Unit 2
Analysis of Algorithms + Divide-and-Conquer

T.H. Cormen et al., "Introduction to Algorithms", 3rd ed., Chapters 3-4.

Analysis of Algorithms

- Issues:
  - Correctness
  - ***Time efficiency*** (discussed in this unit)
  - Space (or other resources) efficiency
  - Optimality

只討論時間分析, 並非其他資源不重要而是....

Theoretical Analysis of Time Efficiency

- Decide on parameter ***n*** indicating *input size*.
- Identify algorithm's *basic operation* (the operation that contributes most towards the running time).
- Let $T(n)$ = # basic operations. Then,
  running time $\approx cT(n)$.
- In the following, $T(n)$ may denote # basic operations or running time of the algorithm.

Cases of Analysis

- Let $\mathcal{I}$ be the set of all inputs.
- Let function $t_A(i):\mathcal{I} \to \mathbb{R}^*$ be the running time of algorithm A with input i.
- Worst case analysis :
  $T(n) = \max\{ t_A(i) : i \in \mathcal{I}, |i| = n \}$
- Average case analysis : not $(T_{worst} + T_{best})/2$
  $T(n) = \sum_{|i|=n} t_A(i) \cdot p(i)$, where $\sum_{|i|=n} p(i) = 1$

Some statistical distribution of inputs must be assumed.

Analysis of Insertion Sort (複習)

- $T(n) = T(n-1) + f(n)$
  $T(1) = 0$
- For each case:

  Best case: $f(n) = 1 \Rightarrow T(n) = n - 1$

  Worst case: $f(n) = n \Rightarrow T(n) = n(n+1)/2$

  Average case: $f(n) = (n+1)/2$

  (uniform distribution) $= (1+2+3+\ldots+n)/n$

  $\Rightarrow T(n) \approx$ ___.

複雜度表示法的簡化

- 例： Selection Sort $T(n) = n^2 / 2 - n / 2$

  可寫成 $T(n) \approx n^2 / 2$ （Drop low order terms）

  或 $T(n) = O(n^2)$ (Ignore the leading constant)

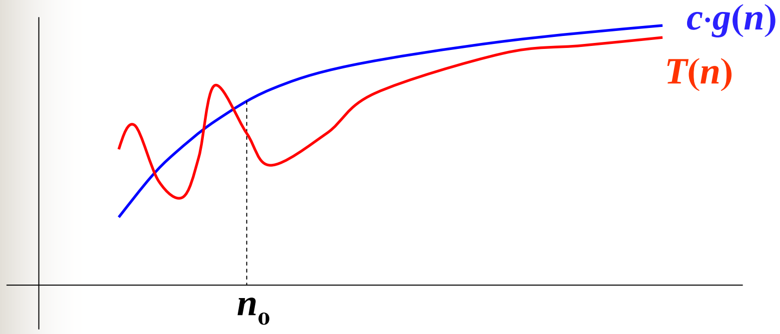
- 好處：有些書寫成： $T(n) \in O(n^2)$
  - 簡化時間複雜度表示法 (就算時間複雜度的單位是基本運算量, 仍可能相當複雜).
  - 時間複雜度的單位為何變不重要.
  - 較容易計算, 若不需精準表示演算法的計算量.

Asymptotic Notation : Big-O

Let $f, g, T, : \mathbb{N} \rightarrow \mathbb{R}^*$. $O(g(n))$ denotes the set

$\{ f(n) : \exists\, c \in \mathbb{R}^+,\ \exists\, n_o \in \mathbb{N} \ni \forall n \geq n_o,\ f(n) \leq c \cdot g(n) \}$

$$\boxed{T(n) = O(g(n)) \cong T(n) \in O(g(n))}$$



Notes on Big-O Notation

- If $T(n) = O(g(n))$, then we say that $g(n)$ is an ***asymptotically upper bound*** for $T(n)$.
- In general, $T(n)$ is positive (and complicated) and $g(n)$ is asymptotically positive (and simple).
- If $T(n) = n^2 / 2 - n / 2$, then we can write :

