Algorithms
Chapter 4 Divide-and-Conquer

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **The substitution method**
- The recursion-tree method
- The master method
- The maximum-subarray problem
- Strassen's algorithm for matrix multiplication

The purpose of this chapter

- When an algorithm contains a recursive call to itself, its running time can often be described by a recurrence.
- A **recurrence** is an equation or inequality that describes a function in terms of its value on small inputs.
  - For example: the worst-case running time of Merge-Sort

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

  - The solution is $T(n) = \Theta(n \lg n)$.
- Three methods for solving recurrences 三種方法
  - the substitution method 置換法
  - the recursion-tree method 遞迴樹
  - the master method 大師方法

Technicalities$_{1/2}$ T(n)：只定義n是整數時

- The running time $T(n)$ is only defined when n is an integer, since the size of the input is always an integer for most algorithms.
  - For example: the running time of Merge-Sort is really

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

- Typically, we ignore the boundary conditions. 忽略邊界條件
- Since the running time of an algorithm on a constant-sized input is a constant, we have $T(n) = \Theta(1)$ for sufficiently small n.
- Thus, we can rewrite the recurrence as

當n夠小時，$T(n) = \theta(1)$

忽略 ceiling 與 floor

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

Technicalities$_{2/2}$ 在計算時，忽略floors，ceiling以及boundary conditions

- When we state and solve recurrences, we often omit floors, ceilings, and boundary conditions.
- We forge ahead without these details and later determine whether or not they matter. 先忽略細節，再回頭看是否重要
- They usually don't, but it is important to know when they do.
  通常不重要，但知道背後原因是重要的

The substitution method$_{1/3}$ 置換法

- The substitution method entails two steps: 2步驟
  - Guess the form of the solution 依照經驗猜答案
  - Use mathematical induction to find the constants and show that the solution works 用數歸法證明是對的[把常數找出來]
- This method is powerful, but it obviously can be applied key only in cases when it is easy to guess the form of the answer

功能強大，但只對答案容易猜時有用

The substitution method$_{2/3}$

- For example: determine an upper bound on the recurrence

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

  - guess that the solution is $T(n)=O(n \lg n)$ 依照經驗猜答案
  - prove that there exist positive constants $c>0$ and n_0 such that $T(n) \le cn \lg n$ for all $n \ge n_0$ 證明big O的要件[找c與n_0兩個常數]
  - **Basis step**:
    - when $n=1$, $T(1) \le c_1 1 \lg 1 = 0$, which is at odds with $T(1)=1$

      $1 \le c_1 \cdot 0$
      $\Rightarrow$ 所以 c_1 不存在

    找 $n_0=2$

    Basis: 對於 $n=2,3$ 成立

    - since the recurrence does not depend directly on $T(1)$, we can replace $T(1)$ by $T(2)=4$ and $T(3)=5$ as the base cases

      $T(2) = 2T(1) + 2 = 2 \cdot 1 + 2 = 4$
      $T(3) = 2T(1) + 3 = 2 \cdot 1 + 3 = 5$

    - $T(2) \le c_1 2 \lg 2$ and $T(3) \le c_1 3 \lg 3$ for any choice of $c_1 \ge 2$
    - thus, we can choose $c_1=2$ and $n_0 = 2$

      $4 \le c_1 \cdot 2$, $5 \le c_1 \cdot 3 \cdot 1 \ldots$
      $\Rightarrow c_1 \ge 2$ 可以滿足

∵ $n \geq 2$，在做除以2取floor後，一定會遇到 2 or 3

The substitution method$_{3/3}$

- **Induction step**: 假設 $T(\lfloor \frac{n}{2} \rfloor)$ 成立 證明 $T(n)$ 成立
  - assume $T(\lfloor n/2 \rfloor) \leq c_2 \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$ for $\lfloor n/2 \rfloor$
  - then, $T(n) = 2T(\lfloor n/2 \rfloor) + n$

置換 $\leq 2(c_2 \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$

$\leq c_2 n \lg(n/2) + n$

$= c_2 n \lg n - c_2 n \lg 2 + n$

算出的結果 $= c_2 n \lg n - c_2 n + n$

目標 $\leq c_2 n \lg n$

  - the last step holds as long as $c_2 \geq 1$

⇒ 觀察後得 $c_2 \geq 1$

T(2) is true = basis
T(3) is true = basis
T(4) is true = T(2) is true + induction step
T(5) is true = T(2) is true + induction step
T(6) is true = T(3) is true + induction step
T(7) is true = T(3) is true + induction step
T(8) is true = T(4) is true + induction step
⋮
T(n) is true = $T(\lfloor \frac{n}{2} \rfloor)$ is true + induction step

- There exist positive constants $c = \max\{2, 1\}$ and $n_0 = 2$ such that $T(n) \leq cn \lg n$ for all $n \geq n_0$

c_1 c_2

$c = 2$，$n_0 = 2$ ⇒ 得證

Making a good guess 如何猜出答案

- **Experience**: $T(n)=2T(\lfloor n/2 \rfloor+17)+n$
  經驗
  - when n is large, the difference between $T(\lfloor n/2 \rfloor)$ and $T(\lfloor n/2 \rfloor+17)$ is not that large: both cut n nearly evenly in half.
  - we make the guess that $T(n) = O(n \lg n)$ 當n很大時，17就顯得相對不重要
- **Loose upper and lower bounds**: $T(n)=2T(\lfloor n/2 \rfloor)+n$
  - prove a lower bound of $T(n) = \Omega(n)$ and an upper bound of $T(n) = O(n^2)$ 分別先猜上界和下界　慢慢將上界往下，下界往上
  - gradually lower the upper bound and raise the lower bound until we converge on the tight solution of $T(n) = \Theta(n \lg n)$

目標 $O(n^2)$ ↓ $\theta(n\log n)$ ↑ $\Omega(n)$

- **Recursion trees**:
  - will be introduced later 第二節

Subtleties 小細節

- Consider the recurrence:

