演算法 效能分析

- 時間複雜度
- 空間複雜度
- 漸進表示法 O Ω Θ

效能分析

- 效能量測

  直接利用實驗的方式，計算所花費的資源。

- 效能分析

  找出該演算法所進行的各種運算，分析所花費的時間與空間等資源。

理想狀態計算機

- 單一處理器
- 隨機存取機器(Random Access Machine, RAM)
  1) 指令一個接著一個順序執行，而非同時執行。
  2) 包含真實電腦中常出現的指令：
     - 運算（加減乘除、餘數、最大值、最小值等）
     - 資料移動（載入、儲存）
     - 控制（條件與非條件分支、副程式的呼叫與返回）

最差情況與平均情況

- 演算法的最差情況 (worst case) 執行時間是任何輸入的執行時間或空間上限
- 演算法的平均情況 (average case) 執行時間則是一般輸入的平均執行時間或空間
- 最差情況與平均情況複雜度可能相同也可能不同

時間複雜度

- T(n) 爲一個完全理想狀態的計算機中其程式所執行的計算時間 (或實際指令次數)。
- 時間複雜度: 演算法的計算時間 T(n) 與問題規模 n 的關係

時間複雜度

- 選擇排序法爲例，$T(n)=A+Bn+Cn^2$ 爲多項式
  - 當問題規模 n 增加， n^2 項將會遠大於其他項
  - 係數並不重要，給予必要的 n，則 Cn^2 均會大於 Bn。
- 因此可以寫成 $T(n) \in O(n^2)$ ，表示此演算法的計算時間 T(n) 具有 n^2 "order" 的時間複雜度 (級數)

空間複雜度

- S(n) 爲執行完演算法所需要的記憶體空間
  - 指令空間（instruction space）
  - 資料空間（data space）
  - 環境堆疊空間（environment stack space）

- 若需要空間爲 S(n)=Bn
  因此寫成 $S(n) \in O(n)$ ，表示此演算法的
  所需空間 S(n) 具有 n "order" 的空間複雜度

O符號表示

- O (big-oh) 表示函數的漸進上限

  $f(n) \in O(g(n))$ (或是 $f(n)=O(g(n))$ 此寫法易誤解)

  表示存在正值常數 c 與 n_0 ，對所有的 $n \geq n_0$ 的值

  $$f(n) \leq cg(n)$$

  例: $T(n)=A+Bn+Cn^2$ 則 $T(n) \in O(n^2)$ $T(n) \in O(n^3)$

  但是 $T(n) \notin O(n)$

- 通常選擇最小級數的漸進上限

  代表演算法的時間或空間複雜度

Ω符號表示

- Ω (big-omega) 表示函數的漸進下限

  f(n) ∈ Ω(g(n))，(或是 f(n)=Ω (g(n)) 此寫法易誤解)

  表示存在正值常數 c 與 n_0 ，對所有的 $n \geq n_0$ 的值

  $$f(n) \geqq cg(n)$$

例: T(n)=A+Bn+Cn² 則 $T(n) \in \Omega(n^2)$ $T(n) \in \Omega(n)$

但是 $T(n) \notin \Omega(n^3)$

Θ符號表示

- Θ (big-theta) 表示函數的漸進上下限

  $f(n) \in \Theta(g(n))$，(或是 $f(n)=\Theta(g(n))$ 此寫法易誤解)
  表示存在正值常數 c_1、c_2 與 n_0 ，對所有的 $n \geq n_0$