  $T(n) = O(n^2)$, or $T(n) = O(n^3)$, … , $T(n) = O(n^{100})$, … , $T(n) = O(n^{10000})$, …

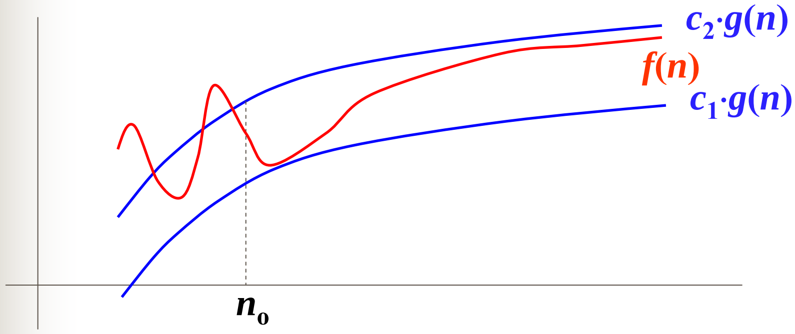
  but $T(n) \neq O(n^{1.99})$, and $O(n^2) \neq O(n^3)$.

Asymptotic Notation : Θ

Let $f, g : \mathbb{N} \to \mathbb{R}^*$. $\Theta(g(n))$ denotes the set of functions

$\{ f(n) : \exists\ c_1, c_2 \in \mathbb{R}^+,\ \exists\ n_o \in \mathbb{N} \ni \forall n \geq n_o,$

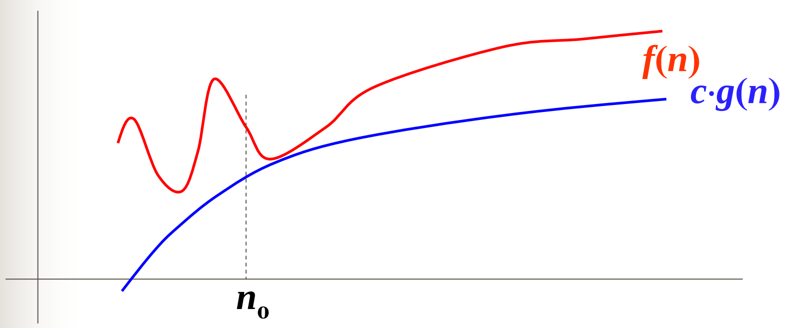
$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \}$



Asymptotic Notation : Ω

Let $f, g : \mathbb{N} \to \mathbb{R}^*$. $\Omega(g(n))$ denotes the set of functions

$\{ f(n) : \exists\ c \in \mathbb{R}^+,\ \exists\ n_o \in \mathbb{N} \ni \forall n \geq n_o,\ 0 \leq c \cdot g(n) \leq f(n) \}$



Notes on Big-O, Θ, and Ω

- Θ- and Ω-notations provide an ***asymptotically tight bound*** and an ***asymptotically lower bound***, respectively.
- If $T(n) = n^2 / 2 - n / 2$, then we can write :

  $T(n) = O(n^2)$, or $T(n) = \Theta(n^2)$, or $T(n) = \Omega(n^2)$.
- $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

  $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$.

  $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.

