算法

單元三

+ Outline

- 算法 (Algorithm)
  1. 搜索問題
  2. 排序問題
  3. 優化問題
- 算法複雜度 (Algorithm Complexity)

+ 算法 Algorithm (3.1)

- 算法(Algorithm): 接收有效輸入值 (Input)並產生所需輸出值(Output)的包含**<u>指定步驟</u>**的一系列流程。

+ 範例

- 描述一個能在有限序列的整數中搜索最大值的算法:
  1. 把第一個數(1st #)視為暫時的最大值(Max)
  2. 把 Max 跟第二個數(2nd #)比較大小, 若 2nd # 比 Max 大, 則 Max := 2nd #
  3. 重覆直至比較所有值, max中的值為所求最大值。

- 偽代碼 (Pseudocode):

procedure $max(a_1, a_2, \ldots, a_n$: integers)

  $max := a_1$

  for $i := 2$ to n

    if $max < a_i$ then $max := a_i$

  return max {max is the largest element}

- E.g. 11, 23, 4, 67, 9

+ 偽代碼 (Pseudocode)

- 定義:
  Procedure algorithm name (list and properties of variables)
- 注釋: {comment}
- 賦值: variable := expression
- 條件結構:
  - **if** condition **then** statement or block of statements
  - **if** condition **then** statement 1
    else statement 2
  - **if** condition 1 **then** statement 1
    else if condition 2 **then** statement 2
    else if condition 3 **then** statement 3 …

+ 偽代碼 (Pseudocode)

- 循環結構:
  - **for** variable := initial value **to** final value
    statement or block of statements
  - **for** elements with a certain property
    statement or block of statements
  - **while** condition
    statement or block of statements

- 返回語句: **return** output of algorithm

更多偽代碼請參考附錄**C (Appendix 3)**

+ 利用算法解決問題的例子

- 比如以下三類:
    一. 搜索算法: 搜索特定元素在列表中的位置
    二. 排序算法: 把元素由小到大作排序
    三. 優化算法: 確定所有**Input**的可能值中的最優值(最大值或最小值)。

+ 一、搜索算法 Searching Algorithms

- 一般的搜索問題是在一含不同元素 $a_1,a_2,\ldots,a_n$ 的列表中定位元素 x，或者確認它不在列表中。
  - 如: 圖書館在允許某人借閱另一本書之前可能想要先檢查Ta是否在書籍過期的名單中。

一、線性搜索 Linear Search

- 搜索特定元素 **x** 在列表中的位置的算法:

如: (**x** = 9) 4, 7, 3, 2, 1, 0, 9

1. 預設位置為1, 比較 x 和 a_1, 若不相等, 位置+1;
2. 繼續比較 x 和 a_2, 若不相等, 位置+1;
3. 重覆直至比較所有值直到找到相等的值, 若最後沒有相等的即傳回 0。

procedure *linear search* (x:integer, a_1, a_2, …,a_n: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
  $i := i + 1$
if $i \leq n$ **then** *location* := i
else *location* := 0
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

+ 一、二分搜索 Binary Search

- 在一個由小到大排列的列表中以比較中間位置數值來搜索 x 的算法:

如: (x = 9)　　0, 1, 2, 3, 4, 7, 9

1. 將要找的元素 x 與中間元素進行比較。如果中間元素較小，則搜索會跳到列表的上半部分繼續；否則搜索從列表的下半部分繼續(包括中間位置)。
2. 重複此過程，直到我們有一個大小為 1 的列表。如果我們要查找的元素與列表中的元素相等，則傳回對應位置。否則，傳回 0 以表示未找到該元素。

procedure binary search(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {i is the left endpoint of interval}
$j := n$ {j is right endpoint of interval}
while $i < j$
　　$m := \lfloor (i + j)/2 \rfloor$
　　if $x > a_m$ then $i := m + 1$
　　else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{location is the subscript i of the term a_i equal to x, or 0 if x is not found}

