科目：演算法

教科書：**T.H. Cormen et al.** ,
"**Introduction to Algorithms**", **3rd ed.,**
The MIT Press, 2009.
（1292頁35章＋4附錄；進口商:開發）
教師：何錦文
單位：國立中央大學 資工系

參考書目：

- R. Sedgewick, Algorithms in C++, Addison-Wesley.
- A. Levitin, Introduction to the Design & Analysis of Algorithms, Addison Wesley.
- R.E. Neapolitan & K. Naimipour, Foundations of Algorithms using C++ pseudocode, Jones and Bartlett.

相關網頁

- 學校數位學習平台 ee-class 系統：本課程討論區與作業、文件等公告區；發表訊息時請用真實姓名並加學號）
- http://uva.onlinejudge.org/
  There are hundreds of problems that are like the ones used during programming contests (with ONLINE JUDGE).

程式競賽或檢定網頁

- **教育部年度全國大專電腦軟體設計競賽**
- 大學程式能力檢定 (Collegiate Programming Examination, CPE)
  http://cpe.cse.nsysu.edu.tw/
  http://acm-icpc.tw/cpe/
- 「ITSA線上程式設計大賽」
  http://algorithm.csie.ncku.edu.tw/ITSA/
- PTC線上競賽
  http://ptc.moe.edu.tw/

Unit 1
Introduction

T.H. Cormen et al., "Introduction to Algorithms", 3rd ed., Chapters 1-2.

Outlines

- What are algorithms?
- Problems and modeling.
- Training for an algorithm designer:
  - Design (strategies for solving problems).
  - Analysis.
  - Implementation.
  - Some related topics.

What are Algorithms

- An *algorithm* is a *sequence of unambiguous instructions* for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- *Problem solving methods* suitable for implementation as computer programs.

What are Problems

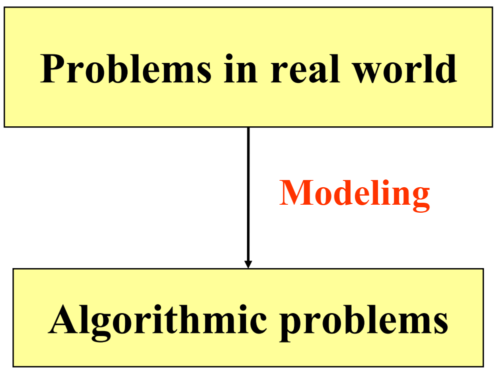
- Problems considered here are *algorithmic problems* (or *well-specified computational problems*).
- There are 3 components of a problem:
  - A set of inputs (or instances).
  - A set of outputs (or solutions, answers).
  - A ***precise description*** of the relation between inputs and outputs.

Inputs and outputs are all of finite lengths.