例： Merge Sort

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n-1,\ T(1) = 0$

假設 $n = 2^k$, 可得： $T(n) = n\lg n - n + 1$

$\lg n = \log_2 n$

對一般 n 可證得：

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

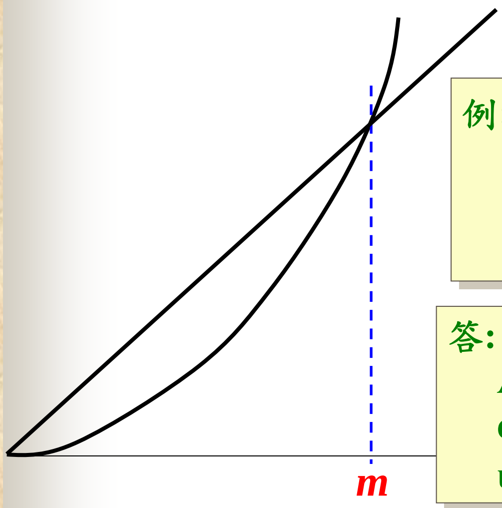
也可以更精確的表示為：

$T(n) = n\lg n - O(n)$ 或 $T(n) = n\lg n - \Theta(n)$

Asymptotic Notation in Descriptions

- ☛ The running time of insertion sort is (in) $O(n^2)$.
  - ≅ The worst-case running time (which is a function of n) of insertion sort is $O(n^2)$.
  - ≅ No matter what particular input of size n is chosen for each value of n, the running time on that set of inputs is $O(n^2)$.
- ☛ The (best-case) running time of insertion sort is $\Omega(n)$.
- ☛ The worst-case running time of insertion sort is $\Theta(n^2)$.

The Limitations of Asymptotic Notations



例: An $O(n)$ time algorithm is faster than an $O(n^2)$ time algorithm ?

答: Asymptotically, yes. Actually, the answer depends on an unknown value m.

Little-oh and Little-omega

- $o(g(n)) = \{ f(n) : \forall\, c \in \mathbb{R}^+,\ \exists\, n_0 \in \mathbb{N}$
  $\ni \forall n \geq n_0,\ 0 \leq f(n) < c \cdot g(n) \} \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- e.g. $2n = o(n^2)$, but $2n \neq o(n)$.
- $\omega(g(n)) = \{ f(n) : \forall\, c \in \mathbb{R}^+,\ \exists\, n_0 \in \mathbb{N}$
  $\ni \forall n \geq n_0,\ 0 \leq c \cdot g(n) < f(n) \} \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$
- $f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$ (a simpler definition)
- e.g. $2n = \omega(\log^k n)$, for $k > 0$.

An Analogy

- Let a and b be two real numbers :

$f(n) = O(g(n)) \approx a \leq b$

$f(n) = \Omega(g(n)) \approx a \geq b$

$f(n) = \Theta(g(n)) \approx a = b$

$f(n) = o(g(n)) \approx a < b$

$f(n) = \omega(g(n)) \approx a > b$

$f(n)$ corresponds to a
$g(n)$ corresponds to b

- 但三一律不成立, 不是任意兩函數 f, g 都滿足
  $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$

Properties of Asymptotic Notations

- ***Transitivity*** **:** For $X = O, \Theta, \Omega, o, \omega$,
  $f(n) = X(g(n)) \wedge g(n) = X(h(n)) \Rightarrow f(n) = X(h(n))$
- ***Reflexivity*** **:**
  $f(n) = O(f(n)),\ f(n) = \Theta(f(n)),\ f(n) = \Omega(f(n))$
- ***Symmetry*** **:**
  $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$
- ***Transpose symmetry*** **:**
  $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
  $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$

Asymptotic Notation & Limits

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0 \Rightarrow f(n) = \Theta(g(n))$$

The reverse is not necessarily correct.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Leftrightarrow f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Leftrightarrow f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n))$$

Some Additional Properties

P1: $T(n) = O(T(n))$

P2: If $c \geq 0$, $d > 0$, $g(n) = O(f(n))$ and $h(n) = \Theta(f(n))$

then, $c\,g(n) + d\,h(n) = \Theta(f(n))$

例: $5n + 3\lg n + 10n\lg n + n^2 = \Theta(n^2)$

P3: If $a > 1$, $b > 1$, then $\log_a n = \Theta(\log_b n) = \Theta(\lg n)$

P4: For any $\varepsilon > 0$, $k > 1$, $\lg^k n = o(n^{\varepsilon})$.

P5: For any $k > 1$, $c > 1$, $n^k = o(c^n)$.

P6: For any $c > 1$, $c^n = o(n!)$.