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

  - **guess** that the solution is $O(n)$, i.e., $T(n) \le cn$
  - then, $T(n) \le c\lfloor n/2 \rfloor + (c\lceil n/2 \rceil + 1$ 置換
    $= cn + 1$ 算出的結果
  - c does not exist $cn+1 \not\le cn \Rightarrow c$ 不存在
  - **new guess** $T(n) \le cn - b$ 再猜
  - then, $T(n) \le (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$
    $= cn - 2b + 1$
    $\le cn - b$ for $b \ge 1$
  - also, the constant c must be chosen large enough to handle the boundary conditions
    選擇c時，要滿足basis條件

Avoiding pitfalls 避免犯錯

- It is easy to err in the use of asymptotic notation.
- For example: $T(n) = 2T(\lfloor n/2 \rfloor) + n$
  - guess that the solution is $O(n)$, i.e., $T(n) \le cn$
  - then, $T(n) = 2T(\lfloor n/2 \rfloor) + n$

    $\le 2(c\lfloor n/2 \rfloor) + n$ 置換

    $\le cn + n$ 算出的結果

    $= O(n)$ ← **wrong** c 還沒找出來 [事實上 c 不存在]

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

Outline

- The substitution method
- **The recursion-tree method** 用遞迴樹
- The master method
- The maximum-subarray problem
- Strassen’s algorithm for matrix multiplication

用遞迴樹產生答案，用置換法驗證

The recursion-tree method

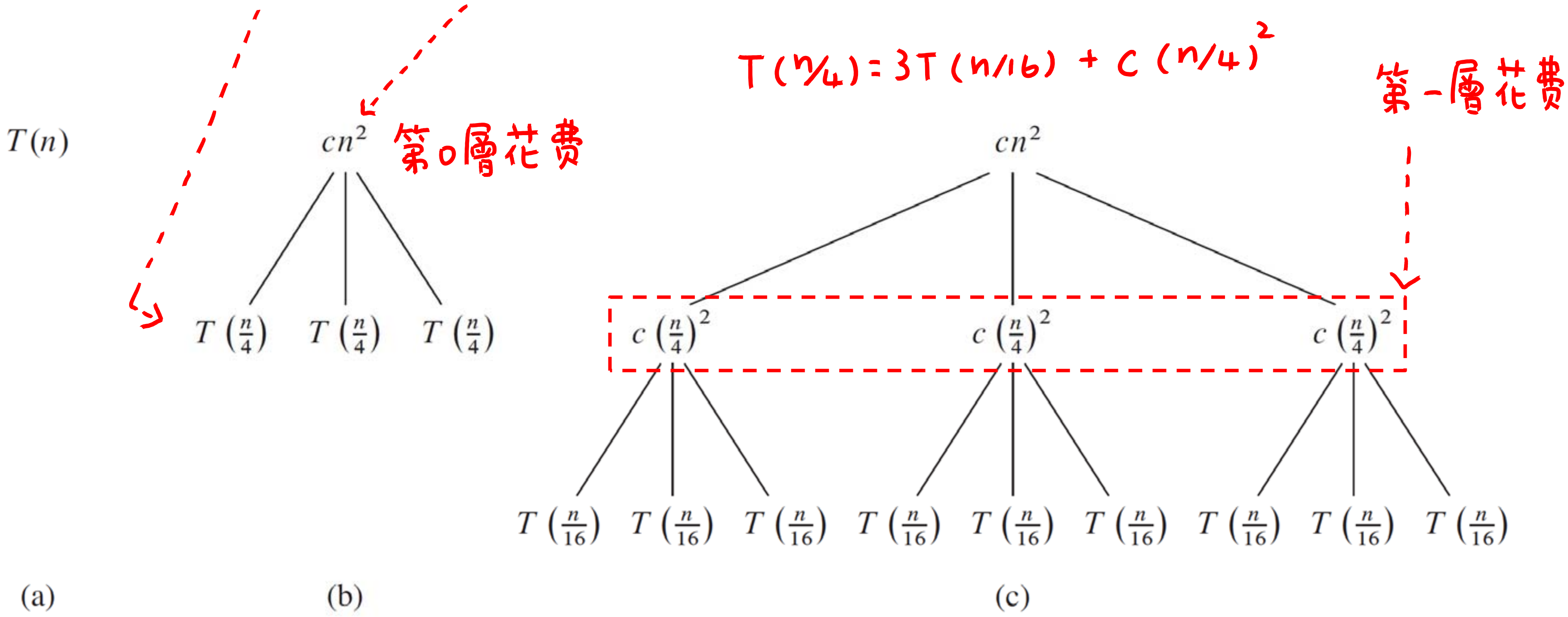
- A recursion tree is best used to generate a good guess, which is then verified by the substitution method.
- 忽略細節
  Tolerating a small amount of “sloppiness”, we could use recursion-tree to generate a good guess.
- One can also use a recursion tree as a direct proof of a solution to a recurrence. 也可用遞迴樹證明
- Ideas:
  - in a **recursion tree**, each node represents the cost of a single subproblem 点代表 cost
  - 將同一層花費加起來
    sum the costs within each level to obtain a set of per-level costs
  - sum all the per-level costs to determine the total cost
    將每一層花費加起來