+ Linear Search vs. Binary Search

0, 1, 2, 3, 4, 7, 9

+ 二、排序算法 Sorting Algorithms

- 用以對一列表進行排序
  - 如: 電話、價格、用戶名等排序

二、冒泡排序 Bubble Sort

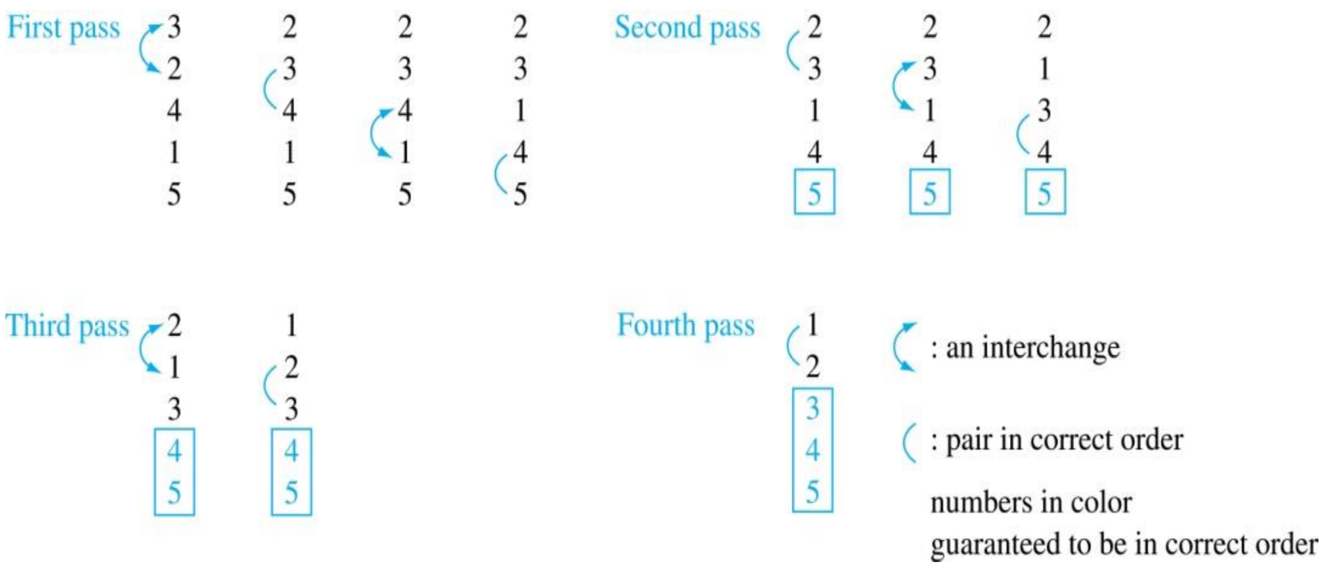
- 通過比較相鄰元素並交換順序不對的元素作排序。

procedure *bubblesort*($a_1,\ldots,a_n$: real numbers
with $n \geq 2$)

  for $i := 1$ to $n - 1$

    for $j := 1$ to $n - i$

      if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1} $\{a_1,\ldots, a_n$ is now in increasing order$\}$



二、插入排序 Insertion Sort

- 通過不斷比較前方的元素大小並插入至正確順序位置排序。

procedure *insertion sort* ($a_1,\ldots,a_n$: real numbers with $n \geq 2$)
  for j := 2 to n
    $i := 1$
    while $a_j > a_i$
      $i := i + 1$
    $m := a_j$
    for k := 0 to $j - i - 1$
      $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{Now $a_1,\ldots,a_n$ is in increasing order}

+ Bubble Sort vs. Insertion Sort

42, 19, 32, 11, 8

+ 三、貪婪算法 Greedy Algorithms

- 根據定義何謂“最優”，算法會在每一步都選擇“最優”方案。
  - 如: 在一段路線上選擇最短路程。

三、找零錢的貪婪算法 Greedy Change-Making Algorithm

- 以美元的硬幣為例, 利用最少硬幣數找零: (其中: $c_1 = 25$, $c_2 = 10$, $c_3 = 5$, and $c_4 = 1$)

procedure *change*($c_1, c_2, \ldots, c_r$: values of coins, where $c_1 > c_2 > \ldots > c_r$; n: a positive integer)

for $i := 1$ to r

  $d_i := 0$ [d_i counts the coins of denomination c_i]

  while $n \geq c_i$

    $d_i := d_i + 1$ [add a coin of denomination c_i]

    $n = n - c_i$

[d_i counts the coins c_i]

+ 函數的增長 Growth of Functions (3.2)

- 在計算機科學和數學領域，很多時候我們會在意函數的增長速度。
- 計算機科學中，我們想知當輸入的**Input**增加，一個算法運算速度的變化。
  - 比較兩個解決同樣問題的算法的效率
  - 確定一個算法在輸入值變多時是否實際上可用