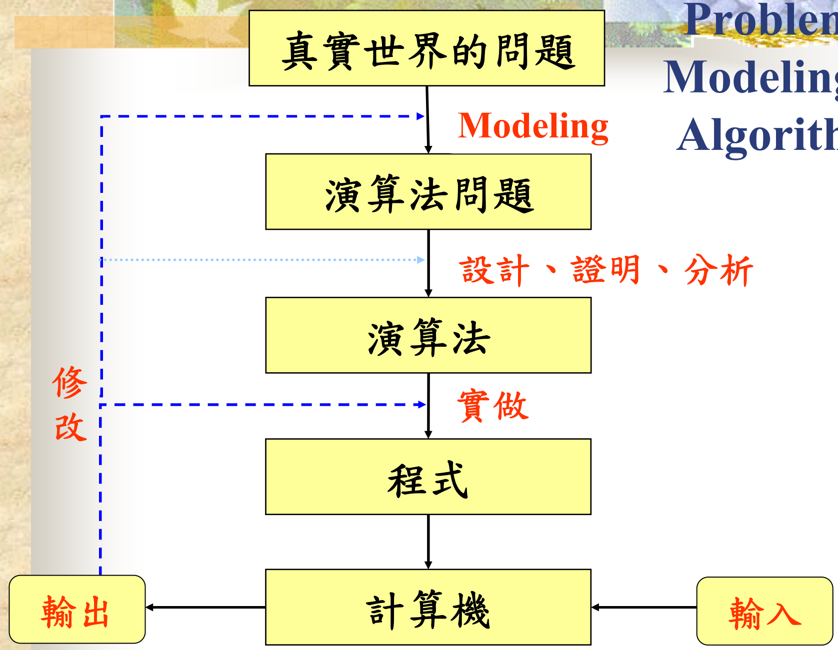
An Example : Sorting

- Statement of problem:
  - *Input:* A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
  - *Output:* A reordering of the input sequence $\langle a'_1, a'_2, \ldots, a'_n \rangle$ so that $a'_i \le a'_j$ whenever $i < j$
- For example: The sequence ⟨5, 3, 2, 8, 3⟩
- Algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
  - And many others …

Problems & Modeling



Problems , Modeling, & Algorithms



Problems & Modeling 範例：字串比對

- 給兩個字串要決定它們之間的相似程度
  e.g. “ABCBDAB” “BDCABA”
- 問題可能來源
  - 語音辨識 Model 1
  - 文字編輯器 Model 2
  - 生物資訊比對 Model 3
  - 抓抄襲

Model 1: LCS (Longest Common Subsequences)

- 用於語音辨識
- 例：

  Input: ABCBDAB BDCABA

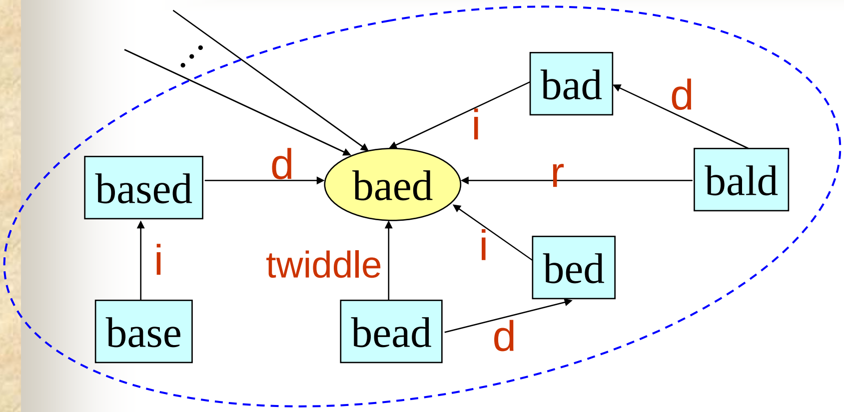
  C.S.'s: AB, ABA, BCB, BCAB, BCBA …

Longest: BCAB, BCBA, … Length = 4

A B C B D A B

B D C A B A

Model 2: Edit Distance (用於文字編輯器)



- 上圖中，兩個字串之間的最短路徑的長度即為它們的 ***edit distance***

Model 3 : Alignment of Two Strings

- An *alignment* of strings S and T is a pair of strings ($S^\star$, $T^\star$) obtained by insertion of spaces in S and T such that
  1. $|S^\star| = |T^\star|$
  2. For each i, $S^\star[i]$ is aligned with $T^\star[i]$, and either $S^\star[i]$ or $T^\star[i]$ is not a space.

$S^\star$: `v-intner-`	$S^\star$: `v--intner-`
✓	✗
$T^\star$: `wri-t-ers`	$T^\star$: `wr-i-t-ers`

The Score of an alignment

- **Scoring Matrix**:
  - $s(x,y)$: the score by aligning x with y
  - $s(x,y) \geq 0$ if $x=y$; otherwise, $s(x,y) \leq 0$. (emphasize matches, penalize mismatches or inserted spaces)
- The score of an alignment of S,T:

  Let $|S^\star|=|T^\star|=L$.