An example

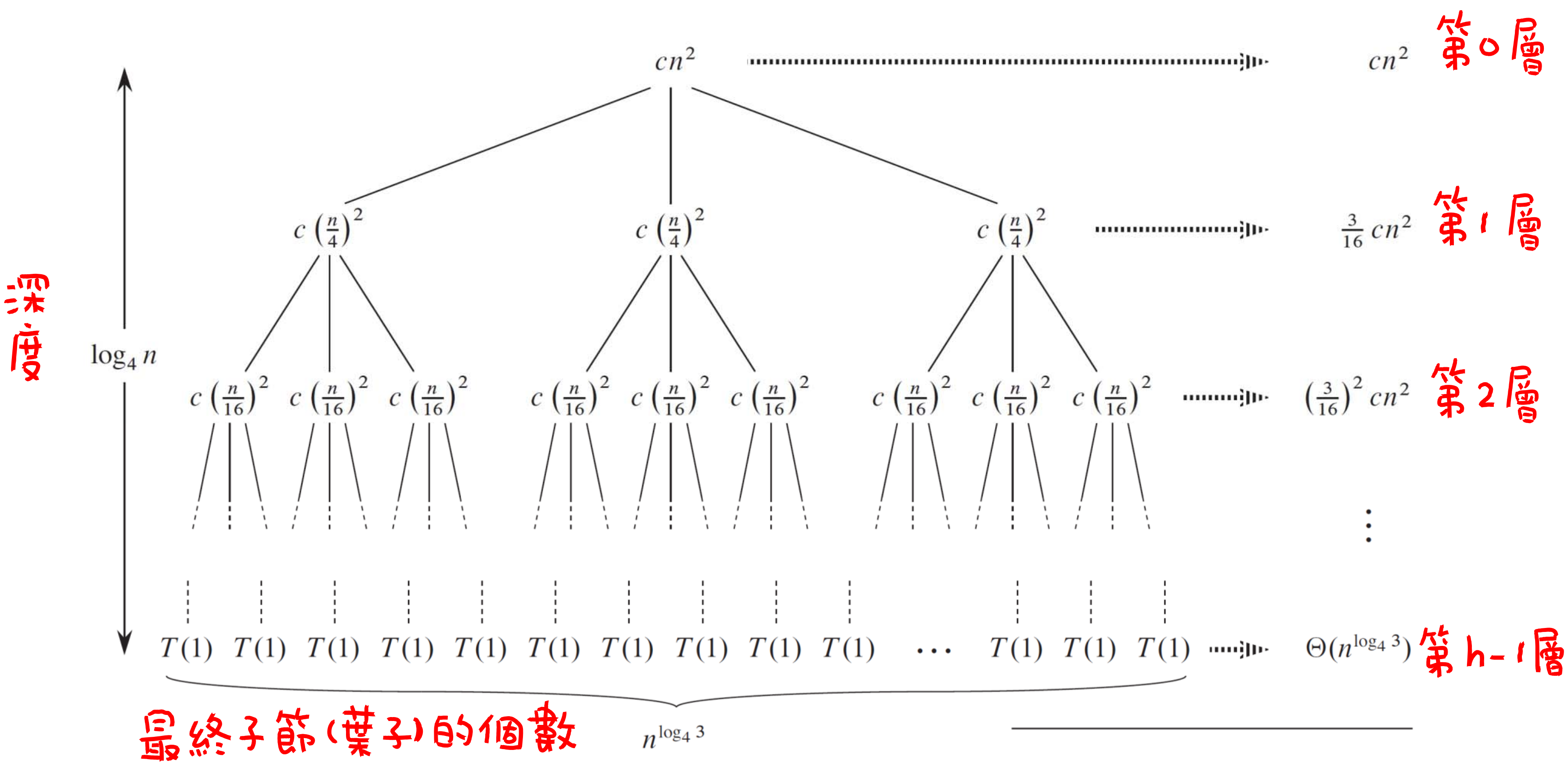
- For example: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$
- Tolerating the sloppiness:
  - ignore the floor in the recurrence 將floor忽略
  - assume n is an exact power of 4 $n = 4^k$
- Rewrite the recurrence as $T(n) = 3T(n/4) + cn^2$

The construction of a recursion tree$_{1/2}$

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



The construction of a recursion tree$_{2/2}$



(d)

Determine the cost of the tree$_{1/2}$

共h層

第i層 0 1 2 ... h-1

$n \quad \frac{n}{4} \quad \frac{n}{16} \cdots \frac{n}{4^{h-1}} = 1$

- The subproblem size for a node at depth i is $n/4^i$. $4^{h-1} = n$, $h = \log_4 n+1$
- Thus, the tree has $\log_4 n + 1$ levels (0, 1, 2,..., $\log_4 n$).
- Each node at depth i, has a cost of $c(n/4^i)^2$ for $0 \le i \le \log_4 n$.
- So, the total cost over all nodes at depth i is $3^i * c(n/4^i)^2 = (3/16)^i cn^2$. 第i層的花费 node數 每個node的花费
- The last level, at $\log_4 n$, has $3^{\log_4 n} = n^{\log_4 3}$ nodes. (3^i) 最後一層的桌數
- The cost of the entire tree:

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

$$= \sum_{i=0}^{\log_4 n} (\frac{3}{16})^i cn^2$$

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

最後一層花费 $= n^{\log_4 \frac{3}{16}} \cdot cn^2$

$= n^{\log_4 3 - \log_4 16} \cdot cn^2$

$= cn^{\log_4 3}$

$=$ 最後一層的桌數 $\cdot c$

Determine the cost of the tree$_{2/2}$

- Take advantage of small amounts of sloppiness, we have

多算一些 → ∞

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

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 = \frac{1}{1-(3/16)} cn^2$$

$$= \frac{16}{13} cn^2 = O(n^2)$$

- Thus, we have derived a guess of $T(n) = O(n^2)$.

遞迴樹產生的答案
會忽略細節，所算出的答案非準確

Verify the correctness of our guess 用置換法證明它正確

- Now we can use the substitution method to verify that our guess is correct.
- 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

$$\begin{aligned}
T(n) &= 3T(\lfloor n/4 \rfloor) + \theta(n^2) \\
&\le 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\le 3d\lfloor n/4 \rfloor^2 + cn^2 \quad \text{置換} \\
&\le 3d(n/4)^2 + cn^2 \\
&= 3/16dn^2 + cn^2 \quad \text{算出的結果} \\
&\le dn^2, \quad \text{目標}
\end{aligned}$$

$\theta(n^2)$
$\Rightarrow c_1 n^2 \le f(n) \le c_2 n^2$

where the last step holds for $d \ge (16/13)c$.

