CHAPTER 01

資料結構概念

對於喜歡撰寫程式設計的你來說，資料結構是一門讓你進入專業程式設計師的重要學科，舉凡你想撰寫資料庫程式設計，網頁程式設計，手機程式設計，電玩程式設計或是機器人程式設計，甚至你想要撰寫系統程式，如編譯器、作業系統或韌體等，資料結構是你必須了解的科目。你是否準備好了呢？請站穩腳步，放鬆心情，我們要出發囉！

一般對於剛學會程式設計的讀者來說，對於資料結構一定相當陌生，甚至對於資料結構一知半解，因此本章所要學習的重點，就是要讓讀者對於資料結構有一個基本認識，然後再對資料結構的議題做全盤了解，最後再稍微對與資料結構有關聯的演算法做一簡略之說明，以讓讀者在學習資料結構時能清楚了解自己的學習目標。讓我們趕快來了解這些基本議題。

1-1 資料結構的意義

理論上學習資料結構前應該先有程式設計的基礎，因為資料結構學科是程式設計的進階課程，因此建議讀者，若對於程式設計完全沒有基礎，應該先行了解一下程式設計的基本概念。

1-1-1 何謂資料結構

在說明資料結構的定義之前，我們先來看一個範例，假設請你寫一支程式，讓使用者輸入國文、英文與數學分數，然後將三個分數加總後平均，你會怎麼撰寫程式呢？以下為兩種程式設計的寫法：（假設使用C語言）

第一種寫法：

```
int main(void){
      int x,y,z;
      int sum=0;
      double avg=0.0;
      scanf("%d",&x) ;
      scanf("%d",&y) ;
      scanf("%d",&z) ;
      sum=x+y+z ;
      avg=sum/3.0;
      printf("%f\n",avg);
      system("pause");
```

```
    return 0;
}
```

第二種寫法：

```
int main(void){
    int student[3];
    int sum=0, n=0, i;
    double avg=0.0;
    do{
        scanf("%d",&student[n]) ;
        n++ ;
    }while(n<3);

    for(i=0 ;i<3 ;i++)
        sum=sum+ student[i];
    avg=sum/3.0;
    printf("%f\n",avg);
    system("pause");
    return 0 ;
}
```

從上面的程式來看，他們有什麼不同？第一個程式運用三個變數來儲存國文、英文和數學資料，然後運用3個scanf來讀取資料後再做加總與平均。第二個程式則使用陣列來儲存所需資料，再運用while迴圈與for迴圈來讀取資料與加總和平均。從不同程式的寫法可了解，不同資料儲存方式將會有不同之程式設計方式。對於這些資料擺放在變數或陣列內的方式（或稱結構），我們就稱為資料結構，而且不同的資料結構設計，將會影響程式設計（演算法）的方法與效率。我們可以這麼說，資料結構就是在學習對於各種問題中，如何將資料擺放在變數或陣列中，好讓我們撰寫程式時能夠更有效率。因此，我們可以針對資料結構作以下的定義：

所謂資料結構(Data Structure)，就是對於要解決的問題，為了配合演算法的運算，考慮如何運用變數，好讓所要解決問題的資料，在程式內有結構化的存放，以方便演算法的計算，並提升演算法的效率。

因為宣告變數就是向記憶體索取空間，因此，資料結構的定義也可用另一種方式說明：

所謂資料結構，就是在探討如何將資料有組織的存放在記憶體中，以提升程式的執行效率。

日常生活中有很多類似的範例，例如你要查英文字典，查詢字典會有一定的步驟，可稱為演算法，而英文字典內的英文單字排列方式，如以ABC字母順序排列，就是所謂資料結構。因此，我們要查字典時，一定要配合英文單字的排列方式來查詢才會提升我們的查詢效率。在圖書館借書，書籍的擺放方式也是一種資料結構；家長到學校找學生，學校的班級規劃也是一種資料結構。

1-1-2 靜態與動態資料結構

了解資料結構的定義之後，我們再提出靜態與動態資料結構的不同，提供讀者參考。

1. 靜態資料結構(static data structure)：所謂靜態資料結構，是指資料所占用的空間大小及資料數目都必須事先宣告，因此在程式編譯的同時，空間的大小及資料數目就無法改變了。
2. 動態資料結構(dynamic data structure)：所謂動態資料結構，資料所使用之空間大小及資料數目都不必事先宣告，在程式執行時才做記憶體的空間配置。

