

資料結構













CH01 程式語言複習及演算法分析

Java簡介

- Java
 - 1995年5月SUN正式發表
 - Java是一個物件導向的程式語言(object-oriented language)
 - 跨作業系統平台，程式的移植性強
- Java SE 7、8、10
 - Java Standard Edition 8
 - 開發 Java 程式之前，須在電腦裡安裝程式開發環境
 - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- 開發工具
 - 文字編輯工具
 - Eclipse
 - NetBeans、IntelliJ、...
- Java SE API 文件
 - <https://docs.oracle.com/javase/10/docs/api/overview-summary.html>

程式編輯與開發 工具- Eclipse

Eclipse Photon R Packages

	<h3>Eclipse IDE for Eclipse Committers</h3> <p>318 MB 118,625 DOWNLOADS</p> <p>Package suited for development of Eclipse itself at Eclipse.org; based on the Eclipse Platform adding PDE, Git, Marketplace Client, source code and developer documentation.</p> <p>Click here to file a bug against Eclipse Platform. Click here to file a bug against Eclipse Git team provider.</p>	 Windows 32-bit 64-bit Mac Cocoa 64-bit Linux 32-bit 64-bit
	<h3>Eclipse IDE for C/C++ Developers</h3> <p>223 MB 111,945 DOWNLOADS</p> <p>An IDE for C/C++ developers with Mylyn integration.</p>	 Windows 32-bit 64-bit Mac Cocoa 64-bit Linux 32-bit 64-bit
	<h3>Eclipse IDE for Java and DSL Developers</h3> <p>348 MB 52,428 DOWNLOADS</p> <p>The essential tools for Java and DSL developers, including a Java & Xtend IDE, a DSL Framework (Xtext), a Git client, XML Editor, and Maven integration.</p>	 Windows 32-bit 64-bit Mac Cocoa 64-bit Linux 32-bit 64-bit
	<h3>Eclipse IDE for Java Developers</h3> <p>195 MB 327,038 DOWNLOADS</p> <p>The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration</p>	 Windows 32-bit 64-bit Mac Cocoa 64-bit Linux 32-bit 64-bit
	<h3>Eclipse IDE for JavaScript and Web Developers</h3> <p>172 MB 19,072 DOWNLOADS</p> <p>The essential tools for any JavaScript developer, including JavaScript, HTML, CSS, XML languages support, Git client, and Mylyn.</p>	 Windows 32-bit 64-bit Mac Cocoa 64-bit Linux 32-bit 64-bit
	<h3>Eclipse IDE for Java EE Developers</h3> <p>345 MB 531,454 DOWNLOADS</p> <p>Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.</p>	 Windows 32-bit 64-bit Mac Cocoa 64-bit Linux 32-bit 64-bit