Another example

分成二部份 divide and combine 時間

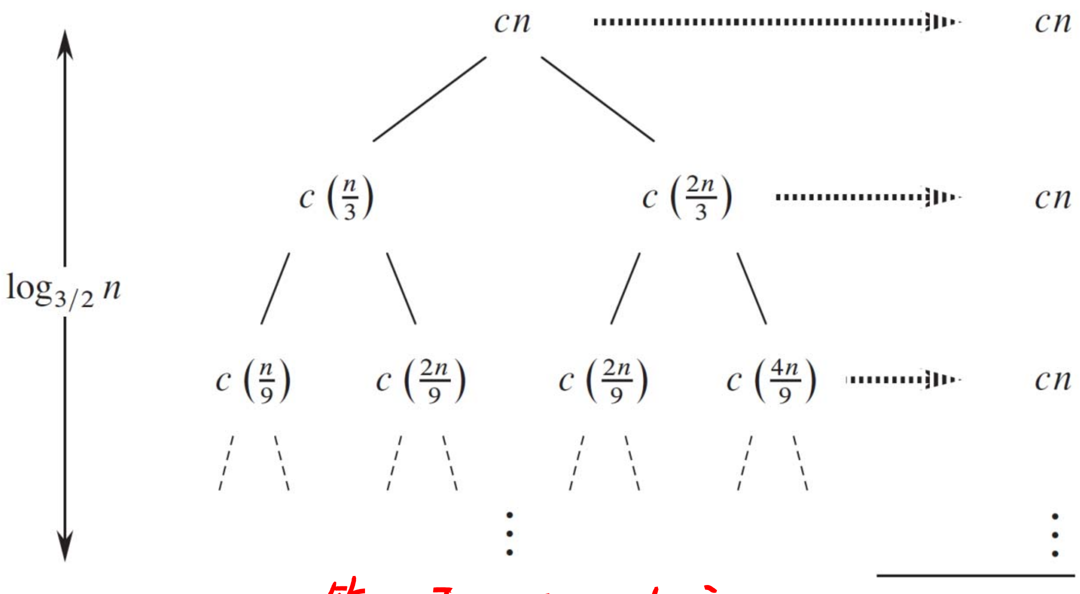
- Another example: $T(n) = T(n/3) + T(2n/3) + O(n)$.
- The recursion-tree:

共h層

第i層 $0 \quad 1 \quad 2 \quad \cdots \quad h-1$

$n \quad \frac{n}{(\frac{3}{2})^1} \quad \frac{n}{(\frac{3}{2})^2} \quad \cdots \quad \frac{n}{(\frac{3}{2})^{h-1}} = 1$

$(\frac{3}{2})^{h-1} = n, \quad h = \log_{\frac{3}{2}} n + 1$



第1種可能性答案 → Total: $O(n \lg n)$

Determine the cost of the tree

- In the figure, we get a value of cn for every level.
- The height of tree is $\log_{3/2}n$.
- Intuitively, we expect the solution to the recurrence to be at most $O(cn\log_{3/2}n) = O(n\lg n)$. ⇐ 第1種可能性答案
- The recursion tree has fewer than $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ leaves.
- The total cost of all leaves would then be $\theta(n^{\log_{3/2} 2})$, which is $\omega(n\lg n)$. ⇐ 第2種可能性答案
- Also, not all levels contribute a cost of exactly cn.
  不是每一層都花 cn 有些 leaf 尚未到最後一層就消失了(結束)
- Thus, we derived a guess of $T(n)=O(n\lg n)$.

所以,還是猜 $T(n)=O(n\log n)$

共有 $h=\log_{\frac{3}{2}} n+1$ 層 ⇒ 樹高 $=\log_{\frac{3}{2}} n$
⇒最後一層 leaf 個數 $< 2^{\log_{\frac{3}{2}} n} = n^{\log_{\frac{3}{2}} 2}$
⇒比常數倍的 $n\log n$ 大
$(n^{\varepsilon} > \log n,\ \varepsilon > 0, n\to\infty)$

假設T(n/3)和T(2n/3)成立
假設 T(n/3) ≤ d(n/3) lg (n/3) 和 T(2n/3) ≤ d(2n/3) lg (2n/3)

Verify the correctness of our guess

- We can verify the guess by the substitution method.
- We have
$$\begin{aligned} T(n) &= T(n/3)+T(2n/3)+O(n) \\ &\le T(n/3)+T(2n/3)+cn \\ &\le d(n/3)\lg(n/3)+d(2n/3)\lg(2n/3)+cn \quad \text{置換} \\ &= (d(n/3)\lg n-d(n/3)\lg 3) \\ &\quad +(d(2n/3)\lg n-d(2n/3)\lg(3/2))+cn \\ &= dn\lg n-d((n/3)\lg 3+(2n/3)\lg(3/2))+cn \\ &= dn\lg n-d((n/3)\lg 3+(2n/3)\lg 3-(2n/3)\lg 2)+cn \\ &= dn\lg n-dn(\lg 3-2/3)+cn \quad \text{算出的結果} \\ &\le dn\lg n \quad \text{目標} \Rightarrow \text{求 } d=? \end{aligned}$$

⇒ d 的範圍

for $d \ge c/(\lg 3-(2/3))$.

Outline

- The substitution method
- The recursion-tree method
- **The master method**
- The maximum-subarray problem
- Strassen's algorithm for matrix multiplication

The master method$_{1/2}$ 大師方法給一個公式解

- The master method provides a "cookbook" method for solving recurrences of the form

a個子問題，每一個大小$\frac{n}{b}$

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

$T(\frac{n}{b}) \Rightarrow$ 子問題所需時間

$f(n) =$ 分割和合併所需時間

  - $a \geq 1$ and $b > 1$ are constants
  - $f(n)$ is an asymptotically positive function n 夠大時，f(n) 值為正

- It requires memorization of three cases, but then the solution of many recurrences can be determined quite easily.