在後面的資料結構議題中，陣列的使用就是靜態資料結構的代表，而動態資料結構則以鏈結串列結構為代表，靜態與動態資料結構各有各的特性，不同之處整理如下：

靜態資料結構	動態資料結構
使用陣列結構。	使用鏈結串列結構。
需事先宣告空間大小與資料數量。	不必事先宣告。
程式編譯後，空間大小即已固定。	程式執行時才做記憶體空間配置。
不需額外連結欄位，比較節省記憶體空間。	每一個資料必須多一個指標欄位作連接，因此較浪費記憶體空間。
加入或刪除資料，必須做大量搬動。	加入或刪除資料，只須改變指標的指向。
運用索引，可直接做存取。	無法對資料直接做存取。
搜尋資料時，可使用二分搜尋法。	無法使用二分法搜尋。

1-2 資料結構的議題

一般在學習資料結構時，常見的議題包括陣列、鏈結串列、堆疊、佇列、樹狀結構、圖形結構、排序與搜尋等8種，以下我們就針對這8種資料結構簡單的作說明。

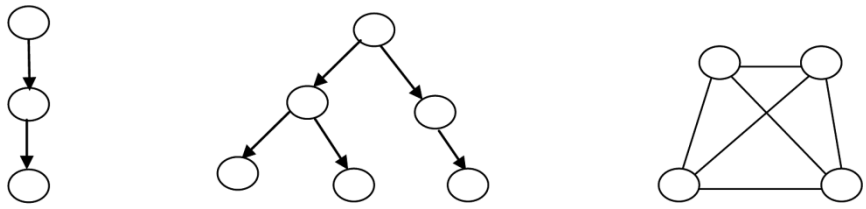
1-2-1 常用的資料結構

資料結構包括陣列、鏈結串列、堆疊、佇列、樹狀結構、圖形結構、排序與搜尋等，其說明分述如下：

資料結構	說明
陣列	是一種只需使用一個名稱卻可以存放大量資料的變數。
鏈結串列	是一種比陣列更有彈性，且不必事先宣告大小的變數。
堆疊	是一種類似碟盤子的資料型態，取出與放入資料都在同一邊執行。
佇列	是一種類似排隊購物的資料型態，出入口在不同邊執行。
樹狀結構	是一種類似族譜的資料型態，非線性集合，有階層關係。
圖形結構	是一種類似地圖的資料型態，有目標地與路徑，為非線性組合。
排序	是一種對資料排列的技術，有遞增或遞減兩種方式。
搜尋	是一種尋找資料的技術。

1-2-2 資料結構的分類

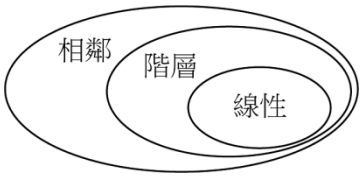
一般資料存在關係，可分為三種，包括一個在前及一個在後的線性關係；一個在上，而可能多個在下的階層關係；以及多個資料相互連結的相鄰關係，如下圖所示：



線性關係　　階層關係　　相鄰關係

線性關係在資料結構裡是最單純、最簡單的結構，陣列、鏈結串列、堆疊與佇列等資料結構就是這種線性關係，而階層關係可以說是相鄰關係的一個特例，其中

樹狀結構就是一種階層關係，圖形結構就是一種相鄰關係。因此，這三者的關係我們可以用下圖來表示之。



1-2-3 資料結構的議題

讀者學習資料結構的目的，除了要提升程式設計能力之外，如果還想要參加國家考試，如高普考、地方特考、關務特考等，以下各資料結構的議題建議你要熟讀，內容包括資料結構的定義（配合圖形）、資料結構表示方式、各種操作（或稱實作）之演算法（包括遞迴及時間複雜度），以及資料結構的應用等等。