A Simple Test

(i): $n^2 = O(n^3)$

(ii): $n^3 = O(n^2)$

(iii): $2^{n+1} = \Theta(2^n)$

(iv): $(n+1)! = \Theta(n!)$

(v): $f(n) = O(n) \rightarrow f(n) \times f(n) = O(n^2)$

(vi): $f(n) = O(n) \rightarrow 2^{f(n)} = O(2^n)$

(vii): $\lg^{100000} n = o(n^{0.000001})$.

(viii): $n^{1.01} = O(n \lg n)$

演算法中常見之函數

Big-Oh form	Name
$O(1)$	Constant
$O(\lg n)$	Logarithmic
$O(n)$	Linear
$O(n \lg n)$	$n \log n$
$O(n^2)$	Quadratic, Square
$O(n^3)$	Cubic
$O(n^m)$, $m \geq 1$	Polynomial
$O(c^n)$, $c > 1$	Exponential
$O(n!)$	Factorial

Order of Complexity

Problem size $n =$	2	16	64
$\log n$	1	4	6
n	2	16	64
$n \log n$	2	64	384
n^2	4	256	4096
2^n	4	6.5×10^4	1.84×10^{19}
$n!$	2	2.1×10^{13}	$> 10^{89}$

Exponential explosion

1.84×10^{19} μ secs ≒ ______days ≒ ______centuries

Basic Analysis Methods

- For algorithms ***suitable for recursive implementation***, set up a ***recurrence relation*** and initial condition(s) for $T(n)$ and then solve the recurrence to obtain a closed form or estimate the order of magnitude of the solution.
- For algorithms ***not*** suitable for ***recursive*** (e.g. DP) ***implementation***, set up summation for $T(n)$ reflecting algorithm's loop structure and then simplify summation using standard formulas (see Appendix A).

Recurrences

- A ***recurrence*** is an equation or inequality that describes a function in terms of its value on small inputs, e.g. $T(n) = T(n - 1) + n$, $T(1) = 0$.
- Methods for solving recurrences:
  - The recursion-tree method
  - Changing variables
  - The substitution method
  - The master method
  - Other methods discussed in discrete math.

Technicalities

- If we only want to find an asymptotical bound for $T(n)$, sometimes we can neglect certain technical details such as integer argument assumption, boundary condition, floors, ceilings,… etc., for example:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n), \text{ for } n > 1$$
$$T(1) = \Theta(1)$$

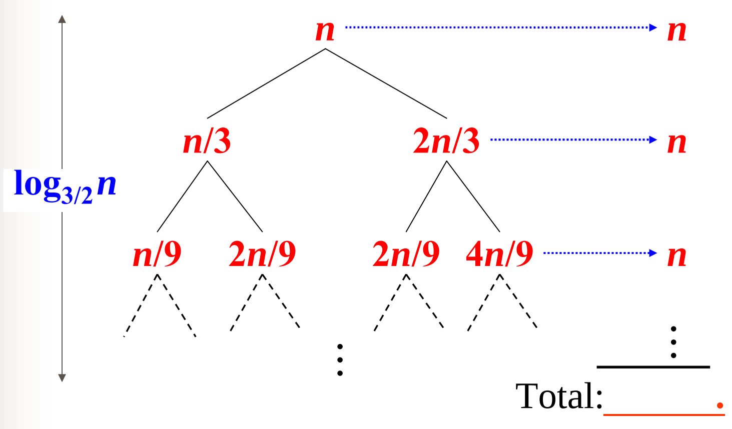
$$\downarrow$$

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

We forge ahead without these details and later determine whether or not they matter.