三種情形，可解大多數的 recurrences

⇒ 有一些不能解

The master method$_{2/2}$

- The recurrence $T(n) = aT(n/b) + f(n)$ describes the running time of an algorithm that
  - divides a problem of size n into a subproblems, each of size n/b
  - each of subproblems is solved recursively in time $T(n/b)$
  - the cost of dividing and combining the results is $f(n)$
- For example, the recurrence arising from the MERGE-SORT procedure has $a = 2$, $b = 2$, and $f(n) = \Theta(n)$. $f(n) = \underbrace{\text{divide}}_{\Theta(1)} + \underbrace{\text{merge}}_{\Theta(n)}$

  Merge-Sort：分割成2個子問題，子問題大小為$\frac{n}{2}$
- Normally, we omit the floor and ceiling functions when writing divide-and-conquer recurrences of this form.

  忽略 ceiling 和 floor

Master theorem $n^{\log_b a}$ v.s. $f(n)$ 比大小 ⇒ 誰大時間複雜度就為誰

- **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

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

可以是 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then, $T(n)$ can be bounded asymptotically as follows.

$n^{\log_b a}$ 夠大，相差 n^{ε} ⇒ $T(n) = \theta(n^{\log_b a})$

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \theta(n^{\log_b a})$.
2. If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} \lg n)$. 相等，乘以 $\lg n$
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 $T(n) = \theta(f(n))$. $n^{\log_b a}$ 小，相差 n^{ε} 且滿足正規情況 ⇒ $T(n) = \theta(f(n))$

Intuition behind the master method

- Intuitively, the solution to the recurrence is determined by comparing the two functions $f(n)$ and $n^{\log_b a}$.
  - Case 1: if $n^{\log_b a}$ is asymptotically larger than $f(n)$ by a factor of n^{ε} for some constant $\varepsilon > 0$, then the solution is $T(n) = \theta(n^{\log_b a})$.
  - Case 2: if $n^{\log_b a}$ is asymptotically equal to $f(n)$, then the solution is $T(n) = \theta(n^{\log_b a} \lg n)$.
  - Case 3: if $n^{\log_b a}$ is asymptotically smaller then $f(n)$ by a factor of n^{ε}, and the function $f(n)$ satisfies the "regularity" condition that $af(n/b) \le cf(n)$, then the solution is $T(n) = \theta(f(n))$.
- The three cases do not cover all the possibilities for $f(n)$.

三種情形，可解大多數的recurrences
⇒有一些不能解

Using the master method$_{1/3}$

- Example 1: $T(n) = 9T(n/3) + n$
  - For this recurrence, we have $a = 9$, $b = 3$, $f(n) = n$.
  - Thus, $n^{\log_b a} = n^{\log_3 9} = \theta(n^2)$.
  - Since $f(n) = O(n^{\log_3 9-\varepsilon})$, where ε = 1, we can apply case 1.
  - The solution is $T(n) = \Theta(n^2)$.

$n \cdot n^{\varepsilon} = n^2$

$n^{\log_b a}$ 較大, $\varepsilon = 1$

- Example 2: $T(n) = T(2n/3) + 1$
  - For this recurrence, we have $a = 1$, $b = 3/2$, $f(n) = 1$.
  - Thus, $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.
  - Since $f(n) = O(n^{\log_b a}) = \theta(1)$, we can apply case 2.
  - The solution is $T(n) = \Theta(\lg n)$.

相等, $n^{\log_b a} \times \lg n = 1 \times \lg n = \lg n$

Using the master method$_{3/3}$

$n^{\log_b a} = n$, $f(n) = n \lg n$
$\therefore n < n \lg n$ ，但是不夠大
[只有大於 $\lg n$，不到 n^{ε}] $n^{\varepsilon} > \lg n$
$\lim_{n\to\infty} \frac{\log_b n}{n^a} = 0$, for $a>0$, $b \geq 1$

- Example 4: $T(n) = 2T(n/2) + n\lg n$
  - For this recurrence, we have $a = 2$, $b = 2$, $f(n) = n\lg n$.
  - The function $f(n) = n\lg n$ is asymptotically larger than $n^{\log_b a} = n^{\log_2 2} = n$.
  - But, it is not polynomially larger since the ratio $f(n)/n^{\log_b a} = (n\lg n)/n = \lg n$ is asymptotically less than n^{ε} for any positive constant ε.
  - Consequently, the recurrence falls into the gap between case 2 and case 3. 遞迴式落在 case 2 和 case 3 之間
- If $g(n)$ is asymptotically larger than $f(n)$ by a factor of n^{ε} for some constant $\varepsilon>0$, then we said $g(n)$ is **polynomially larger** than $f(n)$.

$f(n)$ 大 $\begin{cases} 大於 n^{\varepsilon} \\ af(n/b) \leq cf(n) \end{cases}$

找常數 $c < 1$

Using the master method$_{2/3}$

- Example 3: $T(n) = 3T(n/4) + n \lg n$
  - For this recurrence, we have $a = 3$, $b = 4$, $f(n) = n \lg n$.
  - Thus, $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$.
  - For sufficiently large n, $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$.
  - Since $f(n) = \Omega(n^{\log_4 3 + \varepsilon})$ with $\varepsilon \approx 0.2$ and the regularity condition holds for $f(n)$ case 3 applies.
  - The solution is $T(n) = \Theta(n \lg n)$.

Outline

- The substitution method
- The recursion-tree method
- The master method
- **The maximum-subarray problem**
- Strassen's algorithm for matrix multiplication

最大子陣列

The maximum-subarray problem

- **Input:** An array $A[1 \ldots n]$ of numbers.
  - Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative.
- **Output:** Indices i and j such that $A[i \ldots j]$ has the greatest sum of any contiguous subarray of array A.