資料結構的議題

	定義、表示	種類	如何操作	應用	其他
陣列	定義、特性 表示法	1. 一維陣列。 2. 二維陣列。 3.多維陣列。	讀取、寫入、插入、刪除、搜尋、複製、走訪。	1. 多項式。 2. 矩陣：(a)矩陣運算、(b)稀疏矩陣、(c)上三角矩陣、(d)下三角矩陣。	1. 陣列的定址。 2. 陣列的優缺點。
堆疊	定義 表示法		1.建立、2.新增、3.刪除、4.已滿、5已空。	(1)程式呼叫、(2)資料反轉、(3)運算式的轉換與求值、(4)遞迴應用－河內塔。	
佇列	定義 表示法	環狀佇列、雙向佇列、優先佇列。	加入、刪除、走訪。		
鏈結串列	定義、 如何表示	單向鏈結串列、雙向鏈結串列、環形串列。	讀取、寫入、取代、插入、刪除、搜尋、走訪、計算長度、反轉操作。	(1)多項式、稀疏矩陣。 (2)堆疊、佇列。 (3)階層關係與圖形結構。	
樹狀結構	定義、名詞、表示法	1. 一般樹。 2. 二元樹：(1)完滿、(2)完整、(3)歪斜、(4)嚴格。 3. 樹林。	1.配置新節點、插入節點、刪除節點、走訪。	(1)引線二元樹、(2)二元搜尋樹、(3)運算式樹、(4)二元排序樹、(5)決策樹（遊戲樹）、(6)霍夫曼樹、(7)B tree、(8)2-3樹、(9)2-3-4樹、(10)平衡樹、(11)M樹、(12)堆積樹、(13)B+樹。	決定唯一二元樹、一般樹轉二元樹、化樹林為二元樹。

資料結構的議題（續）

	定義、表示	種類	如何操作	應用	其他
圖形結構	定義、名詞、表示法	無向圖、有向圖、加權圖。	1.新增／移除頂點與邊、2.走訪(DFS、BFS)。	最小花費擴張樹、最短路徑、拓樸排序、臨界路徑。	尤拉圖、擴張樹、連通單元。
排序	定義	1. 依存放位置。 2. 排序後鑑值位置是否改變。	1.選擇排序法、2.插入排序法、3.氣泡排序法、4.快速排序法、5.合併排序法、6.基數排序法、7.謝爾排序法。		堆積排序法、二元樹排序法、各種排序比較。
搜尋	定義	1. 內部／外部。 2. 循序／非循序。	1.循序法、2.二元搜尋法、3.插補插入法、4.費勃納法、5.雜湊。		

1-3 演算法概念

本書雖然是談論資料結構，但資料結構對於演算法來說是密不可分，因此，本單元對於需考慮的演算法基本概念，還是簡略作一些說明，以提供讀者對演算法有基本概念，若讀者對於演算法想深入了解，建議讀者可以參考演算法相關書籍。

1-3-1 演算法與資料結構

演算法與資料結構是密不可分的兩個學科，若以程式概念來看，演算法可以說就是程式碼部分，而資料結構就是變數宣告的部分。因此，我們常說，程式=資料結構+演算法。

所謂演算法，簡單的說，就是探討問題的解決方法，這些解決方式，我們會希望最後的處理由計算機來執行，因此，問題的解決方法希望能訴諸於程式，交由計算機來執行。程式寫得好壞，與資料結構設計的方式有關，就好像要讓烹飪效率提升，準備工作絕對相當重要，不同的問題，就要考慮不同的資料結構設計方式，才能有效提升程式效率。

1-3-2 演算法表示方式

一般來說，演算法的表示方式有四種，包括自然語言、流程圖、虛擬碼及程式碼四種，其中以虛擬碼最為標準，因為他適用於各種程式語言，所以是最符合演算法的表示方式。虛擬碼表示方式請參考下面單元之虛擬碼表示法。

1-3-3 如何判定演算法的好壞

經過資料結構與演算法的配合後，演算法的撰寫是否最有效率，他的判定方式基本上有兩種，一種為時間複雜度，一種為空間複雜度。

時間複雜度是指演算法執行完畢後所需花費的執行時間；空間複雜度是指演算法執行完畢後所需的記憶體空間。由於記憶體空間可用虛擬記憶體(Virtual Memory)技術，或儲存於輔助記憶體(Secondary Memory)，況且目前計算機的記憶體不但便宜，且空間都相當大，其容量可隨之擴充，因此，演算法的好壞，一般都使用時間複雜度來進行程式效能分析（演算法好壞的判定）。

1-4 程式效能分析

怎樣的資料結構與演算法作配合，才能使整個程式在執行時最有效率呢？或許你會考慮有好的設備自然就能提升執行效率，聽起來似乎是沒錯，程式執行效率會受程式語言工具、程式編譯工具或計算機硬體的影響，但是如果單純考慮資料結構與演算法的設計，我們似乎還是需要有一套評定的標準，來說明演算法好壞。程式效能分析就是演算法好壞的判定方式，以下就來說明程式效能的分析方式。