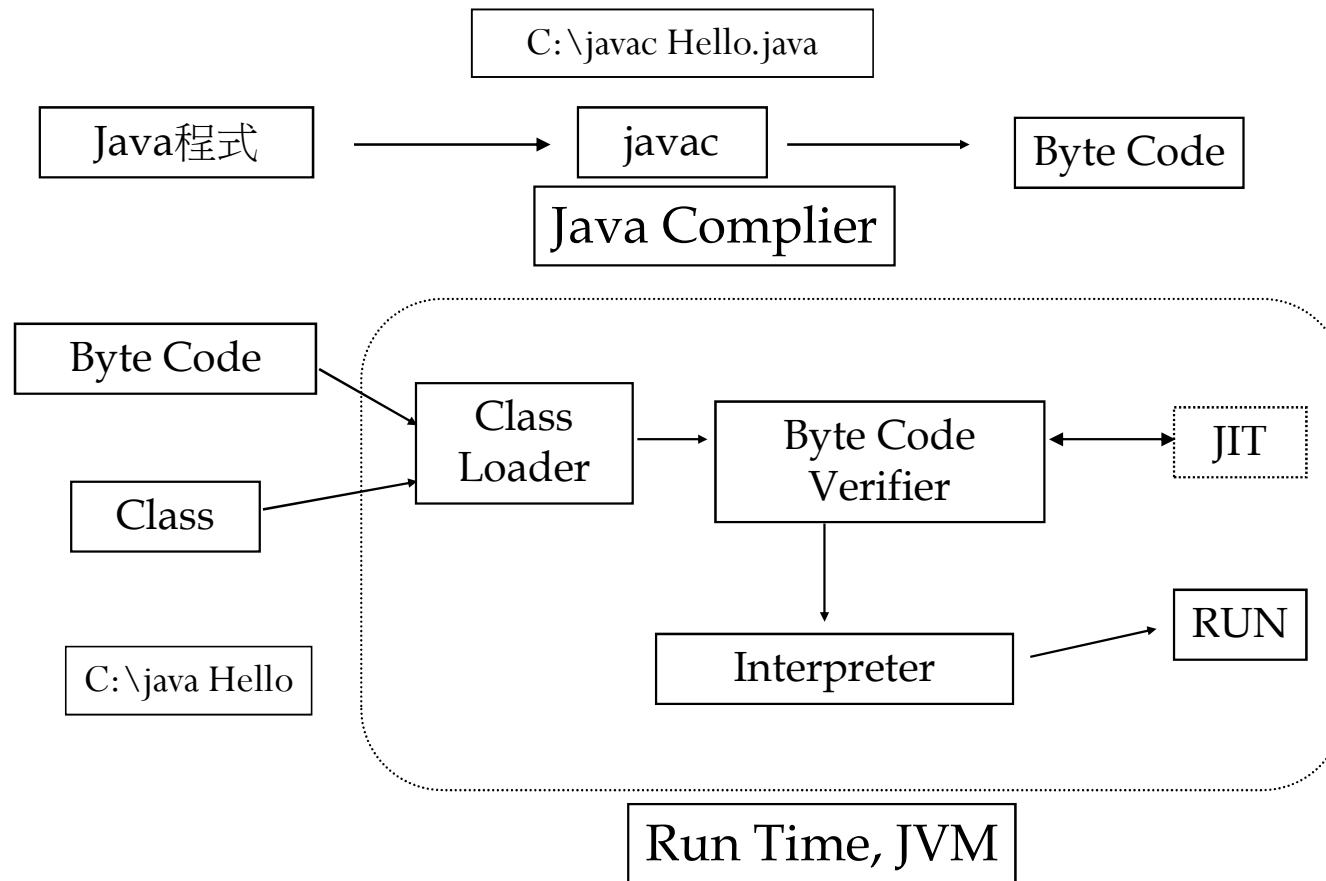
Java程式架構-範例

- Java程式架構是一個「類別」(Class) 宣告，例：

```
01: /* 程式範例: ex1.java */  
02: // 主類別  
03: public class ex1  
04: { // 主程式  
05:     public static void main(String[] args)  
06:     { // 顯示訊息  
07:         System.out.println("第一個Java應用程式");  
08:     }  
09: }
```

分號 ";" 一定要加！

Java程式編譯與執行的過程



Java 的流程控制

1. 條件式流程控制 (decision making)
 - ✓ if-else
 - ✓ switch-case
2. 迴圈式流程控制 (Loop)
 - ✓ for-loop
 - ✓ for-in (for-each)
 - ✓ while-loop
 - ✓ do-while-loop
3. 特定控制單元 (Special Loop)
 - ✓ break
 - ✓ continue

if-else

- if – else

```
1. int money = 100;  
2. if (money == 100) {  
3.     System.out.println("等於 100");  
4. }  
5. else {  
6.     System.out.println("不等於 100");  
7. }
```

執行結果：等於 100

switch-case

- switch-case

```
1. int x = 1;
2. switch(x) {
3.     case 1:
4.         System.out.print("A");
5.         break;
6.     case 2:
7.         System.out.print("B");
8.         break;
9. }
```

執行結果：A

for-loop

- for-loop 程式撰寫格式

```
1. for (int i=1 ; i<=10 ; i++) {  
2.     System.out.print(i + “,”);  
3. }
```

initialization

boolean-expression

stepping

分隔符號（一定要加）

本範例執行結果：1, 2, 3, 4, 5, 6, 7, 8, 9, 10

while-loop

- while-loop 是重複與反覆的意思。當條件為 **true** 時可以不斷反覆執行的迴圈，直到控制項回傳 **false** 為止

- while-loop 程式基本版型

```
while (boolean-expression) {  
    // 程式執行區塊 statement  
}
```

- 例

```
1. int count = 3;  
2. while(count > 0) {  
3.     System.out.print("Java,");  
4.     count=count-1;  
5. }
```

執行結果：Java,Java,Java,

do-while-loop

- do-while-loop 程式基本版型
 - **do {**
 - //程式執行區塊 **statement**
 - **} while(boolean-expression) ;**
- 例
 1. **int count = -1;**
 2. **do {**
 3. **System.out.print("Java,");**
 4. **} while(count > 0);**

執行結果：**Java,**
只執行一次！

一維陣列

```
1.    public void oneDimensionArray(){
2.        int firstArray[]=new int[3];
3.        int secondArray[] =new int[3];
4.        firstArray[0] = 10;
5.        secondArray[1] =2;
6.        for(int i=0;i<firstArray.length;i++){
7.            firstArray[i]=firstArray[i]+secondArray[i];
8.        }
9.        for(int i=0;i<firstArray.length;i++){
10.            System.out.println(firstArray[i]);
11.        }
12.    }
```

二維陣列

```
1.    public void twoDimensionArray(){
2.        int two[][]=new int[3][4];
3.        for(int i=0;i<3;i++)
4.            for(int j=0;j<4;j++)
5.                two[i][j]=(int)(Math.random()*10); // 取亂數
6.        for (int i=0;i<3; i++)
7.        {
8.            for ( int j=0;j<4; j++)
9.                System.out.print(two[i][j]+" ");
10.           System.out.println();
11.        }
12.    }
```


物件導向說明(1)

- Java 是物件導向的語言
 - 我們生活的環境中其實有許多「物件導向」的實例
 - 當我們要描述一個東西時，會敘述它的屬性與行為
 - 例如，要描述一隻拉布拉多犬，會說「有一隻小狗名叫小黃，牠身上的毛是米黃色的 ...牠會幫主人拿報紙、拎拖鞋...」

物件導向說明(2)

- 將小狗設定成一個名叫 **Puppy** 的類別
- 牠的屬性便是狗的品種、名字與毛色等
- 牠具有幫主人拿報紙、拎拖鞋的行為

```
類別 Puppy {  
    品種= 拉布拉多  
    名字 = 小黃  
    毛色 = 米黃  
    特殊行為與技巧 {  
        會幫主人拿報紙與拎拖鞋...  
    }  
}
```