  $s(S^\star,T^\star) = \Sigma_{(1\leq k \leq L)} s(S^\star[k],T^\star[k])$

Similarity of Two Strings

- *Similarity* of *S*, *T*: the maximum value of scores of all possible alignments of *S* and *T*.
- Examples:

s	a	b	c	d	-
a	1	-1	-2	0	-1
b		3	-2	-1	-1
c			0	-4	-2
d				3	-1

```
c a c d b d
c a b b d b
```
Score = 0+1-2-1-1-1 = -4

```
c a c - d b d
c a b b d b -
```
Score = 0+1-2-1+3+3-1 = 3

Important Problem Types

- Sorting
- Searching
- String processing
- Combinatorial problems
- Graph problems
- Geometric problems
- Algebraic problems
- Numerical problems

Studying objects of discrete nature

Studying continuous objects

演算法基本設計方式

- 觀察、直覺 & 試誤法（trial and error）
- 遞迴設計法（數學歸納法）
- 轉換法 (Transform-and-Conquer; 將要解的問題轉成你會解的問題)
- 其他：Brute Force, Space and Time Tradeoffs, Primal-Dual, Line Sweep, Graph Traversal, FFT-Based, Amortized Analysis, … etc.

遞迴方式設計演算法

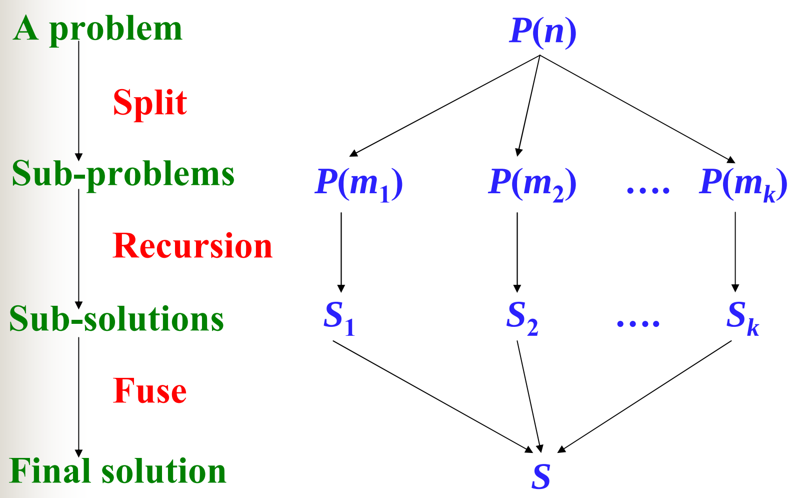
包括一般書上及本書提到的：

- Divide-and-Conquer
- Greedy Algorithms
- Decrease-and-Conquer (Prune-and-Search)
- Dynamic Programming
- Iterative Improvement (e.g. Max-Flow)
- Branch-and-Bound
- Backtracking

How to Express Algorithms

- Programs
- Natural languages
- Flowcharts
- Pseudocodes

遞迴方式設計概念



Notes

(1) $m_1 < n,\ m_2 < n,\ \dots, m_k < n,$

(2) $k \geq 1$

(3) Split & Fuse 可能會是另外兩個問題

(4) $m_1 + m_2 + \dots + m_k$ 可能 $<, =,$ 或 $> n$

(5) 分析訣竅： (有些情形不適用)

$$T(n) = T(m_1)+T(m_2)+ \dots + T(m_k)+S(n)+F(n)$$

遞迴設計：例 1—Selection Sort

[7, 2, 9, 6, 5] $P(5)$

2, [7, 9, 6, 5] $P(4)$

2, 5, [9, 6, 7] $P(3)$

2, 5, 6, [9, 7] $P(2)$

2, 5, 6, 7, 9

Algorithmic Description of Selection Sort

$P(n)$ Input: A[1], A[2], … ,A[n-1], A[n]

$P(n_o)$ S0: If $n \le 1$, do nothing and return

Split { S1: Find a min A[i] in A[1], A[2], … , A[n]

S2: Exchange the values of A[1] and A[i]

$P(n-1)$ S3: Recursively sort A[2], … , A[n]

Fuse is trivial

Analysis of Selection Sort

- Correctness: By induction.
- Time complexity :

$T(n)$ = # comparisons

$T(1) = 0$

$T(n) = T(n-1) + n-1$

$\therefore\ T(n) = n-1 + n-2 + \ldots + 1 =$ ______.