1-4-1 運算執行次數

對於演算法的好壞，執行時間的長短是首要考慮因素，演算程式在計算機上的執行時間，會牽涉到硬體好壞的問題，一般要評估演算法執行時間的長短，我們必須排除硬體設備問題，因此，我們一般都是依據演算法中的運算執行次數多寡來當作演算法的執行時間。

1. 運算執行次數：

如何計算一支程式的運算執行次數呢？通常每執行一行程式，我們就計算其執行程式一次，若迴圈從1–n每次累進1，則我們就說程式執行了n次，請看下面計算範例。

演算法	最佳情況	最差情況	平均情況
int sum(int a[], int n)			
{			
int i;	1	1	1

演算法	最佳情況	最差情況	平均情況
for(i=0; i<n; i++)	1	n+1	n/2+1
{			
if(a[i]= =9)	1	n	n/2
break;	1	1	1
}			
printf(“%d\n”,i);	1	1	1
}			
合計執行次數	5	2n+4	n+4

2. 種類：

程式執行時，可能因資料的排序或處理方式的不同造成處理的時間會有不同，例如要搜尋一筆資料，結果第一筆就是我們要的資料，我們稱為最佳情況，也有可能所有資料都搜尋完後才搜尋到資料，我們把他稱為最差情況，另一種就是平均情況，因此，在分析程式執行的次數，一般都要考慮這三種情況，即所謂的最差情況(Worse Case)、最佳情況(Best Case)與平均情況(Average Case)。

範例練習

1. 有一段程式如下，請計算其執行次數。

```
int sum(int n)
{
    int i=0, ttl=0;
    while(i<=n)
    {
        ttl+=i;
        i++;
    }
    return ttl;
}
```

2. 請設計一支程式，可以計算全班n位同學多益(TOEIC)成績的總分，並計算其執行次數。

答

1.

演算法	執行次數
int sum(int n)	
{	
int i=1, ttl=0;	1
while(i<=n)	n+1
{	
ttl+=i;	n
i++;	n
}	
return ttl;	1
}	
合計執行次數	3n+3

共執行3n+3次。

2.

演算法	執行次數
int ttlscore(int score[], int n)	
{	
int i, ttl=0;	1
for(i=0;i<n,i++)	n+1
{	
ttl+= score[i];	n
}	
return ttl;	1
}	
合計執行次數	3n+3

共執行3n+3次。

隨堂練習

請設計一支程式，可以計算奇數相加程式(1~n)，並計算其執行次數。

1-4-2 時間成長幅度函數

從上一小節我們已經了解如何計算演算法的執行次數了，現在我們來試著比較一下不同演算法的優劣。假設有一個問題用兩種不同之演算法來處理，經過執行次數的計算後，第一個演算法的執行次數是1000n+3，第二個演算法的執行次數是n^2+4，請問哪一個演算法比較好呢？從這兩個方程式來看，真的不容易判斷哪一個演算法較好，我們試著用下表來分析：

n	(1)：1000n+3	(2)：n^2+4	(1)減(2)
1	1003	5	998
10	10003	104	9899
100	100003	10004	89999
1000	1000003	1000004	-1
10000	10000003	100000004	-90000001
100000	100000003	10000000004	-9900000001

從上表可知，當n在1~100時，似乎第一個演算法效果較差，因為他的執行次數較多，但當n超過1000時，第二個演算法的執行次數開始快速增加，此種增加方式，我們將他稱為執行時間的成長幅度，也就是第二種演算法的執行時間較長，並不是我們所能接受的。可見得，時間複雜度不應只關心他的執行次數，更應考慮執行時間的成長幅度。那我們是否可以將每一個演算法的執行次數轉換成執行時間的成長幅度，請看以下的執行時間的成長幅度表示法。

若演算法執行次數為f(n)，如果我們可以找到兩數c與n_0，且對於所有的n當中，我們可以找到一個n_0，當$n \geq n_0$時，使得$f(n) \leq cg(n)$均成立。此時我們可以說，f(n)的成長是漸趨近於g(n)，但永遠不會超過g(n)，也就是說g(n)是f(n)的上限，記作f(n)=O(g(n))，g(n)稱為成長幅度函數。理論上，我們習慣用成長幅度函數來表示時間複雜度。

範例練習

1. 某演算法執行次數為T(n)=1001n+5，其成長幅度函數為多少。
2. 請問下列程式中，若只考慮x++運算式，請問其成長幅度函數為多少 ？

```
i=1
while(i<=n){
    for(j=i ;j<=n ;j++)
        x++ ;
    i++
}
```