找出i,j使得從i加到j的和是最大

- For example:
  - The subarray $A[8 \ldots 11]$, with sum 43, has the greatest sum of any contiguous subarray of array A. 回傳8和11

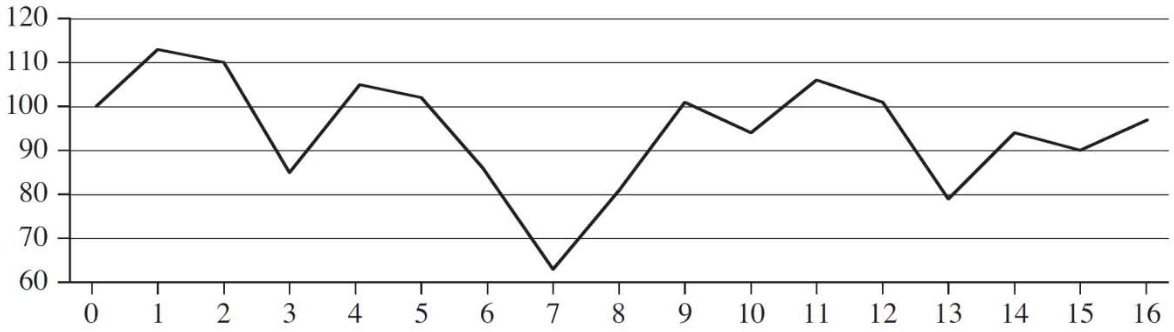
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

An application

- You have the prices that a stock traded at over a period of *n* consecutive days.
- When should you have bought the stock? When should you have sold the stock? 第7天買，第11天賣賺最多

回傳8和11



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

Solving by brute-force algorithm

- **Brute-force algorithm:**
  - Check all $\binom{n}{2} + n = \theta(n^2)$ subarrays. 有 $\binom{n}{2} + n$ 種可能性
  - Organize the computation so that each subarray *A*[*i*…*j*] takes *O*(1) time.
  - So that the brute-force solution takes $\theta(n^2)$ time.

A[9…13] = -1

	8	9	10	11	12	13	14	15	16	17
A	…	-6	5	-4	3	1	-2	3	-6	…

算A[9…13]後算A[9…14]
只需O(1)時間
暴力法θ(n²)時間

O(1) time

A[9…14] = -3

	8	9	10	11	12	13	14	15	16	17
A	…	-6	5	-4	3	1	-2	3	-6	…

Solving by divide-and-conquer

- **Divide** by splitting into two subarrays $A[low...mid]$ and $A[mid+1...high]$, mid is the midpoint of $A[low...high]$.

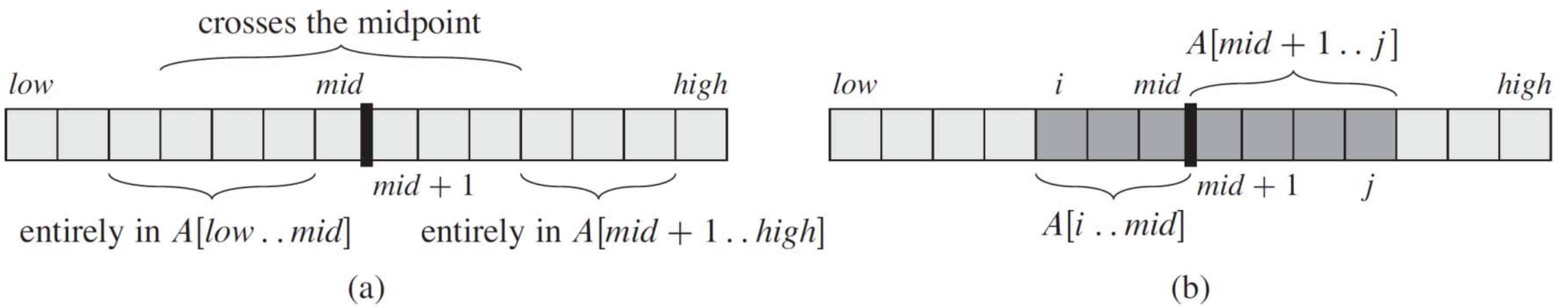
  找出中間那一個

- **Conquer** by recursively finding a maximum subarrays of the two subarrays $A[low...mid]$ and $A[mid+1...high]$.

  分別算左邊最大和右邊最大(子問題)

- **Combine** by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three.

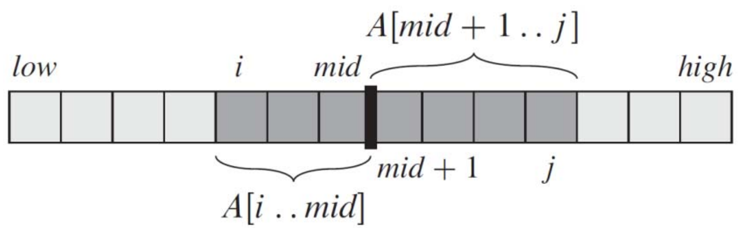
  找出跨越中間的最大，從三種可能性中取最佳(非子問題)