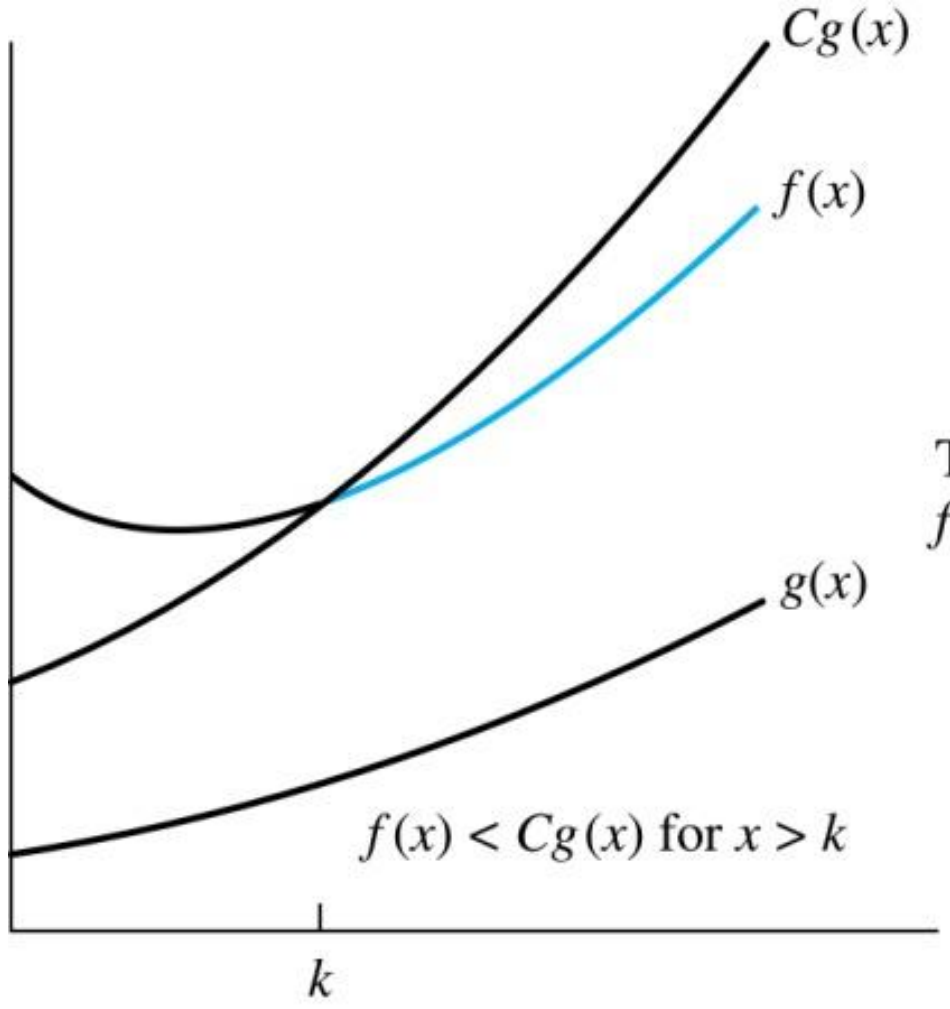
+ 大O記號 Big-*O* Notation

- 稱 $f(x)$ 是 $O(g(x))$ 若存在常數 C 和 k 令每當 $x > k$ 都有

$$|f(x)| \leq |C(g(x))|$$

  - 讀作 "$f(x)$ is big-O of $g(x)$"