3. 某演算法執行次數為 $T(n)=\sum_{i=1}^{n} i$ ，其成長幅度函數為多少。
4. 請問下列程式中，若只考慮y--運算式，請問其成長幅度函數為多少？

```
y-- ;
```

答

1. O(n)

   T(n)=1001n+5≤1001n+n
   也就是T(n) ≤1002n，我們可以找到c=1002
   並且，我們可以找到一個n_0的值，使得T(n) ≤1002n成立
   依定理f(n)≤cg(n)知，f(n)=O(g(n))，也就是T(n)=O(n)
   即T(n)的成長幅度函數為O(n)

2. $O(n^2)$

i值	j=1~n	x++ 執行次數
1	1~n	n
2	2~n	n−1
…		n−2
n	n~n	1
	合計次數	(1+n)*n/2

   也就是演算法共執行了(1+n)*n/2= $n/2+n^2/2$
   我們可以寫成$T(n)=n^2/2+ n/2\leq n^2$
   也就是$T(n)\leq n^2$，我們可以找到c=1
   並且，我們可以找到一個n_0的值=0，當$n\geq n_0$時，使得$T(n)\leq n^2$成立

依定理f(n)≤cg(n)知，f(n)=O(g(n))，也就是T(n)= $O(n^2)$
即T(n)的成長幅度函數為$O(n^2)$

3. $O(n^2)$

$$T(n)=\sum_{i=1}^{n} i=1+2+...+n=\frac{(1+n)*n}{2}=\frac{n^2+n}{2}$$

同上題，T(n)的成長幅度函數為$O(n^2)$

4. O(1)
y--表示只執行y=y+1，執行1次，T(n)=1
T(n)的成長幅度函數為O(1)

1-4-3 時間複雜度

1. 定義：時間複雜度(Time Complexity)就是指程式執行完畢時所需的時間，該時間可包括編譯時間(Compile Time)與執行時間(Execution Time)。一般來說，我們只考慮的是執行時間。

2. 表示方式：上節曾提到，時間複雜度的表示方式是使用成長幅度函數，而此函數可劃分成三種，包括理論上限O(n)、理論下限Ω(n)與理論上下限Θ(n)。一般常用的表示方式都是使用理論上限(O(n))來表示。

3. 理論上限O(n)：讀成Big-Oh of n，即f(n)=O(g(n))，若且為若存在著兩數c與n_0，且對於所有的n，當$n\geq n_0$時，使得f(n)≤cg(n)均成立。

f(n)指的是某演算法的運算執行次數，n_0是不等式f(n)≤cg(n)上的某一個n值，c只是f(n)與g(n)的一個比例值。此理論的意思是指若你能找到2個正的常數c和n_0，使得f(n)≤cg(n)均成立，則cg(n)的運算次數會永遠大於等於f(n)，我們可視為cg(n)是f(n)的上限，則f(n)的運算執行次數可寫成f(n)=O(g(n))，讀成f(n)的運算執行次數的上限是g(n)。

例如：有一個演算法的運算執行次數f(n)是3n+8，若我們要找上限，就可以將最大的加項再加一值，即我們可以讓3n+n，形成4n，則可得f(n)=3n+8≤4n=cg(n)，此時，c=4，g(n)=n，若我們可以在3n+8≤4n式子中找到一個n_0，且$n\geq n_0$時，使得3n+8≤4n式子是合理的，那我們就可以說f(n)=3n+8之上限是g(n)=n。整理3n+8≤4n式子，得8≤n，也就是當n_0=8時，3n+8≤4n成立，從理論得知，我們更可確定f(n)的理論上限是g(n)=n，此範例可記作f(n)=O(n)。

4. 理論下限Ω(n)：讀成Omega of n，f(n)=Ω(g(n))，若且為若存在著兩數c與n_0，且對於所有的n，當$n \geq n_0$時，使得$f(n) \geq cg(n)$均成立。

相同的，此理論是指若你能找到2個正的常數c和n_0，使得不等式$f(n) \geq cg(n)$均成立，則cg(n)的運算次數會永遠小於等於f(n)，我們可視為cg(n)是f(n)的下限，則f(n)的運算執行次數可寫成$f(n)=\Omega(g(n))$，讀成f(n)的運算執行次數的下限是g(n)。

