T(n/2)
  - ✓ Combining takes $\Theta(n^2)$

分割子矩陣需要常數時間，
接著遞迴求解8 個$n/2$ 大小的子矩陣
最後將$n \times n$項相加需$\Theta(n^2)$ 時間

- Recurrence

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

- ➔ $T(n) = \Theta(n^3)$ (by Master Theory)

Strassen's method

- Idea:
  - Make recursion tree less bushy.
  - Perform 7 recursive multiplications.

  想法：利用矩陣加法來減少矩陣乘法運算次數

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$$

Trick:

$P_1 = A\cdot(F-H)$

$P_2 = (A+B)\cdot H$

$P_3 = (C+D)\cdot E$

$P_4 = D\cdot(G-E)$

$P_5 = (A+D)\cdot(E+H)$

$P_6 = (B-D)\cdot(G+H)$

$P_7 = (A-C)\cdot(E+F)$

$AE+BG = P_5 + P_4 - P_2 + P_6$

$AF+BH = P_1 + P_2$

$CE+DG = P_3 + P_4$

$CF+DH = P_5 + P_1 - P_3 - P_7$

Strassen's algorithm

- Algorithm:
  1. As in the recursive method, partition each of the matrices into four n/2 x n/2 submatrices. Time: $\Theta(1)$.
     將矩陣分割為 4 個子矩陣 → $\Theta(1)$
  2. Create 10 matrices $S_1, S_2, \ldots, S_{10}$. Each is n/2 x n/2 and is the **sum** or **difference** of two matrices created in previous step. Time: $\Theta(n^2)$ to create all 10 matrices.
     創建10個矩陣$S_1, S_2, \ldots, S_{10}$, 每個維度為 n/2 x n/2 → $\Theta(n^2)$
  3. Recursively compute 7 matrix products $P_1, P_2, \ldots, P_7$, each n/2 x n/2. Time: $7T(n/2)$
     以遞迴方式算出7個子矩陣乘積$P_1, P_2, \ldots, P_7$ → $7T(n/2)$
  4. Compute n/2 x n/2 submatrices of C by adding and subtracting various combinations of the P_i. Time: $\Theta(n^2)$
     算出4個P矩陣相加減的結果 $C_1, C_2, \ldots, C_4$ → $\Theta(n^2)$

Analysis of Strassen's algorithm

- Recurrence

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

- By master method ➔ $T(n) = \Theta(n^{\lg 7})$.

Details of Strassen's algorithm

- Step 1: 分割子矩陣

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

- Step 2: Create 10 metrices

透過步驟1的小矩陣相**加減**,
創建10個大小為$n/2$ 的矩陣S_1, S_2,...,S_{10},

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

Details of Strassen's algorithm

- Step 3: Create the 7 matrices

  遞迴求解7個子矩陣乘積

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

Details of Strassen's algorithm

- Step4: Add and subtract the Pi to construct submatrices of C 算出4個P矩陣相加減的結果

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 , \\
C_{12} &= P_1 + P_2 , \\
C_{21} &= P_3 + P_4 , \\
C_{22} &= P_5 + P_1 - P_3 - P_7 .
\end{aligned}
$$

$$
\begin{array}{ll}
P_5 & A_{11}\cdot B_{11} + A_{11}\cdot B_{22} + A_{22}\cdot B_{11} + A_{22}\cdot B_{22} \\
P_4 & - A_{22}\cdot B_{11} + A_{22}\cdot B_{21} \\
P_2 & - A_{11}\cdot B_{22} - A_{12}\cdot B_{22} \\
P_6 & - A_{22}\cdot B_{22} - A_{22}\cdot B_{21} + A_{12}\cdot B_{22} + A_{12}\cdot B_{21} \\
\hline
C_{11} & A_{11}\cdot B_{11} + A_{12}\cdot B_{21}
\end{array}
$$

$$
\begin{array}{ll}
P_1 & A_{11}\cdot B_{12} - A_{11}\cdot B_{22} \\
P_2 & + A_{11}\cdot B_{22} + A_{12}\cdot B_{22} \\
\hline
C_{12} & A_{11}\cdot B_{12} + A_{12}\cdot B_{22}
\end{array}
$$

$$
\begin{array}{ll}
P_3 & A_{21}\cdot B_{11} + A_{22}\cdot B_{11} \\
P_4 & - A_{22}\cdot B_{11} + A_{22}\cdot B_{21} \\
\hline
C_{21} & A_{21}\cdot B_{11} + A_{22}\cdot B_{21}
\end{array}
$$