屬性

方法

物件導向說明(3)

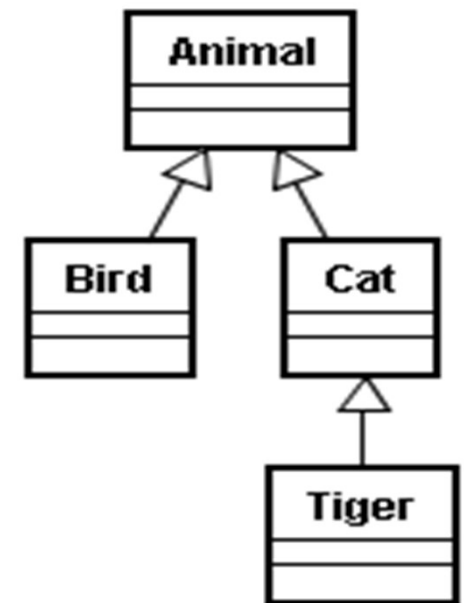
```
1.  class Puppy {
2.      static String dogType = "拉布拉多"; //類別變數
3.      String dogName = "小黃";           //物件變數
4.      String dogColor = "米黃";
5.      Puppy() {}                          //建構子
6.      void skill() {                      // 物件方法
7.          String skill_1 = "拿報紙";    //區域變數
8.          String skill_2 = "拎拖鞋";
9.          System.out.println("幫主人"+ skill_1);
10.         System.out.println("幫主人"+ skill_2);
11.     }
12.     static void move() { // 類別方法
13.         //code for this method
14.     }
15. }
```


物件導向語言的特徵

- 繼承(Inheritance)
- 多型(Polymorphism)
- 封裝(Encapsulation)
- 抽象性(Abstraction)

繼承

- 所謂繼承，是指類別物件的資源可以延伸和重複使用，在程式中可利用 *extends* 關鍵字來表達類別的繼承關係
- 這種延伸類別(*extends class*)的關係也就是 ” is a ” 的概念
- 當子類別繼承了父類別時，可使用父類別中的資源



繼承範例 (1)

- 使用方式
 - 在宣告class時加上extends
- class的繼承範例
 1. class A {
 2. int i,j;
 3. void showA() {
 4. System.out.println("i="+i+", j="+j);
 5. }
 6. }
 7. // class B繼承class A
 8. class B **extends** A {
 9. int k;
 10. void showB() {
 11. System.out.println("i="+i+", j="+j+",k="+k);
 13. }
 14. }

繼承範例 (2)

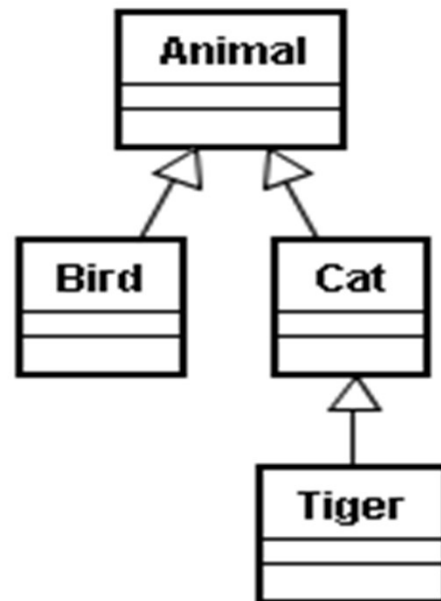
- 範例程式碼：

```
1. class Father {  
2.     public int money = 1000000;  
3.     public void undertaking() {  
4.         System.out.println("父親的事業");  
5.     }  
6. }  
7. class Son extends Father{  
8. }  
9. public class Extends {  
10.     public static void main(String[] args) {  
11.         Son son = new Son();  
12.         son.undertaking();  
13.         System.out.println("金額:" + son.money);  
14.     }  
15. }
```

執行結果：
父親的事業
金額:1000000

多型(Polymorphism)

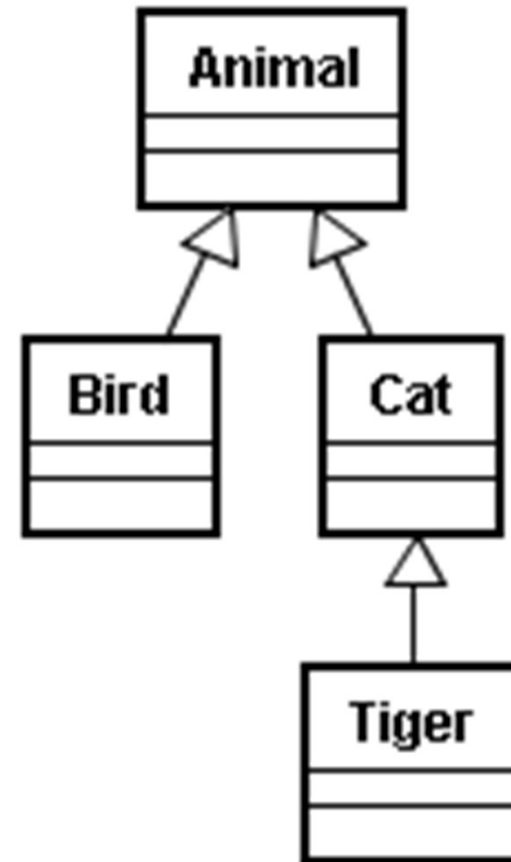
- 所謂多型，泛指在具有繼承關係的架構下，單一的物件實體可以被宣告成多種型別



多型（繼承多型）(1)