例如：有一個演算法的運算執行次數f(n)是$3n^2+8n$，我們要找下限，也就是要找比$3n^2+8n$更小的，我們可以保留最大的加項，刪掉最小的加項，變成$3n^2$，就可以讓$3n^2+8n \geq 3n^2$，也就是$cg(n)= 3n^2$，$c=3$，$g(n)= n^2$，同理，我們要再找一個n_0，使得$f(n) \geq cg(n)$成立。整理$3n^2+8n \geq 3n^2$式子，得$8n \geq 0$，也就是當$n_0=1$時（$n_0=0$~n均可），$3n^2+8n \geq 3n^2$成立，從理論得知，我們更可確定f(n)的理論下限是$g(n)=n^2$，此範例可記作$f(n)=\Omega(n^2)$。

5. 理論上下限Θ(n)：讀成Theta of n，f(n)=Θ(g(n))，若且為若存在著c_1、c_2與n_0，且對於所有的n，當$n \geq n_0$時，使得$c_1g(n) \leq f(n) \leq c_2g(n)$均成立。

相同的，此理論是指若你能找到3個正的常數c_1、c_2與n_0，使得不等式$c_1g(n) \leq f(n) \leq c_2g(n)$均成立，則f(n)的運算次數會永遠介於或等於$c_1g(n)$與$c_2g(n)$之間，我們可視為$c_1g(n)$與$c_2g(n)$是f(n)的上下限，則f(n)的運算執行次數可寫成$f(n)=\Theta(g(n))$，讀成f(n)的運算執行次數會介於$c_1g(n)$與$c_2g(n)$之間。

例如：有一個演算法的運算執行次數f(n)是$2n^2+3n+2$，我們要找上限與下限，也就是要找比$2n^2+3n+2$更小與更大的，我們可以利用前面的作法，類似找理論上限與理論下限，上限可找更大的$3n^2$，下限可找較小的$2n^2$，就可以得到$2n^2 \leq 2n^2+3n+2 \leq 3n^2$，也就是$c_1g(n)=2n^2$，$c_2g(n)=3n^2$，可得$c_1=2$，$c_2=3$，$g(n)=n^2$，同理，我們要再找一個$n_0$，使得$c_1g(n) \leq f(n) \leq c_2g(n)$成立。整理$2n^2 \leq 2n^2+3n+2 \leq 3n^2$式子，也就是先分開計算，$2n^2 \leq 2n^2+3n+2$，得$-1/3 \leq n$，另一個不等式$2n^2+3n+2 \leq 3n^2$，可得$2 \leq n^2-3n=n*(n-3)$，也就是n要大於4才符合$2 \leq n*(n-3)$，另一式$-1/3 \leq n$，由二式可得$n_0=4$，$2n^2 \leq 2n^2+3n+2 \leq 3n^2$式子成立，從理論得知，我們更可確定f(n)的理論上下限是$g(n)= n^2$，此範例可記作$f(n)=\Theta(n^2)$。

6. 常見的時間複雜度：

常數	對數	線性	對數線性	平方	立方	指數	階層
O(1)	$O(\log_2 n)$	O(n)	$O(n\log_2 n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$	O(n!)
無迴圈	單一迴圈		二階與三階槽狀迴圈			特殊	

(1) O(1)：O(1)表示複雜度為常數(Constant)，若演算法裡只有運算式而不包括迴圈或是只有選擇結構，這樣的演算法其時間複雜度多為常數。

例如： A. 告知演算法只有資料量n。

B. 只有運算式，演算法內沒有for或while迴圈，如x++。

C. 演算法只有使用選擇結構，如if(x>5) y-- 等。

(2) $O(\log_2 n)$：$O(\log_2 n)$表示複雜度為對數(Logarithmic)，若演算法裡只有一個迴圈如while或for，執行次數到n，且累計方式(Step)呈指數，如x*=2，這樣的演算法其時間複雜度多為對數。

例如：使用for，且程式內之累計方式呈指數，如x*=2

```
for (i=1;i<=n; i*=2){
    x++}
```

(3) O(n)：O(n)表示複雜度為線性(Linear)，若演算法裡只有一個迴圈如while或for，執行次數到n，且累計方式為線性，如x++，這樣的演算法其時間複雜度多為線性。

例如： A. while，且程式內之累計方式為線性

```
while(x<=n){
    x++}
```

B. for，且程式內之累計方式為線性

```
for (i=1;i<=n;i++){
    x++}
```

(4) $O(n\log_2 n)$：$O(n\log_2 n)$表示複雜度為對數－線性(Log-linear)，若演算法裡只有二階槽狀迴圈如while或for，執行次數都到n，但累計方式一個呈指數，如x*=2，一個呈線性，如x++，這樣的演算法其時間複雜度多為對數線性。