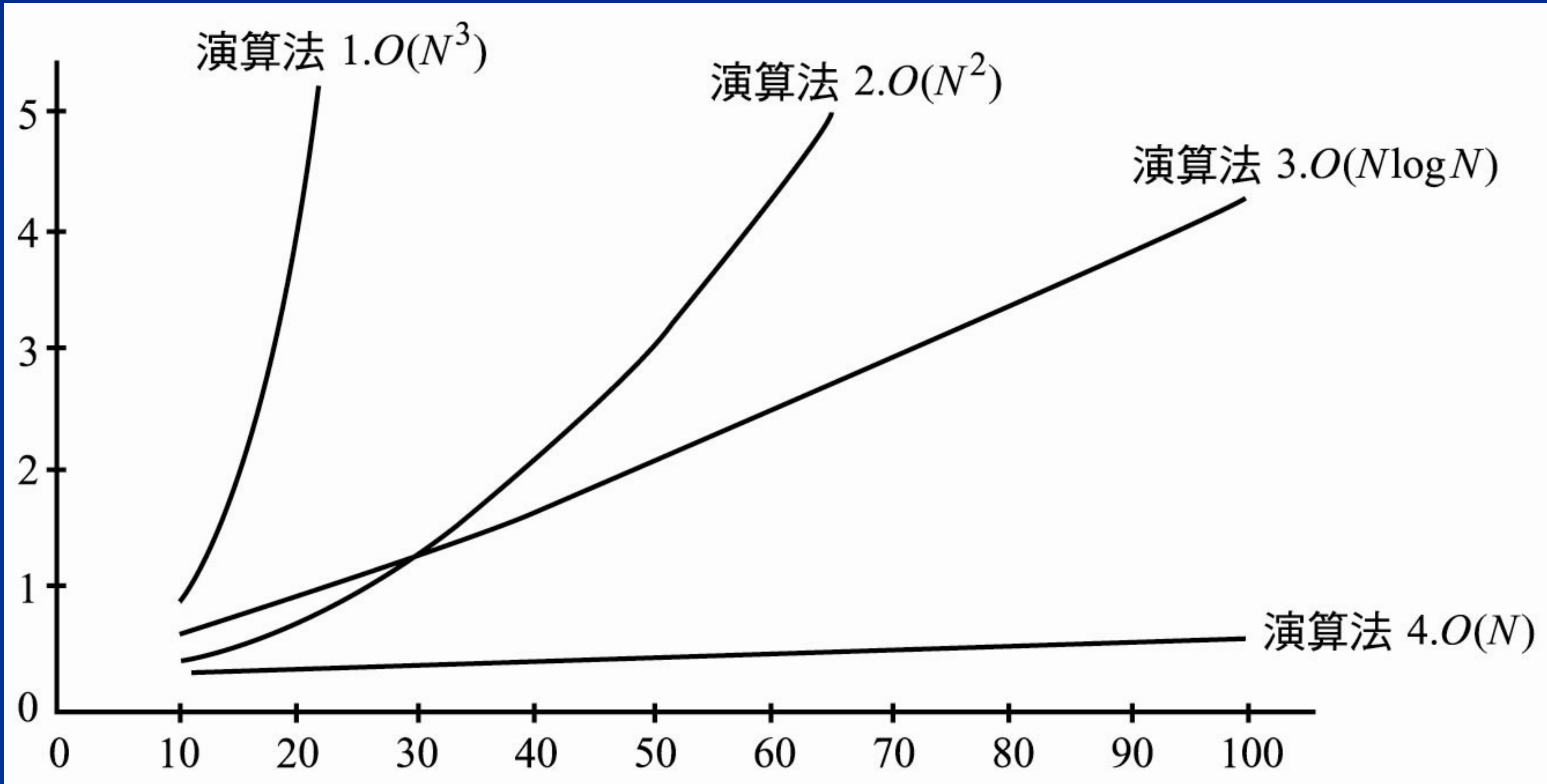
  $$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

  例: $T(n)=A+Bn+Cn^2$ 則 $T(n) \in \Theta(n^2)$

  但是 $T(n) \notin \Theta(n)$ 而且 $T(n) \notin \Theta(n^3)$

  $\Theta(n^2)$ 就是 $T(n)$ 的最小級數漸進上限

常見的O函數成長關係圖



常見的O函數與函數的實際值

函數	名稱	函數的實際值					
O(1)	常數	-	-	-	-	-	-
O(*log* n)	對數	0	1	2	3	4	5
O(n)	線性	1	2	4	8	16	32
O(n *log* n)	n *log* n	0	2	8	24	64	160
O(n^2)	平方	1	4	16	64	256	1,024
O(n^3)	3次方	1	8	64	512	4,096	32,768
O(2^n)	指數	2	4	16	256	65,536	4,294,967,296

定理 2.1

- 如果 $f(n)=a_d n^d + a_{d-1} n^{d-1} + \ldots + a_1 n + a_0,\ a^d > 0$ 是一個 d 次多項式，則 $f(n) \in O(n^d)$

$$T(n) = A + Bn + Cn^2 \quad C > 0 \quad n \geq 0$$

$$T(n) \leq \left|A + Bn + Cn^2\right|$$
$$\leq \left|Cn^2\right| + \left|Bn\right| + \left|A\right| \leq Cn^2 + \left|B\right|n^2 + \left|A\right|n^2 \leq Dn^2$$
$$D = C + \left|B\right|n_0 + \left|A\right|n_0{}^2$$

$$\Rightarrow T(n) \in O(n^2)$$

定理 2.1 的延伸法則

$$f(n)+g(n)\in O(\max\{f(n),g(n)\})$$

$$f(n)\times g(n)\in O(f(n)\times g(n))$$

假設 $f(n)=n^3+n$，$g(n)=n^2+3n+1$

$f(n)\in O(n^3)$ $g(n)\in O(n^2)$

則：

$f(n)+g(n) = n^3+n^2+4n+1\in O(n^3)$

$f(n)g(n) = n^5+3n^4+2n^3+3n^2+n\in O(n^5)$

- 如果 $f(n)=a_d n^d + a_{d-1} n^{d-1} + \ldots + a_1 n + a_0,\ a^d > 0$ 是一個 d 次多項式，則 $f(n) \in \Omega(n^d)$

$$T(n) = A + Bn + Cn^2 \quad C > 0 \;\; n \geq 0$$

$$T(n) \geq Cn^2 - \left|A + Bn\right|$$

$$\geq Cn^2 - \left|\frac{A}{n^2} + \frac{B}{n}\right| n^2 \geq Dn^2$$

$$D = C - \left|\frac{A}{n_0{}^2} + \frac{B}{n_0}\right| > 0$$

$$\Rightarrow T(n) \in \Omega(n^2)$$

$$\Rightarrow T(n) \in \Theta(n^2)$$

效率的差異

- 解決相同問題的不同演算法，通常在效率上會有差異，而且比硬體與軟體的差異還來得大。

- 「插入排序」需時 c_1n^2 ，用快的A電腦執行
  「合併排序」需時 $c_2n\,log_2n$，用慢的B電腦執行
  - A電腦每秒執行10億個指令，B電腦每秒執行1000萬個指令，則可知A電腦比B電腦快100倍
  - 假設要排序一個100萬個數字的陣列，c_1=2，c_2=50
  - 在A電腦花費 $2 \times (10^6)^2 / 10^9 = 2000$ 秒
  - 在B電腦花費 $50 \times (10^6) log_2(10^6) / 10^7 \approx 100$ 秒