- Java 程式碼架構：

```
1. class Animal {  
2. }  
3. class Cat extends Animal {  
4. }  
5. class Bird extends Animal {  
6. }  
7. class Tiger extends Cat {  
8. }
```



多型（繼承多型）(2)

```
01 class Animal {  
02     String move() {  
03         return "動";  
04     }  
05 }  
06 class Cat extends Animal {  
07     String move() {  
08         return "跳";  
09     }  
10 }
```

```
01 class Tiger extends Cat {  
02     String move() {  
03         return "跑";  
04     }  
05     String skill() {  
06         return "獵殺";  
07     }  
08 }
```

```
Tiger tiger = new Tiger();  
System.out.println(tiger.move());  
System.out.println(tiger.skill());
```



跑
獵殺

多型（繼承多型）(3)

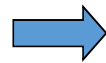
```
01 class Animal {  
02     String move() {  
03         return "動" ;  
04     }  
05 }  
06 class Cat extends Animal {  
07     String move() {  
08         return "跳" ;  
09     }  
10 }
```

```
01 class Tiger extends Cat {  
02     String move() {  
03         return "跑" ;  
04     }  
05     String skill() {  
06         return "獵殺" ;  
07     }  
08 }
```

```
Cat cat = new Cat();  
System.out.println(cat.move());  
  
Cat cat2 = new Tiger();  
System.out.println(cat2.move());  
  
System.out.println(cat2.skill());
```



跳



跑



在 Cat 型別中並不知曉 skill() 方法

作業01-01

- 請用您熟悉的物件導向程式語言，完成多型（繼承多型）的 Animal, Cat, and Tiger 的類別需告，並執行多型（繼承多型）
(3) System.out.println 的程式碼。

多型（介面多型）（1）

- 介面（interface）
 - 是一種規格用來規範程式實作時應遵循的標準
 - 單一類別可以實作多個介面，所以儘管在 Java 語言中沒有所謂的多重繼承的機制，我們仍可以利用實作多個介面類別的方式來達到多重繼承的效果
 - 介面也是類別的一種，它是用戶端程式碼(或類別)與類別之間的溝通管道，利用相同的介面可讓程式撰寫時有一定的規範可循

多型（介面多型）（2）

- implements 關鍵字
 - 在類別中利用 implements 關鍵字可以實作介面類別（實作多個介面類別時可用逗號，來區隔），語法如下：
 - 存取修飾元 class 類別名稱 implements 介面名稱1, 介面名稱2 ..., 介面名稱n
{ }
- 例

```
01. interface Animal {
02.     public String iPetName();
03.     public String iPetColor();
04. }
public class MyDog implements Animal {
    //程式中要實作 iPetName()與iPetColor()
}
```

多型（介面多型）(3)

```
01. public class MyDog implements Animal {
```

```
02.     public String iPetName() {
```

```
03.         return "小黑";
```

```
04.     }
```

```
05.     public String iPetColor() {
```

```
06.         return "黑色";
```

```
07.     }
```

```
08.     public static void main(String[] args) {
```

```
09.         MyDog Dog = new MyDog();
```

```
10.         System.out.println("我家有一隻狗名叫"
```

```
11.             + Dog.iPetName() +
```

```
12.             "，因為牠的顏色是"
```

```
13.             + Dog.iPetColor() +
```

```
14.             "的！");
```

```
15.     }
```

```
16. }
```

實作 *interface*
所定義的2個方法
*iPetName()*與
iPetColor()

```
public static void main(String[] args) {
```

```
    Ch00MyDog Dog = new Ch00MyDog();
```

```
    Dog.execute(Dog);
```

```
}
```

```
public void execute(Ch00IAnimal animal){
```

```
    System.out.println("我家有一隻狗名叫  
        "+animal.iPetName()+"，因為牠
```

```
    的顏色是"+animal.iPetColor()+"的！");
```

```
}
```

封裝(Encapsulation)

- 封裝是指，一種將抽象性函式介面的實作細節部份包裝、隱藏起來的方法。同時，它也是一種防止外界呼叫端，去存取物件內部實作細節的手段，這個手段是由程式語言本身來提供的。
 - 資料隱藏(Data Hiding)
 - 讓程式更容易維護與再利用

Data Hiding範例討論 (1)

```
1. public class Demo {  
2.     public int divider = 1;  
3.     public void dataHidingDemo(int number){  
4.         int result = number/this.divider;  
5.         System.out.println(result);  
6.     }  
7.     public static void main(String[] args) {  
8.         CH00Demo ch00Demo = new CH00Demo();  
9.         ch00Demo.divider = 0;  
10.        ch00Demo.dataHidingDemo(50);  
11.    }  
12. }
```