例如： 使用for，且程式內之累計方式一個呈指數，如x*=2，一個呈線性，如x++

```
for (i=1;i<=n; i*=2)
    for (j=1;j<=n; j++){
        x++}
```

(5) $O(n^2)$：$O(n^2)$表示複雜度為平方(Quadratic)，若演算法裡只有二階槽狀迴圈如while或for，執行次數都到n，但累計方式均呈線性，如x++，這樣的演算法其時間複雜度多為平方。

例如： 使用for，且程式內之累計方式均呈線性，如x++。

```
for (i=1;i<=n; i++)
    for (j=1;j<=n; j++){
        x++}
```

(6) $O(n^3)$：$O(n^3)$表示複雜度為立方(Cubic)，若演算法裡只有三階槽狀迴圈如while或for，執行次數都到n，但累計方式均呈線性，如x++，這樣的演算法其

時間複雜度多為立方。

例如：使用for，且程式內之累計方式(step)均呈線性，如x++。

```
for (i=1;i<=n; i++)
    for (j=1;j<=n; j++)
        for (k=1;k<=n; k++){
            x++}
```

(7) $O(2^n)$：$O(2^n)$表示複雜度為指數(Exponential)，若演算法裡只有一個迴圈如while或for，但他的執行次數需要到指數型，如I=1 TO 2^n，這樣的演算法其時間複雜度多為指數。

例如：使用for，執行次數需要到指數型

```
for (i=1;i<=2^n; i++){
    x++}
```

(8) O(n!)：O(n!)表示複雜度為階層(Factorial)，若演算法裡只有一個迴圈如while或for，但他的執行次數需要到階層，如I=1 TO n!，這樣的演算法其時間複雜度多為階層。

例如：使用for，執行次數需要到階層

```
for (i=1;i<=n!; i++){
    x++}
```

7. 大小關係：

$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$。

比較分析如下：加大n值後，可看出各種形式的時間複雜度很明顯的差異。

常數	對數	線性	對數線性	平方	立方	指數	階層
a	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	n!
1	0	1	0	1	1	2	1
1	1	2	2	4	8	4	2
1	2	4	8	16	64	16	24
1	3	8	24	64	512	256	40320
1	3.3	10	33	100	1000	1024	3628800
1	4	16	64	256	4096	65536	20922789888000

範例練習

1. 請計算下列演算法之時間複雜度。

(1) x^3+x^2+1;

(2)
```
if(x<100)
    x+=2;
else
    x--;
```

(3)
```
long recursion_fact(int n){
    if(n==1)
        return 1;
    else
        return n* recursion_fact(n-1);
}
```

2. 請計算下列演算法之時間複雜度。

(1)
```
for(i=0; i<=n; i++)
    a++;
```

(2)
```
for(j=0; j<=n; j+=3)
    b++;
```

(3)
```
for(k=0; k<=n; k*=5)
    c++;
```

3. 請計算下列演算法之時間複雜度。

(1)
```
for(i=0; i<=n; i++)
    for(j=0; j<=n; j+=3)
        a++;
```

(2)
```
for(i=0; i<=n; i++)
    for(j=i;j<=n;j+=3)
        for(k=0;k<=n;k+=5)
            b++;
```

(3)
```
for(i=0; i<=n; i++)
    for(k=0;k<=n;k/=2)
        c++;
```

4. 請計算下列演算法之時間複雜度。

(1)
```
for(i=0 ; i<=100 ; i++)
    for(j=0; j<=100 ; j+=3)
        a++ ;
```

(2)
```
for(i=0 ; i<=n ; i++)
    for(j=0; j<=n ; j+=3)
        a++ ;
for(i=0 ; i<=n ; i++)
    for(j=i ;j<=n ;j+=3)
        for(k=0 ;k<=n ;k+=5)
            b++;
```

5. 若執行次數為以下結果，則成長幅度函數為多少。
   (1) 3n+4
   (2) 9n+2logn
   (3) $3n^2+8n+1$
   (4) $5n^2+3nlogn+7n$
   (5) $5*2^n+n^3+1$
   (6) n^4+2^n

答

1. (1) 正常的運算式且無迴圈，基本上其實間複雜度為O(1)。
   (2) 正常的運算式且無迴圈（只有選擇結構），基本上其時間複雜度為O(1)。
   (3) 這題目雖然無迴圈（只有選擇結構），但演算法內之程式是呼叫本身之函數，且函數之參數是n，故視同迴圈，時間複雜度為O(n)。