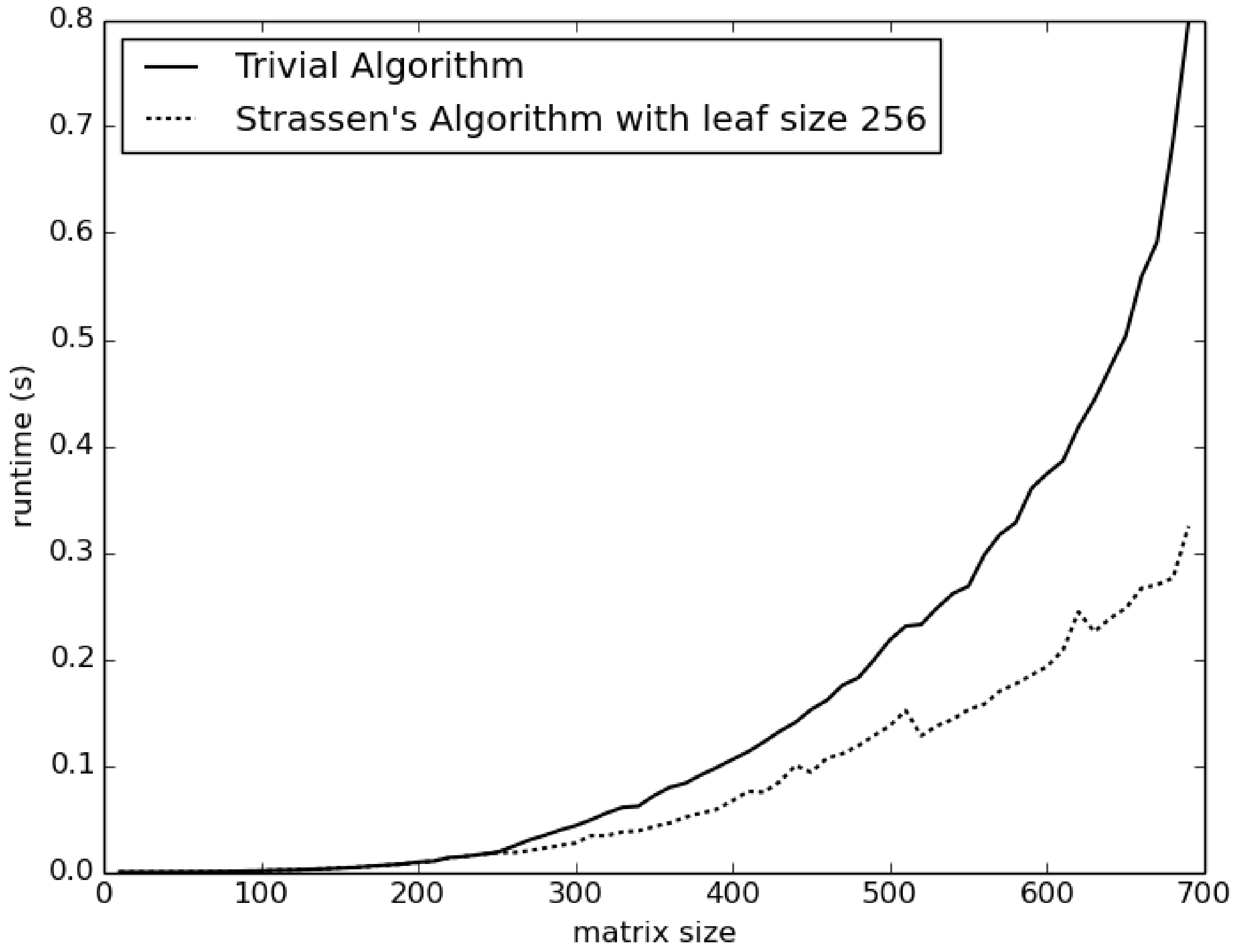
$$
\begin{array}{ll}
P_5 & A_{11}\cdot B_{11} + A_{11}\cdot B_{22} + A_{22}\cdot B_{11} + A_{22}\cdot B_{22} \\
P_1 & - A_{11}\cdot B_{22} + A_{11}\cdot B_{12} \\
P_3 & - A_{22}\cdot B_{11} - A_{21}\cdot B_{11} \\
P_7 & - A_{11}\cdot B_{11} - A_{11}\cdot B_{12} + A_{21}\cdot B_{11} + A_{21}\cdot B_{12} \\
\hline
C_{22} & A_{22}\cdot B_{22} + A_{21}\cdot B_{12}
\end{array}
$$