Data Hiding範例討論 (2)

```
1. public class Demo {  
2.     private int divider = 1;  
3.     public int getDivider() {  
4.         return divider;  
5.     }  
6.     public void setDivider(int divider) {  
7.         if(divider == 0)  
8.             System.out.println("Divider can not be 0!");  
9.         else  
10.            this.divider = divider;  
11.     }
```

```
12. public void dataHidingDemo(int number){  
13.     int result = number/this.divider;  
14.     System.out.println(result);  
15. }  
16. public static void main(String[] args) {  
17.     CH00Demo ch00Demo = new CH00Demo();  
18.     ch00Demo.setDivider(0);  
19.     ch00Demo.dataHidingDemo(50);  
20. }  
21. }
```

作業01-02

- 請用您熟悉的物件導向程式語言，完成Data Hiding範例討論 (2)的程式碼。

抽象化(Abstraction)

- 抽象化（英語：**Abstraction**）是將資料與程式，以它的語意來呈現出它的外觀，但是隱藏起它的實作細節。抽象化是用來減少程式的複雜度，使得程式設計師可以專注在處理少數重要的部份。
- 一個電腦系統可以分割成幾個[抽象層](#)（**Abstraction layer**），使得程式設計師可以將它們分開處理。[\[wikipaida\]](#)

抽象化- abstract class

- 抽象類別
 - 只定義方法的名稱與參數介面，但沒有方法內的程式碼
 - 由子類別來完成方法的內容
- 若父類別是一個抽象類別，其所產生的抽象方法都必須由子類別來重新實作
- 與實作介面類別的方法同
- 如果子類別沒有全部實作出父類別提供所有的抽象方法，則子類別必須宣告成抽象類別

抽象化- abstract class 範例(1)

```
1. abstract class Line {  
2.     private int length;  
3.     Line(int length) {  
4.         this.length = length;  
5.     }  
6.     int getLength() {  
7.         return length;  
8.     }  
9.     abstract double getArea();  
10. }
```

一般的實作方法

抽象方法

抽象化- abstract class 範例(2)

```
11. public class Square extends Line {  
12.     Square(int length) {  
13.         super(length);  
14.     }  
15.     double getArea() {  
16.         return Math.pow(getLength(), 2);  
17.     }  
18.     public static void main(String[] args) {  
19.         Square sq = new Square(10);  
20.         System.out.println("邊長為 " + sq.getLength() +  
21.             " 的正方形的面積 = " + sq.getArea());  
22.     }  
23. }
```

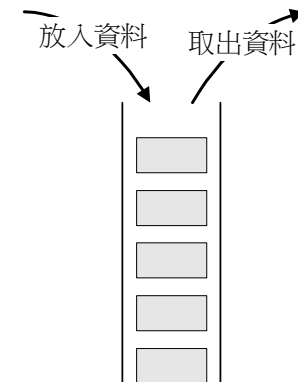
覆寫 Line 的
abstract method

Abstract Data Type (ADT)

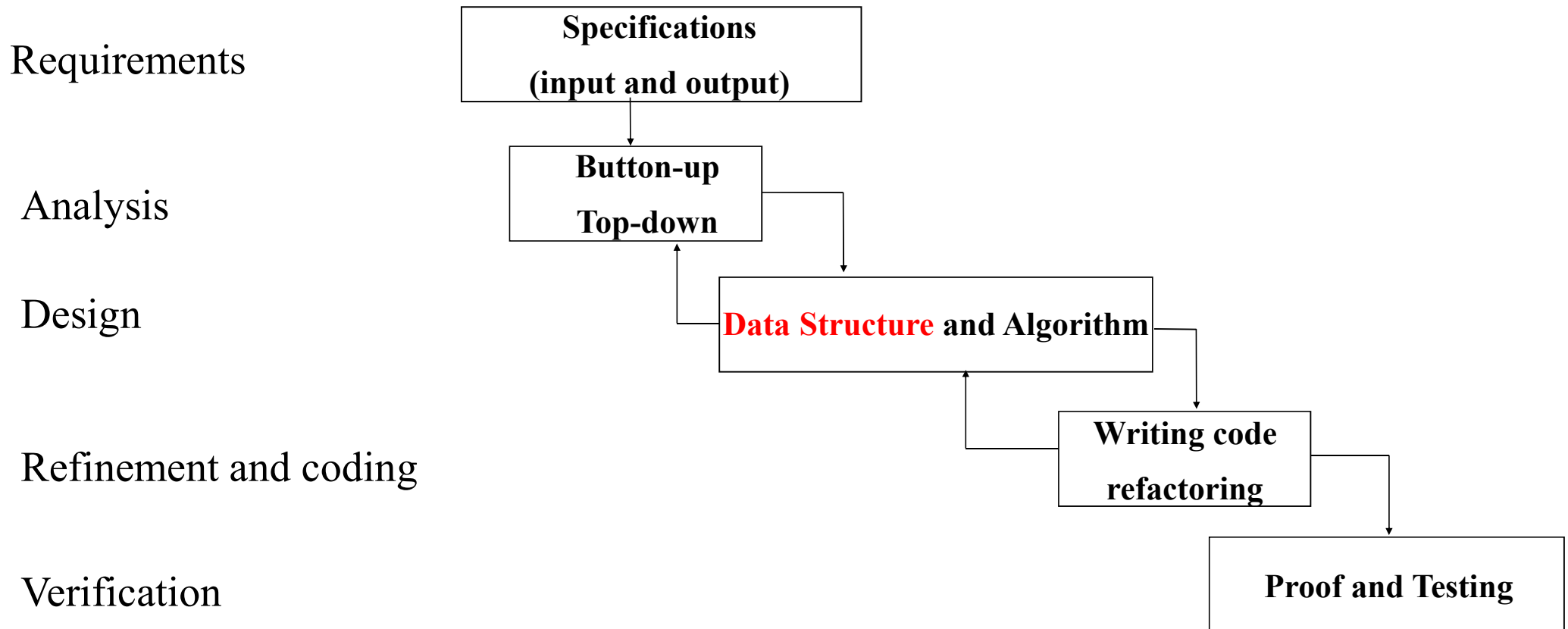
```
public abstract class Matrix {  
    private int[][] matrix;  
    IntMatrix( int nrows, int ncols );  
    public abstract int numRows();  
    public abstract int numCols();  
    public abstract int getItem(int row, int col);  
    public abstract void setItem(int row, int col, int value);  
    public abstract void scaleBy(int scalar);  
    public abstract void transpose();  
    public abstract void add(Matrix secondMatrix );  
    public abstract void subtract(Matrix secondMatrix );  
    public abstract void multiply(Matrix secondMatrix );  
    public abstract void printMatrix();  
}
```