The recursion-tree method

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



Changing variables

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n-1,\ T(1) = 0$$

Assume $n = 2^k$

$$T(n) = 2T(n/2) + n-1,\ T(1) = 0$$

$$T(2^k) = 2T(2^{k-1}) + 2^k-1,\ T(1) = 0$$

Let $a_k = T(2^k)$

$$a_k = 2a_{k-1} + 2^k - 1,\ a_0 = 0$$

$$a_k = k2^k - 2^k + 1 \longrightarrow T(n) = n\lg n - n + 1$$

The substitution method

- The method entails two steps:
  1. Guess the form of solution.
  2. Use mathematical induction to show it works.
- It is the most rigorous among these methods.
- An example: $T(n) = T(n/3) + T(2n/3) + O(n)$
  A guess: $T(n) \leq d\, n\lg n$, where d is a suitable positive constant (i.e. we need to show that d exists.)

The substitution method (例)

By inductive hypothesis

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

Goal

As long as $d \ge c/(\lg 3 - (2/3))$.

$\therefore$ $T(n) = O(n\lg n)$.

Avoiding pitfalls

- Solve $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- Assume $T(n) \le O(n)$
- We want to prove: $\exists c \forall n \; T(n) \le cn$
- By induction:
  $T(n) \le 2c\lfloor n/2 \rfloor + n \le cn + n = O(n)$
  (since c is a constant.)
- (***WRONG!***) You cannot find such a c.

Stirling's Formula (or Approximation)

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1+\Theta\left(\frac{1}{n}\right)\right)$$

- $c^n = o(n!)$, $c > 1$; $n! = o(n^n)$,
- $\log(n!) = \Theta(n \log n)$
- nth Catalan number $= C(2n, n)/(n+1) = \Omega(4^n/n^{3/2})$

Approximation by Integrals (p.1155)

If $f(n)$ is a *monotonically increasing* function, then

$$\int_{a-1}^{n} f(x)dx \le \sum_{k=a}^{n} f(k) \le \int_{a}^{n+1} f(x)dx$$

If $f(n)$ is a *monotonically decreasing* function, then

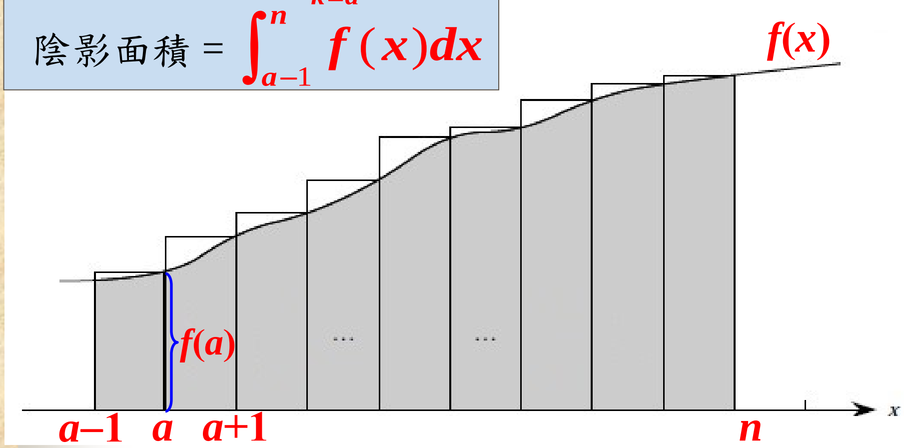
$$\int_{a}^{n+1} f(x)dx \le \sum_{k=a}^{n} f(k) \le \int_{a-1}^{n} f(x)dx$$

- e.g. $\log(n!) = \Theta(n \log n)$

Approximation by Integrals (reasoning)

長方形總面積 = $\sum_{k=a}^{n} f(k)$

陰影面積 = $\int_{a-1}^{n} f(x)dx$



The Master Theorem (p. 94)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined 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 $T(n)$ can be bounded asymptotically as follows.

1. $f(n) = O(n^{\log_b a - \varepsilon})$, $\varepsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$
2. $f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$
3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$, $\varepsilon > 0$, and $af(n/b) < cf(n)$ for $c > 1$ and all sufficiently large $n \Rightarrow T(n) = \Theta(f(n))$

A simple version of the master theorem

Let $T(n)$ be defined by the recurrence :

$$T(n) = aT(n/b) + \Theta(n^k), \quad \text{for } n > n_0$$
$$T(n) = O(1) \quad \text{for } n \le n_0$$

$a \ge 1, b > 1, k \ge 0$

Then $T(n)$ can be bounded asymptotically as follows.