Technicalities 技術細節

- We neglect certain technical details when we state and solve recurrences.
- A good example of a detail that is often glossed over is the assumption of integer arguments to functions.
- Boundary conditions is ignored.
- Omit floors, ceilings.

技術細節不管它!

Strassen's 演算法與一般矩陣相乘法比較



4.3 Substitution method 替身法

- 1. Guess the solution. 先猜猜看
- 2. Use induction to find the constants and show that the solution works. 再用歸納法驗證
- E.g.:

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

1. Guess: $T(n) = n \lg n + n$
2. Induction: Basis: $n = 1 \Rightarrow n \lg n + n = 1 = T(n)$

   Inductive step:
$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \\ &= 2\left(\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}\right) + n \\ &= n \lg n + n\,. \end{aligned}$$

Example of substitution method

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(n/3) + T(2n/3) + n & \text{if } n > 1. \end{cases}$$

$$Guess: T(n) \le dn\lg n$$

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

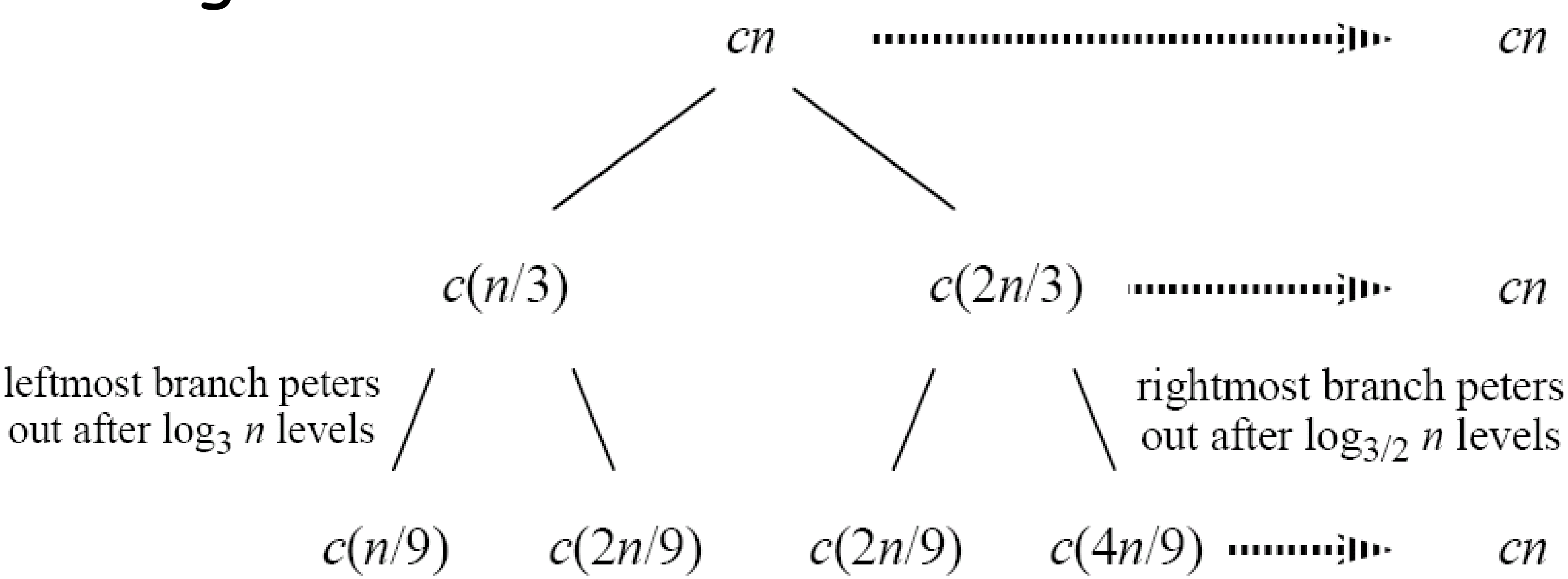
$$\text{As long as } d \ge \frac{c}{\lg 3 - 2/3}.$$