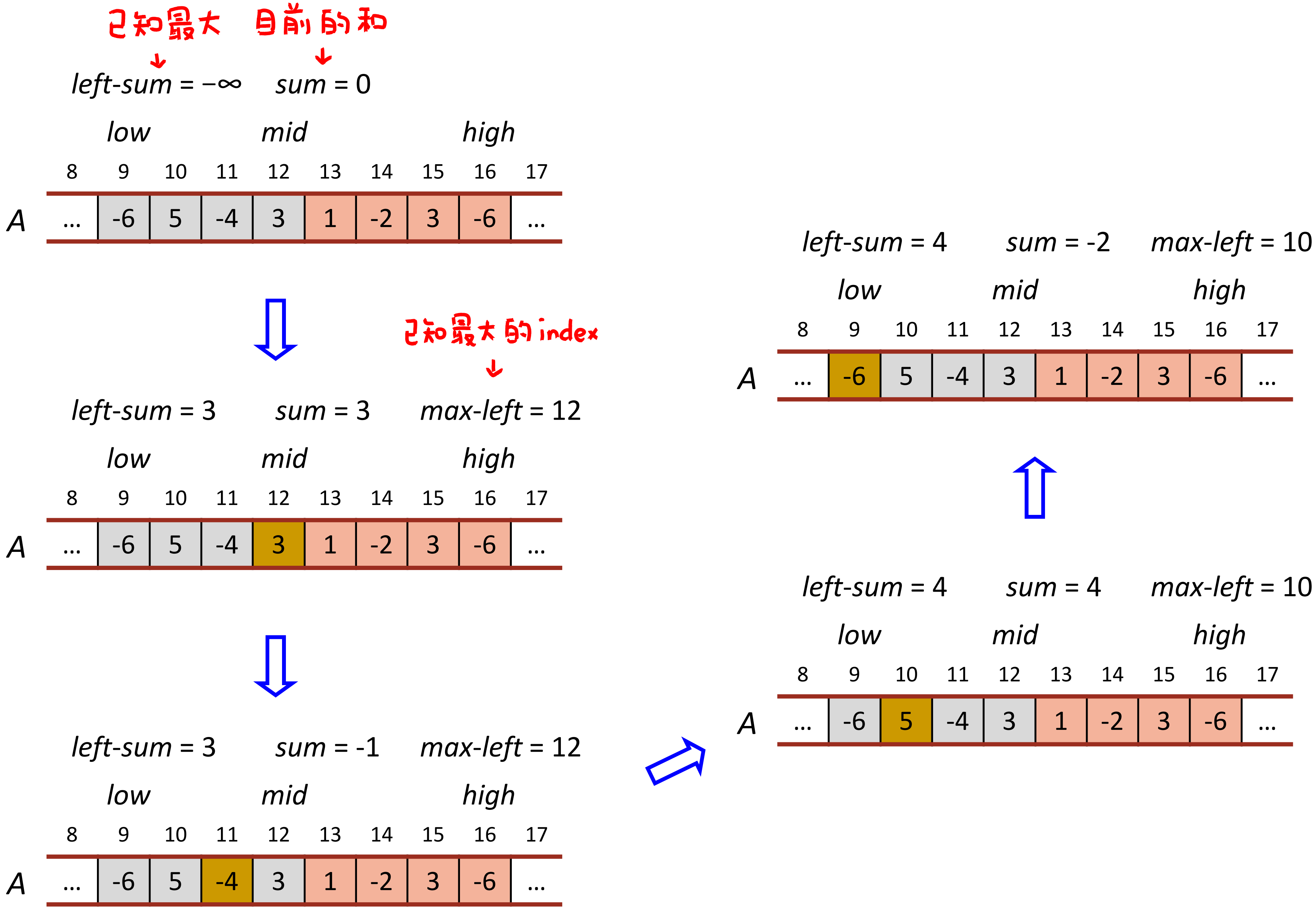
算跨中最大

Maximum subarray that crosses the midpoint

- **Not** a smaller instance of the original problem: has the added restriction that the subarray must cross the midpoint.
  不是原問題的子問題，因增加要跨中的條件
- Any subarray crossing the midpoint $A[mid]$ is made of two subarrays $A[i...mid]$ and $A[mid+1...j]$.
  分別找出A[i…mid]的最大和A[mid+1…j]的最大
- Find maximum subarrays of the form $A[i...mid]$ and $A[mid+1...j]$ and then combine them.



→將i, j找出


已知最大
目前的和
left-sum = −∞
sum = 0
low
mid
high
8 9 10 11 12 13 14 15 16 17
A
... -6 5 -4 3 1 -2 3 -6 ...
已知最大的index
left-sum = 3
sum = 3
max-left = 12
low
mid
high
8 9 10 11 12 13 14 15 16 17
A
... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 3
sum = -1
max-left = 12
low
mid
high
8 9 10 11 12 13 14 15 16 17
A
... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 4
sum = 4
max-left = 10
low
mid
high
8 9 10 11 12 13 14 15 16 17
A
... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 4
sum = -2
max-left = 10
low
mid
high
8 9 10 11 12 13 14 15 16 17
A
... -6 5 -4 3 1 -2 3 -6 ...

Find-Max-Crossing-Subarray

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

算左边

```
left-sum ← −∞
sum ← 0
```

$\Theta(1)$ 初始化

```
for i ← mid downto low
    sum ← sum + A[i]
    if sum > left-sum
        left-sum ← sum
        max-left ← i
```

$\Theta(n)$ 從mid往左,一次加一個
如果目前的和 > 已知最大
→更新資訊

算右边

```
right-sum ← −∞
sum ← 0
```

$\Theta(1)$ 初始化

```
for j ← mid +1 to high
    sum ← sum + A[j]
    if sum > right-sum
        right-sum ← sum
        max-right ← j
```

$\Theta(n)$ 從mid往右,一次加一個
如果目前的和 > 已知最大
→更新資訊

```
return (max-left, max-right, left-sum + right-sum)
```

Time: $\Theta(n)$.

Divide-and-conquer procedure

```
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)
```

只有一個元素 $\Theta(1)$

解決子問題

$T(n/2)$ 算左邊最大

$T(n/2)$ 算右邊最大

合併

$\Theta(n)$ 算跨中最大

$\Theta(1)$ 回傳3種可能性中最大的

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

Analyzing maximum-subarray

- For simplicity, assume that n is a power of 2.

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

$f(n) = \underbrace{\text{divide}}_{\theta(1)} + \underbrace{\text{combine}}_{\theta(n)}$

  - The base case occurs when $n = 1$.
  - **Divide**: compute the middle of the subarray, $D(n) = \Theta(1)$.
  - **Conquer**: Recursively solve 2 subproblems, each of size $n/2$.
  - **Combine**: Combining consists of calling FIND-MAX-CROSSING-SUBARRAY, which takes $\Theta(n)$ time, and a constant number of constant-time tests $\Rightarrow C(n) = \Theta(n) + \Theta(1)$ time for combining.
- By using master method, we have $T(n) = \Theta(n \lg n)$.