1. $T(n) = \Theta(n^{\log_b a})$ if $k < \log_b a$,
2. $T(n) = \Theta(n^k \lg n)$ if $k = \log_b a$,
3. $T(n) = \Theta(n^k)$ if $k > \log_b a$,

A Recurrence Equation for D&C

$$T(n) = aT(n/b) + cn^k, \quad \text{for } n > n_0$$
$$T(n) \le d \quad \text{for } n \le n_0$$

☛ 代表將問題分解 (假設等分的話) 成 a 塊每塊大小為 n/b (取上或下高斯符號). 例：

- Binary search ($a = 1, b = 2, k = 0$)
- Merge sort ($a = 2, b = 2, k = 1$)
- Quicksort (不合等分假設)
- Tree traversals (不等分, 但 $k = 0$)

矩陣相乘

- 給兩個 $n \times n$ 矩陣 A, B 要計算它們的乘積: $C = AB$
- 傳統做法需 $\Theta(\_\_)$ 的計算時間.
- 想法：每個矩陣各分成四個 $n/2 \times n/2$ 矩陣.

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

$$\boxed{T(n) = 8T(n/2) + \Theta(\_\_)} \xrightarrow[\log_b a = 3]{a = 8,\ b = 2,} \boxed{T(n) = \Theta(\_\_)}$$

矩陣相乘 (續1)

- 想辦法降低遞迴公式中的 8

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

- Strassen’s method :

$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$, $M_2 = (A_{21} + A_{22})\, B_{11}$

$M_3 = A_{11}(B_{12} - B_{22})$, $M_4 = A_{22}(B_{21} - B_{11})$

$M_5 = (A_{11} + A_{12})\, B_{22}$, $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$

$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

$$\boxed{C = \begin{bmatrix} M_1+M_4-M_5+M_7 & M_3+M_5 \\ M_2+M_4 & M_1+M_3-M_2+M_6 \end{bmatrix}}$$

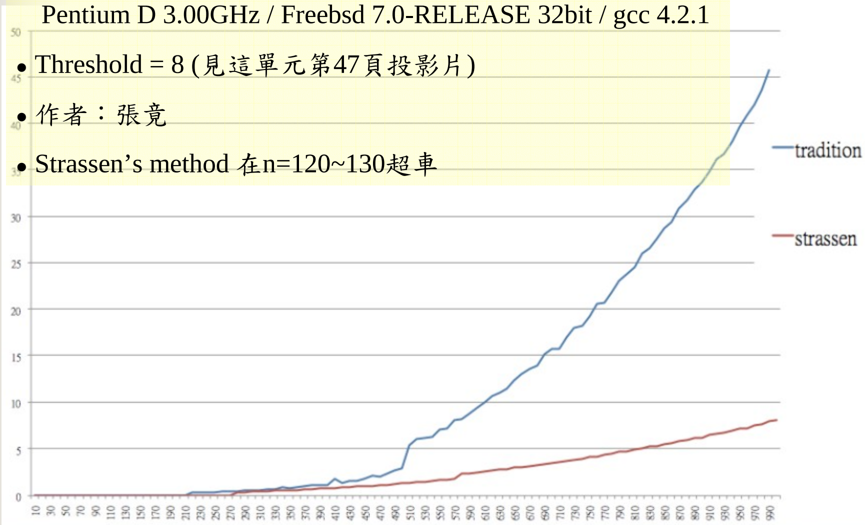
矩陣相乘 (續2)

$\boxed{T(n) = 7T(n/2) + \Theta(n^2)} \xrightarrow[\log_b a = \lg 7]{a=7,\ b=2,} \boxed{T(n) = \Theta(\_\_) = O(\_\_)}$

- 可進一步改進（再細分; Ex.4.2-4,5 p.82）
- 世界記錄： $O(n^{2.376})$
- 以下幾個問題也可在同一時間複雜度內解決
  - Given A, b find x s.t. $Ax = b$ (that is why the title of Strassen’s paper is “Gaussian elimination is not optimal”)
  - Given A find A^{-1}, det(A).