遞迴樹法

4.4 The recursion-tree method

- Generate a guess 先猜猜看
- Verify by substitution method 再用替身法驗證
- E.g.:

$$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ T(n/3) + T(2n/3) + n & \text{if } n > 1\,. \end{cases}$$



1. Guess lower bound: $\geq dn \log_3 n = \Omega(n \lg n)$
2. Guess upper bound: $\leq dn \log_{3/2} n = O(n \lg n)$
3. Prove by substitution

- Upper bound:

Guess: $T(n) \le dn \lg n$

Substitution:

$$
\begin{aligned}
T(n) &\le T(n/3) + T(2n/3) + cn \\
&\le d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) \\
&\qquad + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\le dn\lg n \qquad \text{if } -dn(\lg 3 - 2/3) + cn \le 0\,,
\end{aligned}
$$

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

$$d \ge \frac{c}{\lg 3 - 2/3}\,.$$

- Lower bound:

Guess: $T(n) \geq dn\lg n.$

Substitution: Same as upper bound, but replacing ≤ by ≥.

➔ $T(n) = \Omega(n\lg n)$

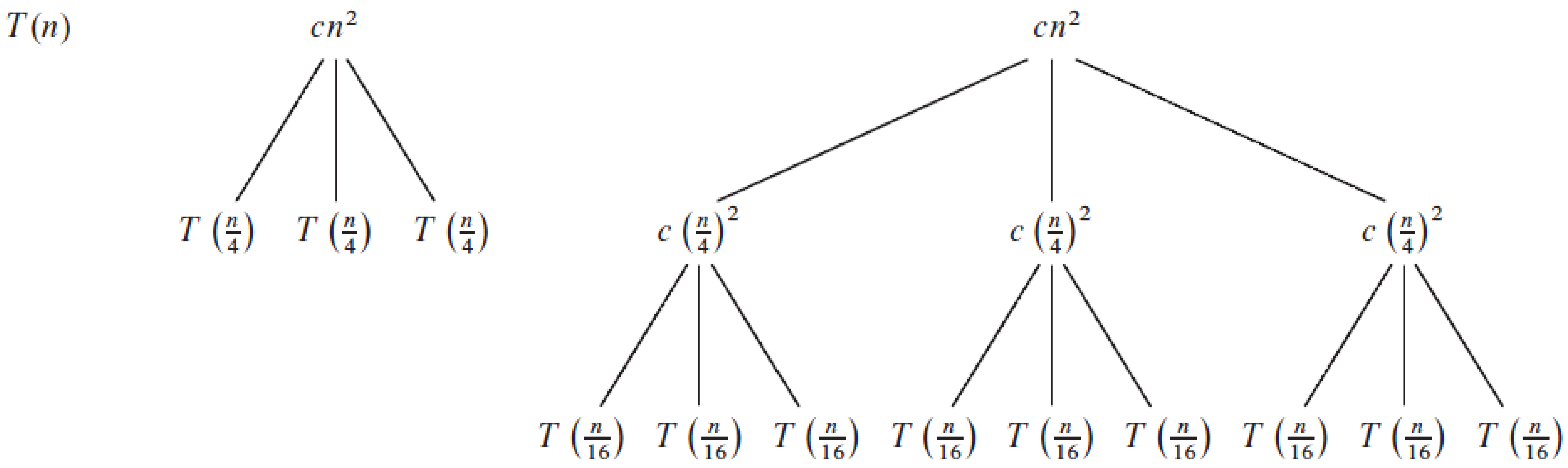
Since $T(n) = O(n\lg n)$ and $T(n) = \Omega(n\lg n)$