概念類似的排序法：

_______Sort

_____ Sort

Bubble Sort

[7, 2, 9, 6, 5] ***P*(5)**

2, 7, 9, 6, 5

2, 7, 9, 6, 5

2, 7, 6, 9, 5

[2, 7, 6, 5] 9 ***P*(4)**

遞迴設計：例 2—Mergesort

[7, 2, 9] [6, 5, 8]

[2, 7, 9] [5, 6, 8]

2, [7, 9] [5, 6, 8]

2, 5, [7, 9] [6, 8]

2, 5, 6, [7, 9] [8]

2, 5, 6, 7, [9] [8]

Algorithmic Description of Mergesort

$P(n)$ Input: A[1], A[2], … ,A[n-1], A[n]

$P(\lfloor n/2 \rfloor)$ S1: Recursively sort A[1], … , A[$\lfloor n/2 \rfloor$]

$P(\lceil n/2 \rceil)$ and A[$\lfloor n/2 \rfloor$+1], … , A[n]

Fuse S2: Merge the two sorted lists

Split is trivial (for array case)

Analysis of Mergesort

- Correctness: By induction
- Time complexity :

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

$M(n) = M(n\text{-}1) + 1$, $M(1) = 0$ /* n = total length of the two lists */

假設 $n = 2^k$, 可得：

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

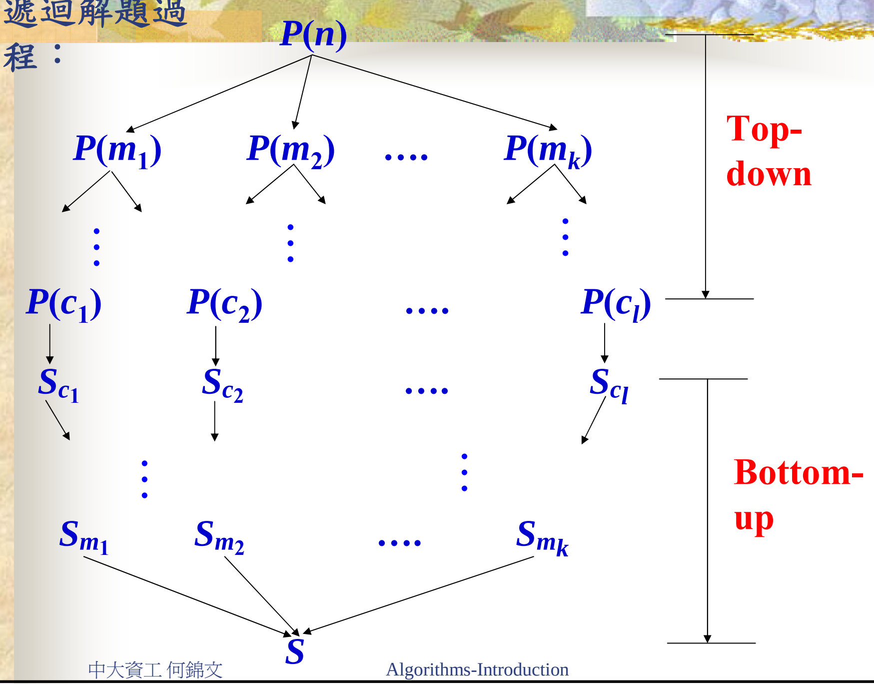
Recursive Programming

例：**Mergesort** (array case)

$P(n)$ mergesort(a[ ], ℓ, r)
{
$P(n_0)$ if (r > ℓ) {
m = (r+ ℓ)/2 ;
$P(\lfloor n/2 \rfloor)$ mergesort(a[ ], ℓ, m);
$P(\lceil n/2 \rceil)$ mergesort(a[ ], m+1, r);
Fuse merge(a[ ], ℓ, r);
}
}

This sort is stable; however it is not in-place.