矩陣相乘 (一個實驗結果)

- 測試環境:
  Pentium D 3.00GHz / Freebsd 7.0-RELEASE 32bit / gcc 4.2.1
- Threshold = 8 (見這單元第47頁投影片)
- 作者：張竟
- Strassen’s method 在n=120~130超車



長整數相乘

- 給兩個 n 位數整數要計算它們的乘積.
- 傳統做法需 $\Theta(\_\_)$ 的計算時間.
- 想法：每個整數各分成兩個 $n/2$ 位數的整數.

u	w	x		v	y	z

$$uv = (w\,\mathrm{d}^{n/2} + x)(y\,\mathrm{d}^{n/2} + z) = wy\,\mathrm{d}^{n} + (wz + xy)\,\mathrm{d}^{n/2} + xz$$

$$T(n) = 4T(n/2) + \Theta(\_) \xrightarrow[\log_b a = 2]{a = 4,\ b = 2,} T(n) = \Theta(\_)$$

長整數相乘 (續)

- 想辦法降低遞迴公式中的 4

$$T(n) = 4T(n/2) + \Theta(n)$$

- 做法：

$$uv = wy\,\mathrm{d}^{n} + (wz + xy)\,\mathrm{d}^{n/2} + xz$$

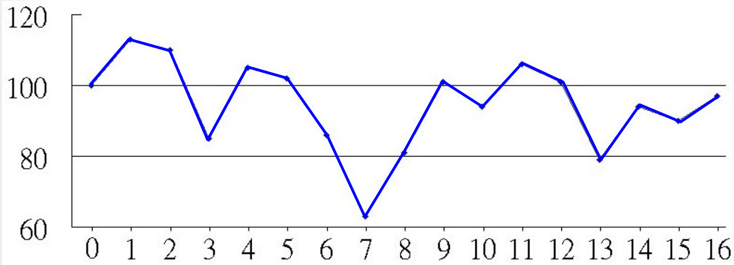
1. Compute $r = (w + x)(y + z) = wy + (wz + xy) + xz$
2. Compute $(wz + xy) = r - wy - xz$

$$T(n) = \_\,T(n/2) + \Theta(n) \xrightarrow[\log_b a = \lg 3]{a = 3,\ b = 2,} \begin{aligned} T(n) &= \Theta(\_\_) \\ &= O(\_\_) \end{aligned}$$

- 除法同級；可進一步改進（再細分）
- 世界記錄： $\Theta(n \log n \log\log n)$

A Stock Buying Problem (p.68)

- You have the prices that a stock traded at over a period of ***n*** consecutive days.
- When should you have bought and sold the stock such that you could get the maximal profit.



Day	0	1	2	3	4	5	6	7	8	9	10	11	12
Price	100	113	110	85	105	102	86	63	81	101	94	106	101

A Formal Description of the Problem

- Given an array of numbers $p[0..n]$, compute

  $\max_{i<j} (p[j] - p[i])$

  (and find the indices if they are needed.)
- If $p[0] > p[1] > \cdots > p[n]$, then 0 is a reasonable solution. (i.e. just don't buy at all.)
- A brute-force method needs________time.

A Transformation to MSP

- Transform the array $p[0..n]$ to another array $a[1..n]$ where $a[i] = p[i] - p[i-1]$ and compute

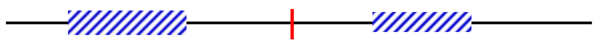
  $\max_{i \le j} \{a[i] + a[i+1] + \cdots + a[j]\}$.