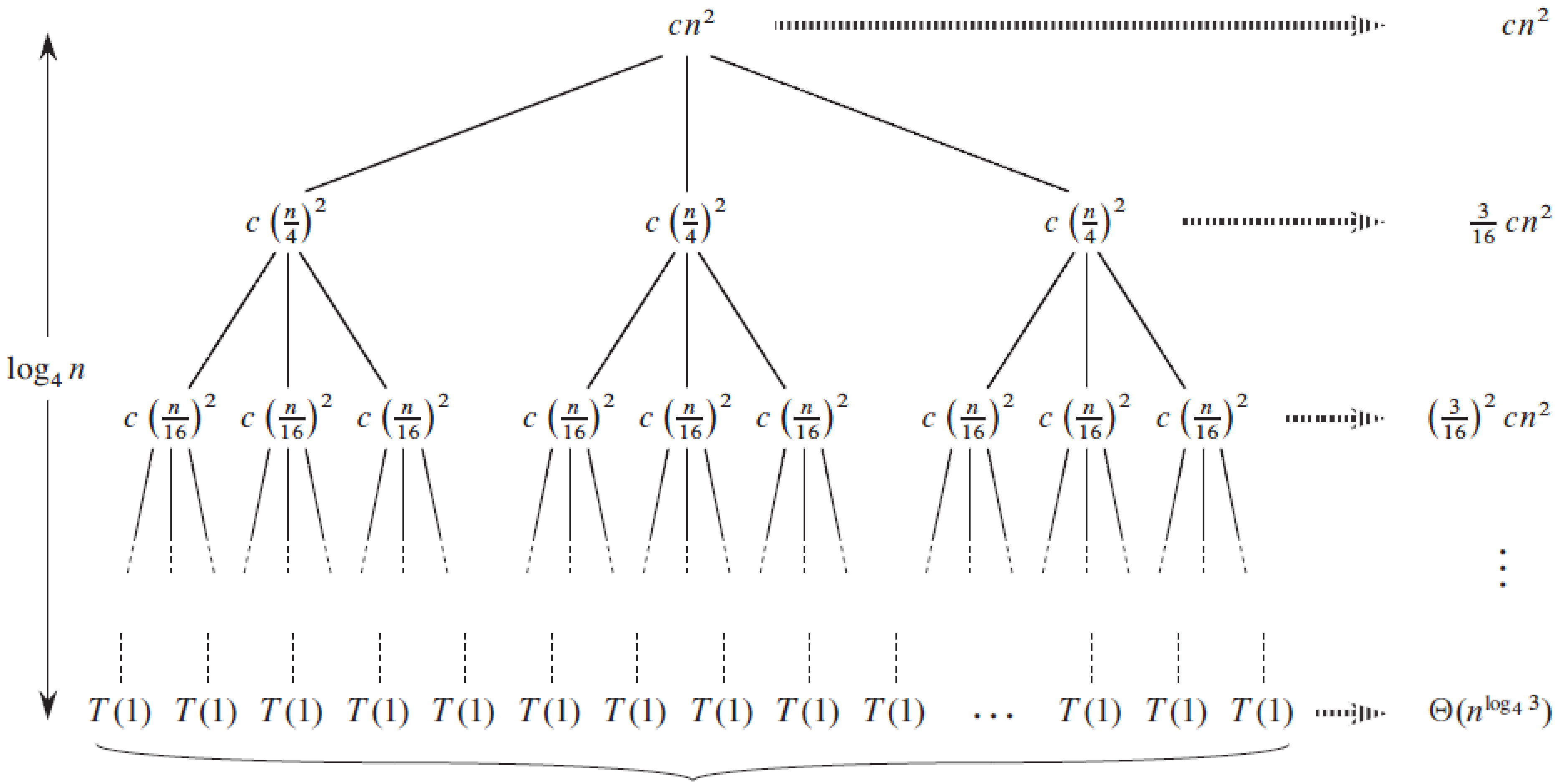
By Theorem 3.1,

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

Example

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





$n^{\log_4 3}$

(d)