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & -2 \\ 2 & -2 & -3 \end{bmatrix}$$

```
public abstract class Stack {  
    public abstract boolean isEmpty();  
    public abstract boolean isFull();  
    public abstract int Top();  
    public abstract void push(int value);  
    public abstract int pop();  
}
```



System Life Cycle



Algorithm decomposition V.S Object-Oriented decomposition

- Divide- and - Conquer
 - Software as a process
 - Decomposed into Functional modules
 - Structured programming
 - Algorithm -> data structure
 - Software as a set of well-defined objects
 - Decomposed into objects
 - Reuse of software
 - Object-Oriented programming
 - Data structure -> algorithm

Abstract Data Type (ADT)

- Predefined data type v.s user-defined data type
- An abstract data type is a data type that is organized in such a way that the **specification** of the objects and the specification of the operations on the objects is separated from the representation of the objects and the **implementation** of the operations.

```
public abstract class Counter {  
    private int count;  
    public Counter( )  
    public Counter(int initial)  
    public abstract int getCount( )  
    public abstract void increment( )  
    public abstract void increment(int delta)  
    public abstract void reset( )  
}
```


Abstract Data Type (ADT)

```
public abstract class Matrix {  
    private int[][] matrix;  
    IntMatrix( int nrows, int ncols );  
    public abstract int numRows();  
    public abstract int numCols();  
    public abstract int getItem(int row, int col);  
    public abstract void setItem(int row, int col, int value);  
    public abstract void scaleBy(int scalar);  
    public abstract void transpose();  
    public abstract void add(Matrix secondMatrix );  
    public abstract void subtract(Matrix secondMatrix );  
    public abstract void multiply(Matrix secondMatrix );  
    public abstract void printMatrix();  
}
```

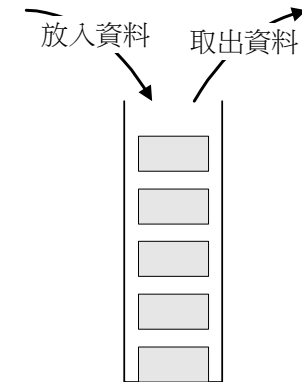
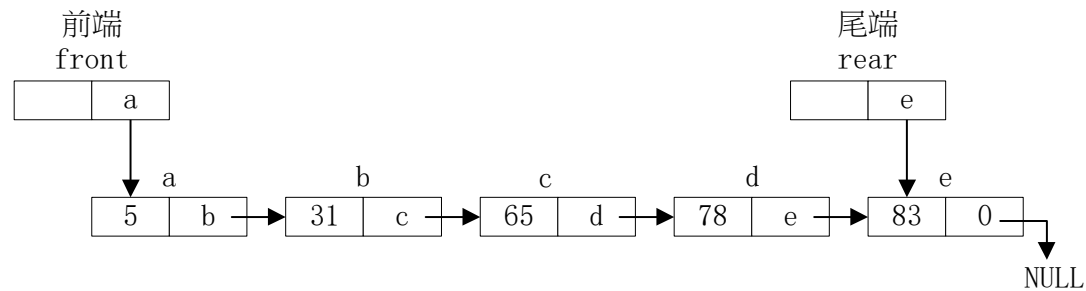
```
public abstract class Stack {  
    public abstract boolean isEmpty()  
    public abstract boolean isFull()  
    public abstract int Top()  
    public abstract void push(int value)  
    public abstract int pop()  
}
```

Data Structure

- **Array**

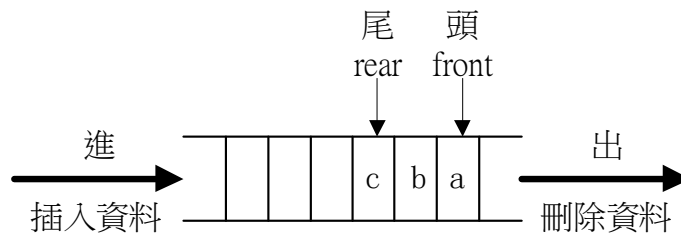
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]	X[9]
73	65	52	24	83	17	35	96	41	9