2. (1) 只有一個迴圈，執行次數到n，累計為i++呈線性，所以時間複雜度為O(n)。
   (2) 只有一個迴圈，執行次數到n，累計為j+=3呈線性，所以時間複雜度為O(n)。
   (3) 只有一個迴圈，執行次數到n，但累計為k*=5呈指數，所以時間複雜度為O(logn)。

3. (1) 有二階槽狀迴圈，執行次數到n，累計為i++與j+=3均呈線性，所以時間複雜度為$O(n^2)$。
   (2) 有三階槽狀迴圈，執行次數到n，累計為i++與j+=3與 k+=5均呈線性，所以時間複雜度為$O(n^3)$。
   (3) 有二階槽狀迴圈，執行次數到n，累計為i++與k/=2各呈現線性與指數，所以時間複雜度為O(nlogn)。

4. (1) 雖然為二階槽狀迴圈，但執行次數只到100，所以時間複雜度為O(1)。
   (2) 雖然有5個迴圈，但其實是一個二階槽狀迴圈，一個三階槽狀迴圈，執行次數可視為n^2+n^3視為，時間複雜度取大值，所以時間複雜度為$O(n^3)$。

5. 我們可以依據$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$之大小關係來判斷本題目（相加者，直接取較大者）。
   (1) 3n之時間複雜度為O(n)，4之時間複雜度為O(1)，所以結果為O(n)。
   (2) 9n之時間複雜度為O(n)，2logn之時間複雜度為$O(\log_2 n)$，所以結果為O(n)。
   (3) $3n^2$之時間複雜度為$O(n^2)$，8n+1之時間複雜度為O(n)，所以結果為$O(n^2)$。
   (4) $5n^2$之時間複雜度為$O(n^2)$，3nlogn之時間複雜度為$O(n\log_2 n)$，所以結果為$O(n^2)$。
   (5) $5*2^n$之時間複雜度為$O(2^n)$，n^3+1之時間複雜度為$O(n^3)$，所以結果為$O(2^n)$。
   (6) n^4之時間複雜度為$O(n^4)$，2^n之時間複雜度為$O(2^n)$，所以結果為$O(2^n)$。

1-4-4 空間複雜度

空間複雜度(Space Complexity)是指執行程式完畢後所需要的記憶體空間。包括固定記憶體空間與變動記憶體空間。

1. 固定記憶體空間(Fixed Space)：這是指程式本身的指令空間、變數或結構所需的空間。
2. 變動記憶體空間(Variable Space)：這是指程式執行時所需的額外空間，例如執行遞迴程序、動態記憶體空間配置等。

1-5 虛擬碼表示法

虛擬碼(Pseudo Code)是一種用來表示演算法的最好的一種工具之一，他的寫法並未統一，因此沒有所謂的標準寫法，我們運用演算法的三種結構：循序、選擇與重複，來說明虛擬碼的寫法，以提供讀者參考。

結構	關鍵字	虛擬碼寫法	C、JAVA等程式	VB、PB等程式
循序	運算式	i←x1+x2	i=x1+x2；	i=x1+x2
	=	=	= =	=
	not	<>	!	<>
	and	and	&&	and
	or	or	||	or
選擇	if	If條件then end If	if（條件）{ }	If條件Then End If

結構	關鍵字	虛擬碼寫法	C、JAVA等程式	VB、PB等程式

	if, else	If條件then
		else
		end If
			if（條件）{
			} else {
			}
				If條件Then
				Else
				End If

	switch	Case條件
		:1:
		:2:
		:else:
		end Case
			switch（條件）{
			case 1:
			break;
			default:
			break;
			}
				Select Case運算式
				Case 1
				Case 2
				End Select

重複	while	While條件do end While	while（條件）{ }	Do While條件 Loop 或 While條件 End While 另一種 Do Until條件 Loop
	do while	Do end While條件	do{ } while（條件）;	Do Loop While條件 另一種 Do Loop Until條件
	for	For i←x1 to x2 do end For	for(i=a;i<b;i++){ }	For i=a To b Step 1 Next i
	Exit	exit for	break	Exit For Exit Do Exit Sub Exit Function
	Continue	continue	Continue	Continue
	Return	return	return	return
宣告	dim	var x：integer	int x	Dim x As integer
Array	Array	A[]	A[] A[][]	A() A(,)
Pointer	Pointer	Tree->data		
函數	Procedure	Begin end	void 名稱(){ }	Sub名稱() End Sub