演算法分析

- 分析內部迴圈的重複次數, 與問題規模 n 有關的迴圈越複雜, 時間複雜度越高
- 空間複雜度與問題規模 n 通常呈線性關係, 但在有遞迴的情況則不一定

選擇排序法

- 最差情況與平均情況相同, $T(n)=A+Bn+Cn^2$, 時間複雜度爲 $\Theta(n^2)$

- 最差情況與平均情況相同,
  除輸入外, 演算法需要空間 $S(n)=A$
  空間複雜度爲 $\Theta(1)$

範例 2-1 計算 1+2+…+n=?

```
ALGORITHM CountUp(n)
    k ← 1，y ← 0
    while k<=100 do
        y ← y+k
        k ← k+1
    return y
```

範例 2-1 計算 1+2+...+n=?

```
#include <stdio.h>
int count_up(int n){
   int k,y; for(k=1, y=0;k<=n; k++) y=y+k;
   return y;
}
main(){
   int n=100;
   printf("%d\n", count_up(n) );
}
```

複雜度

- 時間複雜度
  最差與平均相同, 只有一個迴圈,
  計算時間 $T(n)=Bn$
  因此為 $\Theta(n)$

- 空間複雜度
  最差與平均相同, 使用記憶體為常數, $S(n)=A$
  因此為 $\Theta(1)$

範例 2-3 氣泡排序法

- 從全部未排序好的元素中，相鄰元素兩兩相比，較小的往前移, 最小的就會移到最邊邊

0	1	2	3	
3	2	1	4	i=0, j=3
3	2	1	4	i=0, j=2
3	1	2	4	i=0, j=1
1	3	2	4	i=1, j=3

0	1	2	3	
1	3	2	4	i=1, j=2
1	2	3	4	i=2, j=3
1	2	3	4	

由演算法設計程式

i=0, j=3

i=0, j=2

i=0, j=1

i=1, j=3

i=1, j=2

i=2, j=3

```
n=4;

for(i=0;i<=n-2;i++)   // i 由 0 到 n-2
     for(j=n-1;j>=i+1;j--)  // j 由 n-1 到 i+1
          if (A[j]<A[j-1])
               swap(A, j, j-1);
               //交換 j 與 j-1 位置
```

氣泡排序法

```
void BubbleSort(int A[ ],int n){
   int i, j;
   for(i=0;i<=n-2;i++)
        for(j=n-1;j>=i+1;j--)
                if (A[j]<A[j-1]) swap(A, j, j-1);
}
void swap(int A[ ], int k, int m){
   int tmp;  tmp=A[k]; A[k]=A[m]; A[m]=tmp;
}
```

演算法的表示

ALGORITHM BubbleSort(A[0..n-1])

for i ← 0 to n-2 do

for j ← n-1 downto i+1 do

if A[j]<A[j-1]

swap A[j] and A[j-1]

完整的程式

問題規模　資料結構: 陣列

輸入:隨機產生數值

```
main(){
  int n=10;  int A[n];   int k;
  srand(time(0));
  for(k=0; k<n; k++) { A[k]=rand()%n;
      printf("%d ", A[k]);
  } printf("\n");
  BubbleSort(A, n);
  for(k=0; k<n; k++) printf("%d ", A[k]);
}
```

演算法

輸出:排序之結果

測試結果

參考範例 2-3

- n=5 0 4 2 3 2
  0 2 2 3 4
- n=10 9 2 1 1 6 1 7 2 8 8
  1 1 1 2 2 6 7 8 8 9
- n=15 11 12 9 8 2 7 13 8 9 0 6 1 4 7 10
  0 1 2 4 6 7 7 8 8 9 9 10 11 12 13

效能分析

for(i=0;i<=n-2;i++)   i 由 0 到 n-2

    for(j=n-1;j>=i+1;j--)   j 由 n-1 到 i+1

迴圈完成約需要 C(n) 個比較

$C(n)= (n-1) + (n-2) + \ldots + 1 = \frac{1}{2}(n-1)n$

$= \frac{1}{2}(-n+n^2)$

運算時間

$T(n) = T_0C(n) = Bn+Cn^2$ 時間複雜度 $\Theta(n^2)$

效能分析

- 演算法所需要的記憶體空間
- 指令空間 ~ 常數
- 資料空間 ~ 常數, 空間複雜度爲 $\Theta(1)$
- 堆疊空間 ~ 常數