Total: $O(n^2)$

每往下一層, 每個節點大小變1/4,
直到第h層節點大小變成1.

$$\frac{n}{4^h} = 1$$

$$\Rightarrow h = \log_4 n$$

每往下一層, 每層節點個數變3倍,
到第h層時, 最底層節點個數3^h ,
而 h= $\log_4 n$, 所以最底層節點數為

$$3^{log_4 n} = n^{log_4 3}$$

The cost of the entire tree

$$T(n) = \underbrace{cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2}_{\text{前面 }\log_4 n - 1\text{ 層個數}} + \underbrace{\Theta(n^{\log_4 3})}_{\text{最底層個數}}$$

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

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

($\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$)

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

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

$$\left(\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, \quad |x| < 1. \right)$$

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

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

$$= O(n^2)$$

substitution method 替身法

證明$T(n) \le dn^2$

We want to Show that $T(n) \le dn^2$ for some constant $d > 0$. using the same constant $c > 0$ as before, we have

$$
\begin{aligned}
T(n) &\le 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\le 3d\lfloor n/4 \rfloor^2 + cn^2 && \text{(利用前面結果)} \\
&\le 3d(n/4)^2 + cn^2 && (x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1) \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\le dn^2,
\end{aligned}
$$

Where the last step holds as long as $d \ge (16/13)c$.

4.5 The Master Theorem 大師法

Let a ≧ 1, b > 1, c < 1 and ε > 0 be constants, let f(n) be a function, T(n) is defined as

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

Then T(n) has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a-\varepsilon})$, 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 $f(n) = \Omega(n^{\log_b a+\varepsilon})$ and $af(n/b) \le cf(n)$, then $T(n) = \Theta(f(n))$

$$f(n) \le n^{\log_b a}$$
$$f(n) = n^{\log_b a}$$
$$f(n) \ge n^{\log_b a}$$

定理4.1

Exercise

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

$$a = 9$$

$$b = 3$$

$$f(n) = n$$

$$n^{\log_3 9} = n^2, \qquad f(n) = O(n^{\log_3 9 - 1})$$

$$\text{Case 1} \Rightarrow T(n) = \Theta(n^2)$$

Exercise

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

$$a = 1$$

$$b = 3/2$$

$$f(n) = 1$$

$$n^{\log_{3/2} 1} = n^0 = 1 = f(n)$$

$$\text{Case } 2 \Rightarrow T(n) = \Theta(\lg n)$$

Exercise

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

$$a = 3$$

$$b = 4$$

$$f(n) = n \lg n$$

$$n^{\log_4 3} = O(n^{0.793}), \quad f(n) = \Omega(n^{\log_4 3+\epsilon})$$

$$\text{Case } 3 \Rightarrow T(n) = \Theta(n \lg n)$$

- The master method does not apply to the recurrence $T(n) = 2T(n/2) + n\lg n$, even though it has the proper form: $a = 2$, $b=2$, $f(n)= n \lg n$, and $n^{\log_b a} = n$. It might seem that case 3 should apply, since $f(n)= n \lg n$ is asymptotically larger than $n^{\log_b a} = n$.

- The problem is that it is not *polynomially larger*.

Summary

- Divide-and-conquer (各個擊破法)
  - Divide, Conquer, Combine (分開, 征服, 合併)
- Recurrence Problem 遞迴問題
  - Problem with one or more base cases, and
  - Itself, with smaller arguments
- Maximum-subarray problem 最大子陣列問題
  - Find-Max-Crossing-Subarray(…)
  - Find-Maximum-Subarray(…)
- Strassen's matrix multiplication
  - Rec-Mat-Mult(…)
  - Strassen's algorithm
- Substitution method 替代法
- Recursion tree method 遞迴樹法
- The Master Theorem 大師法