- **Linked List**

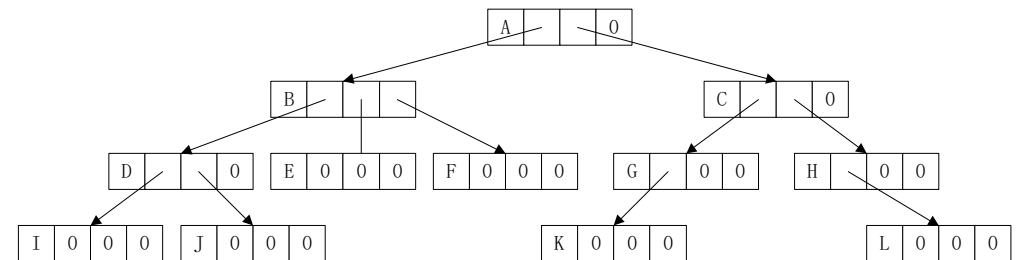


- **Stack**

- **Queue**



- **Tree**



演算法定義

- 演算法是一組可完成特定工作(解決特定問題、提供某功用)的有限指令集合，並且所有的演算法都需滿足下列條件：
 1. 輸入 (input)：可以有多個甚至是沒有輸入；
 2. 輸出 (output)：至少產生一個輸出；
 3. 明確(definiteness)：每個指令都清楚明確；
 4. 有限(finiteness)：在任何情況下，如果逐步追蹤演算法的每個指令，演算法會在有限的步驟內結束；
 5. 有效 (effectiveness)：原則上每個指令都需基本到人只需紙和筆即可實踐之，並且每個指令的運算不止如條件(3)般明確而已，還必須是可實行的。

排序

思考與探索：

欲將整數由小至大排序，可把數字小者放在左邊，數字大者放在右邊，...

- 可以挑出所有資料中最小者，做為左邊第一筆資料，接著再挑出剩下資料中最小者，放在左邊做為第二筆資料，依此類推，直至全部資料都排列完成。
- 若所有資料共計 n 筆，則會執行 n 次挑出最小的運算，其第 i 次的運算，即為挑出未排序資料中最小者，其結果做為第 i 筆資料。

2	8	5	4	7
---	---	---	---	---

挑選排序法程式碼

```
void SelectionSort(int data[], int n)
{
    int i, j;
    int min, temp;
    for (i=0; i<n; i++)
    {
        min = i;
        for (j=i+1; j<n; j++){
            if (data[j]<data[min])
                min = j;
        }
        temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}
```

2	8	5	4	7
---	---	---	---	---

挑選排序法演算法

輸入：data[0], data[1], data[2], ...,
data[n-1]，共 n 筆整數資料

輸出：data[0], data[1], ..., data[n-1]；其中
若 $i < j$ ，則 $\text{data}[i] \leq \text{data}[j]$ ， $1 \leq i, j \leq n$

```
for (i=0; i<n; i++)  
{  
    min = i;  
    for (j=i+1; j<n; j++){  
        if (data[j]<data[min])  
            min = j;  
    }  
    temp = data[i];  
    data[i] = data[min];  
    data[min] = temp;  
}
```

作業01-03

- 請寫一個function or method，`SelectionSort(int data[], int n)`，會執行挑選排序法

演算法的效率分析

- 什麼是有效率的演算法？電腦學家為此衡量準則提供了客觀的標準—分析演算法的執行時間和記憶體需求。以時間複雜度或空間複雜度來討論演算法的效率
 - 空間複雜度(space complexity)：一個程式或演算法所需的記憶體空間。
 - 時間複雜度(time complexity)：一個程式或演算法所需的執行時間；
- 解決相同的問題，演算法所用的時間和空間愈少愈好。

空間複雜度(space complexity)

- 代表演算法的空間需求，包含
 - Fixed space
 - Instruction space
 - 使用的變數及常數之空間
 - Variable space
 - 參數中有結構型別(如array, struct, object, ...)之參數且採用 call by value方式傳遞
 - Stack 空間 for recursion
- 空間複雜度只專注在Variable space

時間複雜度(time complexity)

指令執行的次數

- 陣列元素相加 (Add array members)

Java 片段程式：陣列元素相加	執行次數
<pre>public static int sum(int arr[], int n) { int i, total=0; for (i=0; i<n; i++) total += arr[i]; return total; }</pre>	<div>1</div> <div>n+1</div> <div>n</div> <div>1</div> <hr/> <div>$2n+3$</div>

時間複雜度(time complexity)

指令執行的次數

- 矩陣相乘 (Matrix Multiplication) $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$

Java 片段程式：矩陣相乘	執行次數
<pre>public static void mul(int a[][],int b[][],int c[][],int n) { int i, j, k, sum; for (i = 0; i < n; i++) for (j = 0; j < n; j++){ sum = 0; for (k = 0; k < n; k++) sum = sum + a[i][k] * b[k][j]; c[i][j] = sum; } }</pre>	$\begin{array}{r} 1 \\ n+1 \\ n(n+1) \\ n^2 \\ n^2(n+1) \\ n^3 \\ n^2 \\ \hline 2n^3+4n^2+2n+2 \end{array}$

時間複雜度(time complexity) 計算演算法所需要的執行時間

- 在程式或演算法中，每一敘述（**statement**）的執行時間為：
 - 此敘述執行的次數，
 - 每一次執行所需的時間，兩者相乘即為此敘述的執行時間。
- 由於每一敘述所須的時間必需實際考慮到機器和編譯器的功能，因此通常只考慮執行的次數而已。