大師方法 case 2

Outline

- The substitution method
- The recursion-tree method
- The master method
- The maximum-subarray problem
- **Strassen's algorithm for matrix multiplication**

Matrix multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$

- **Input:** Two $n \times n$ matrices, $A = (a_{ij})$ and $B = (b_{ij})$.
- **Output:** $n \times n$ matrix, $C = (c_{ij})$, where $C = A \cdot B$,i.e.,

$$c_{ji} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad \text{for } i, j = 1, 2, \ldots, n.$$

  - Need to compute n^2 entries of C.
  - Each entry is the sum of n values.

SQUARE-MATRIX-MULTIPLY (A, B)

$n \leftarrow A.rows$
let C be a new $n \times n$ matrix
for $i \leftarrow 1$ **to** n
    for $j \leftarrow 1$ **to** n
        $c_{ij} \leftarrow 0$
        for $k \leftarrow 1$ **to** n
            $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
Return C

$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} + \ldots + a_{1n} \times b_{n1}$

C有n^2個元素，每一個皆為n個值相加

Time: $\Theta(n^3)$.

Simple divide-and-conquer method

- Partition each of A, B and C into four $n/2 \times n/2$ matrices, so that we rewrite the equation $C = A \cdot B$ as C=A·B的另一種看法

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

- The four corresponding equations are:
  - $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21},$
  - $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22},$
  - $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21},$
  - $C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$

8個子問題，每一個大小是$\frac{n}{2}$

$\boxed{A_{11} \cdot B_{11}} + \boxed{A_{12} \cdot B_{21}}$

相加需要$\Theta(n^2)$的時間

- Each of these equations multiplies two $n/2 \times n/2$ matrices and then adds their $n/2 \times n/2$ products.

Procedure of matrix-multiply-recursive

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE (A, B)
    n ← A.rows
    let C be a new n × n matrix
    if n == 1                          // base case: only one element
        c11 ← a11 · b11
    else partition each of A, B and C into four n/2 × n/2 matrices
        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
```

Lines 1–2: $\Theta(1)$

Lines 3–4: $\Theta(1)$

Lines 5–9: $8T(n/2) + \Theta(n^2)$

$f(n) =$ divide + combine

$\Theta(1)$ $\Theta(n^2)$

8個子問題，每一個大小是 $\frac{n}{2}$

$\boxed{A_{11} \cdot B_{11}} + \boxed{A_{12} \cdot B_{21}}$

相加需要 $\Theta(n^2)$ 的時間

Analyzing

- For simplicity, assume that *n* is a power of 2.

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

$f(n) = \underbrace{\text{divide}}_{\theta(1)} + \underbrace{\text{combine}}_{\theta(n^2)}$

  - The base case occurs when *n* = 1.
  - **Divide**: Partition A, B and C into four $n/2 \times n/2$ matrices by index calculation takes Θ(1), *D*(*n*) = Θ(1). 8個子問題，每一個大小是 $\frac{n}{2}$
  - **Conquer**: Recursively solve 8 subproblems, each of size *n*/2.
  - **Combine**: Combining takes Θ(n^2) time to add $n/2 \times n/2$ matrices four times. ⇒ *C*(*n*) = Θ(n^2) time for combining. $\boxed{A_{11} \cdot B_{11}} + \boxed{A_{12} \cdot B_{21}}$

相加需要$\theta(n^2)$的時間

- By using master method, we have *T*(*n*) = Θ(n^3).

大師方法 case 1

Strassen's method

- Step 1: partition each of A, B and C into four $n/2 \times n/2$ matrices. **Time: Θ(1).** 產生 $A_{11}, A_{12}, A_{21}, A_{22}$
  $B_{11}, B_{12}, B_{21}, B_{22}$ $C_{11}, C_{12}, C_{21}, C_{22}$
- Step 2: create 10 matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1. **Time: Θ(n^2).** 產生 $S_1, S_2 \ldots S_{10}$ 這10個矩陣

分割

- Step 3: using the submatrices created in Step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, \ldots, P_7$. Each matrix P_i is $n/2 \times n/2$. **Time: 7T(n/2).**
  解決 $P_1, P_2 \ldots P_7$，這7個子問題

擊破

- Step 4: Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. **Time: Θ(n^2).** 由 $P_1, P_2 \ldots P_7$ 的解產生矩陣 C

合併

Step 2: create the 10 matrices

- $S_1 = B_{12} - B_{22}$,
- $S_2 = A_{11} + A_{12}$,
- $S_3 = A_{21} + A_{22}$,
- $S_4 = B_{21} - B_{11}$,
- $S_5 = A_{11} + A_{22}$,
- $S_6 = B_{11} + B_{22}$,
- $S_7 = A_{12} - A_{22}$,
- $S_8 = B_{21} + B_{22}$,
- $S_9 = A_{11} - A_{21}$,
- $S_{10} = B_{11} + B_{12}$. **Time: $\Theta(n^2)$.**

Step 3: create the 7 matrices

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

Time: 7*T*(*n*/2).

Step 4: construct submatrices of C

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

Time: $\Theta(n^2)$.

Analyzing

- For simplicity, assume that n is a power of 2.

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

$f(n) = \underbrace{\text{divide}}_{\theta(n^2)} + \underbrace{\text{combine}}_{\theta(n^2)}$

  - The base case occurs when n = 1.
  - **Divide**: Partition A, B and C into four $n/2 \times n/2$ matrices by index calculation takes $\Theta(1)$. Creating the matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ takes $\Theta(n^2)$, $D(n) = \Theta(1) + \Theta(n^2) = \Theta(n^2)$.
    7個子問題，每一個大小是$\frac{n}{2}$
  - **Conquer**: Recursively solve 7 subproblems, each of size $n/2$.
  - **Combine**: Combining takes $\Theta(n^2)$ time to add and subtract $n/2 \times n/2$ matrices. $\Rightarrow C(n) = \Theta(n^2)$ time for combining.

- By using master method, we have $T(n) = \Theta(n^{\lg 7})$.

大師方法 case 1