遞迴解題過程：



若 split 為 trivial, Top-down 過程可以省略

例—Mergesort

[7] [2] [9] [6] [5] [8] [1]

[2 7] [6 9] [5 8] [1]

[2 6 7 9] [1 5 8]

[1 2 5 6 7 8 9]

省略 Top-down 例子二：Insertion Sort

7	2	9	6	5
[7]	2	9	6	5
[2	7]	9	6	5
[2	7	9]	6	5
[2	6	7	9]	5
2	5	6	7	9

```
for (i = 2; i <= N; i++)  {
   v = a[i];     j = i;
   while ( j > 1 && a[j-1] > v)  {
        a[j] = a[j-1];    j--;
   }
   a[j] = v;
}
```

Sentinel key :若設定 $a[0] = -\infty$
畫底線部份可去掉

Analysis of Insertion Sort

- **Correctness: By induction**
- **Time complexity :**

$T(1) = 0$

$T(n) = T(n\text{-}1) + f(n\text{-}1)$

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

Worst case: $f(n\text{-}1) = n\text{-}1 \Rightarrow T(n) =$ _______.

Average case: $f(n\text{-}1) \approx n/2 \Rightarrow T(n) \approx$ _____.

not $(T_{worst} + T_{best})/2$

Analysis of an Algorithm

- **Correctness.**
- **Resources (time, space, ...) consumed.**
  - **Complexities (worst, average, best cases).**
- **Does there exist a better algorithm ? (lower bounds, optimality)**

Analysis的必要性

- 分析演算法是設計者的職責.
- 對所設計的演算法有一完整的瞭解, 以利以後改進, 或是培養解題能力.
- 選一適當的演算法
- 適當（達到要求）的演算法不一定是要最快的
  - 因為較快的演算法通常較複雜，不容易 implement，容易出錯.
  - 同理，program 也一樣（指 implement 同一演算法）.

Implementation of an Algorithm

- Selecting a suitable data structure & coding (名言： program = algorithm + data structure.)
- Empirical analysis
- Program optimization (recursion-removal, sentinel, tuning, ...)

學習演算法的境界

- 最高境界：能獨立完成演算法的設計、分析、以及實做.
- 最低境界：當有人告訴你演算法的作法或改進方法，你能將此演算法實做出來.

Some related issues

- Mathematics (主要是 discrete)
  Mathematics appears everywhere in modeling, algorithm design and analysis.
- Data structures
- Computational theory

Data Structures

- Data structures are the heart of any sophisticated program.
- Some algorithm can only work on some special data structure.
  e.g. Binary search $\leftrightarrow$ Array, and …
- Selecting the right data structure can make an enormous difference in the complexity of the resulting implementation.

Related Topics in Computational Theory

- *Machine models*
- *Decidable* $\leftrightarrow$ *Undecidable*
- *Lower bounds*
- *Tractable* $\leftrightarrow$ *Intractable* **problems**
  (easy) **(difficult)**

A notorious one: NP-Complete

How to Handle Intractable Problems

(~)**E**: (non-)Exact algorithms

- Branch-and-bound (**E**)
- Heuristic algorithms (~**E**)
- Randomized algorithms : simulated annealing (~**E**), genetic algorithms (~**E**), probabilistic algorithms (**E** or ~**E**)
- Approximation algorithms (~**E**)
- Fixed-parameter algorithms (**E**)
- Other models of computation : quantum, DNA, parallel, neural nets…
- …