Big-O Notation



The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

例1

A. 證明函數$f(x) = 4x^2 - 8x + 1$是 $O(x^2)$ 的；

B. 判斷函數 $\log(n+1)$ 和 $\log(n^2+1)$ 是否是 $O(\log n)$ 的。

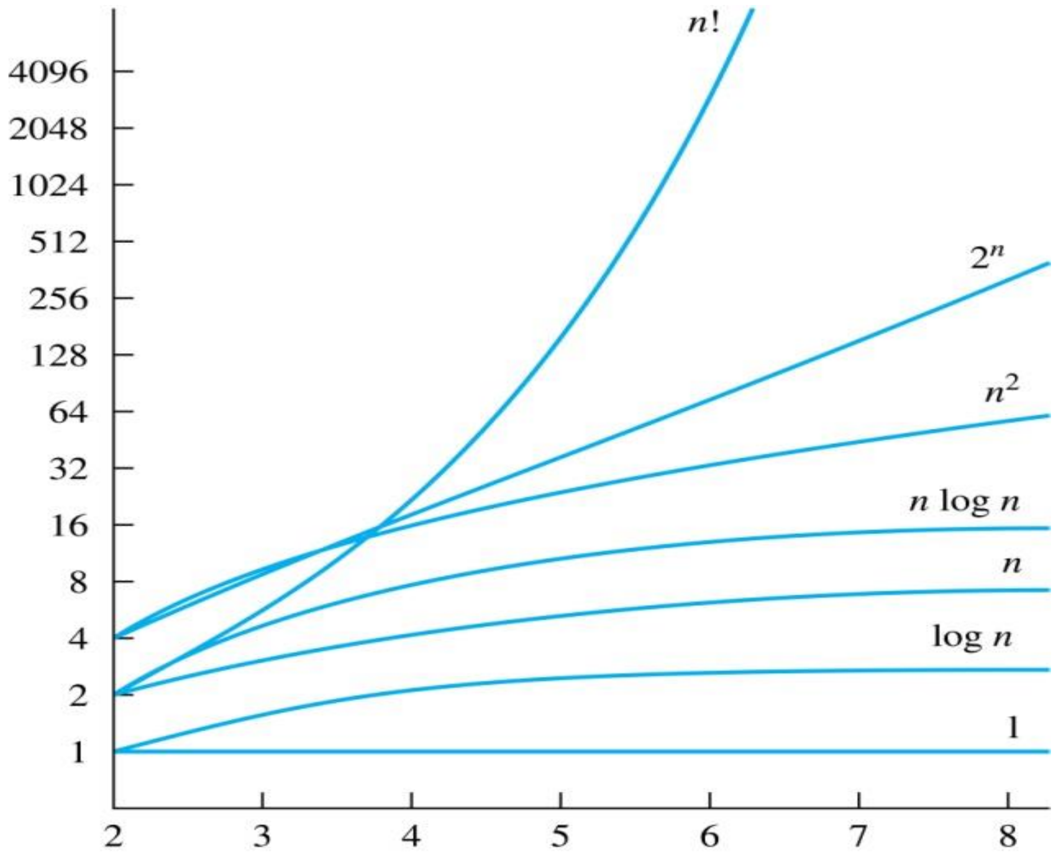
For $n \in \mathbf{N}$:

- $If\ f_1(n)\ is\ O\big(g_1(n)\big), f_2(n)\ is\ O(g_2(n)),\ \ then\ (f_1 f_2)(n)\ is\ O(g_1 g_2(n))$.
- $If\ f_1(n)\ is\ O\big(g_1(n)\big), f_2(n)\ is\ O\big(g_2(x)\big),\ \ then\ (f_1 + f_2)(n)\ is\ O(\max(g_1(n), g_2\,(n)))$.

+ 常見的函數大O估算及其排序

1(常數) < log *n* < *n* < *n* log *n* < n^2 < 2^n < *n*!

- 當 *n* 越大, 各類函數的增長變化:



例2

- 求下列函數的最佳大O估算: (請用階最小的簡單函數)
  A. $(n^2+8)(n+1)$
  B. $(n\log n+n^2)(n^3+2)$
  C. $(n!+2^n)(n^3+\log(n^2+1))$

+ 算法的複雜度 Complexity of Algorithm (2.3)

- 運行一個算法需時多久?
  - 通過計算運算步驟的量

- 算法分析步驟:

1. 準確描述算法步驟
2. 定義大小為 n 的一次運算
3. 計算以 n 作大小的算法之運算量 $f(n)$

+ 算法的複雜度 Complexity of Algorithm

- 假設每次運算需時10^{-11}秒, 運行一個算法需時多久?

TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

* 代表需時多於 10^{100} 年

+ 例2

- 求以下算法的複雜度(Complexity):

procedure *bubblesort*($a_1,\ldots,a_n$: real numbers with $n \geq 2$)

  for i := 1 to $n - 1$

    for j := 1 to $n - i$

      if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1} {$a_1,\ldots, a_n$ is now in increasing order}

教材對應閱讀章節及練習

- 3.1, 3.2(Big-O only), 3.3
- 對應習題: (可視個人情況定量)
  - 3.1: All (試試看能否寫出大致流程, 並根據答案了解步驟)
  - 3.2: 1-8, 18-19, 21-22, 26-27.
  - 3.3: 1-15, 31-32, 42-43