- It is called *the maximum subarray (sum) problem* (MSP). For example:

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

Use D&C to Solve MSP

- Divide the array $a[1..n]$ into $a[1..n/2]$ and $a[n/2+1..n]$. Then the maximum sub-array $a[i..j]$ must lie:
  - entirely in $a[1..n/2]$ or $a[n/2+1..n]$ (case 1)
  - crossing the midpoint (case 2)

Case1: 

Case2: 

Pseudo-Code for the D&C Algorithm

```
F(a[ ], ℓ, r)
  {
    if (ℓ == r)  return a[ℓ];
    m  =  (ℓ+r)/2 ;
    L = F(a[ ], ℓ, m);
    R = F(a[ ], m+1, r);
    C = FIND-MAX-CROSSING-SUBARRAY(a[ ], ℓ, r);
    return max(L, R, C);
  }
```


Analyzing the D&C Algorithm

- The solutions of both left and right parts are not used for solving Case 2.
- Case 2 is solved by a 2-way scanning procedure from the middle and uses $\Theta(n)$ time (p.71).
- $\therefore\ T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n\lg n)$.
- Exercise: Directly solve the original stock buying problem in $\Theta(n)$ time. (Note : Ex. 4.1-5 in p.75 asks you solve MSP in $\Theta(n)$ time.)

An Improvement on D&C

- In a D&C algorithm, it is advisable to use a simpler algorithm when subproblems become small enough.
- For example, we may use insertion sort within merge sort when the sizes of subproblems are no greater than a given ***threshold*** t.

Determining Thresholds for D&C

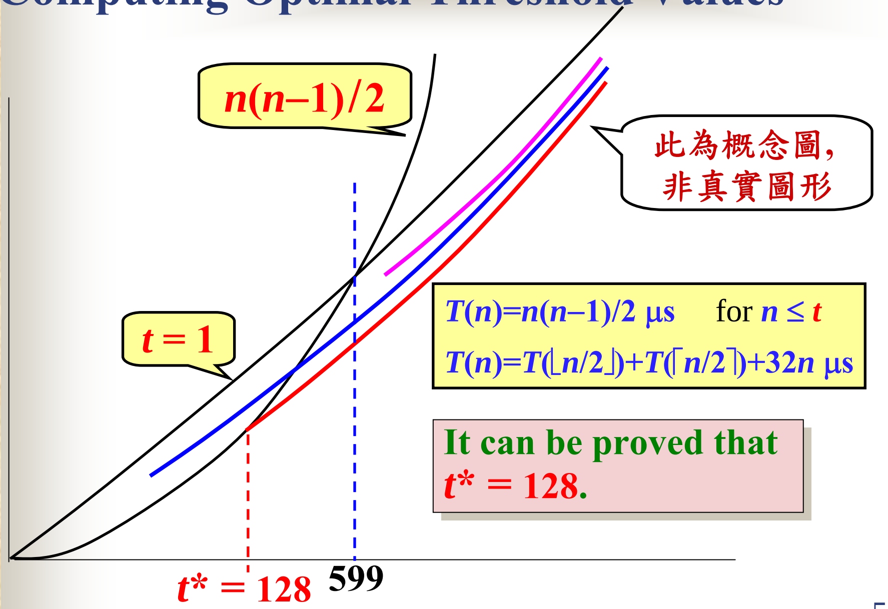
- For merge sort, assume that

$$T(n) = n(n-1) / 2\ \mu s \qquad \text{for } n \le t$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 32n\ \mu s \qquad \text{for } n > t$$

- For any threshold value t, $T(n) = \Theta(n \lg n)$.
- However, we want to find an ***optimal threshold value*** t^*.

Computing Optimal Threshold Values



$T(n)=n(n-1)/2$ μs for $n \le t$

$T(n)=T(\lfloor n/2 \rfloor)+T(\lceil n/2 \rceil)+32n$ μs

It can be proved that $t^* = 128$.

Notes on Optimal Threshold Values

- Optimal threshold values **may not exist**.
- It may be more appropriate to determine an optimal threshold value by **doing experiments** (a kind of tuning a program).