時間複雜度(time complexity)

Big-O (1)

- 使用Big-O來表示程式的時間複雜度
- Big-O定義

Big-O 的定義如下：

$f(n)=O(g(n))$ ，若且唯若存在一正整數 c 及 n_0 ，使得 $f(n) \leq cg(n)$ ，對所有的 n ， $n \geq n_0$ 。

- 陣列元素相加
 - $f(n) = 2n + 3 = O(n)$, $c=3$, $n_0 = 3$
- 矩陣相乘
 - $f(n) = 2n^3 + 4n^2 + 2n + 2 = O(n^3)$, $c=3$, $n_0 = 5$

時間複雜度(time complexity)

Big-O (2)

(a) $3n+2=O(n)$ ， \therefore 我們可找到 $c=4$ ， $n_0=2$ ，使得 $3n+2 \leq 4n$

(b) $10n^2+5n+1=O(n^2)$ ， \therefore 我們可以找到 $c=11$ ， $n_0=6$ 使得 $10n^2+5n+1 \leq 11n^2$

(c) $7*2^n+n^2+n=O(2^n)$ ， \therefore 我們可以找到 $c=8$ ， $n_0=5$ 使得 $7*2^n+n^2+n \leq 8*2^n$

(d) $10n^2+5n+1=O(n^3)$ ，這可以很清楚的看出，原來 $10n^2+5n+1 \in O(n^2)$ ，而 n^3 又大於 n^2 ，理所當然 $10n^2+5n+1=O(n^3)$ 是沒問題的。同理也可以得知 $10n^2+5n+1 \neq O(n)$ ， $\therefore f(x)$ 沒有小於等於 $c*g(n)$ 。

Theorem 1-1

If $f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n^1 + a_0$, then $f(n) = O(n^m)$

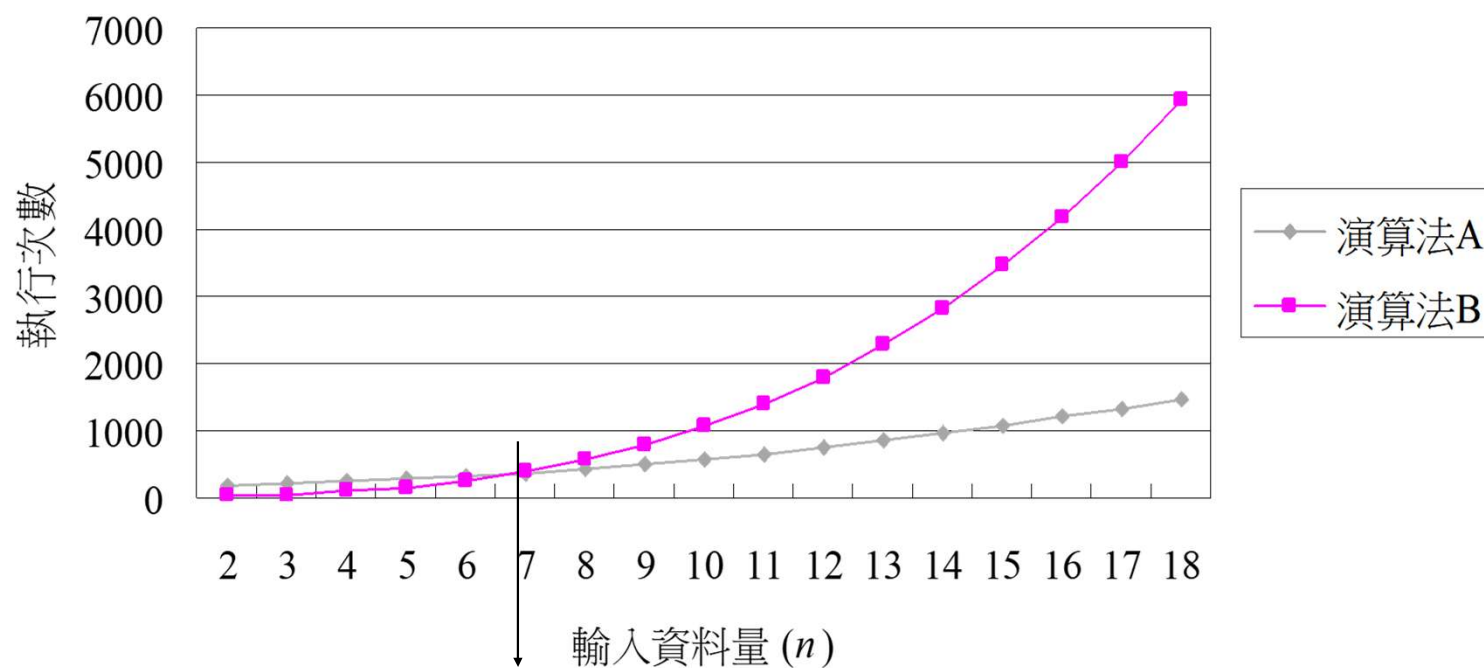
Proof:

$$\begin{aligned} f(n) &\leq \sum_{i=0}^m a_i n^i \\ &\leq n^m \sum_{i=0}^m a_i n^{i-m} \\ &\leq n^m \sum_{i=0}^m a_i, \text{ for } n \geq 1 \end{aligned}$$

So $f(n) = O(n^m)$

演算法優劣

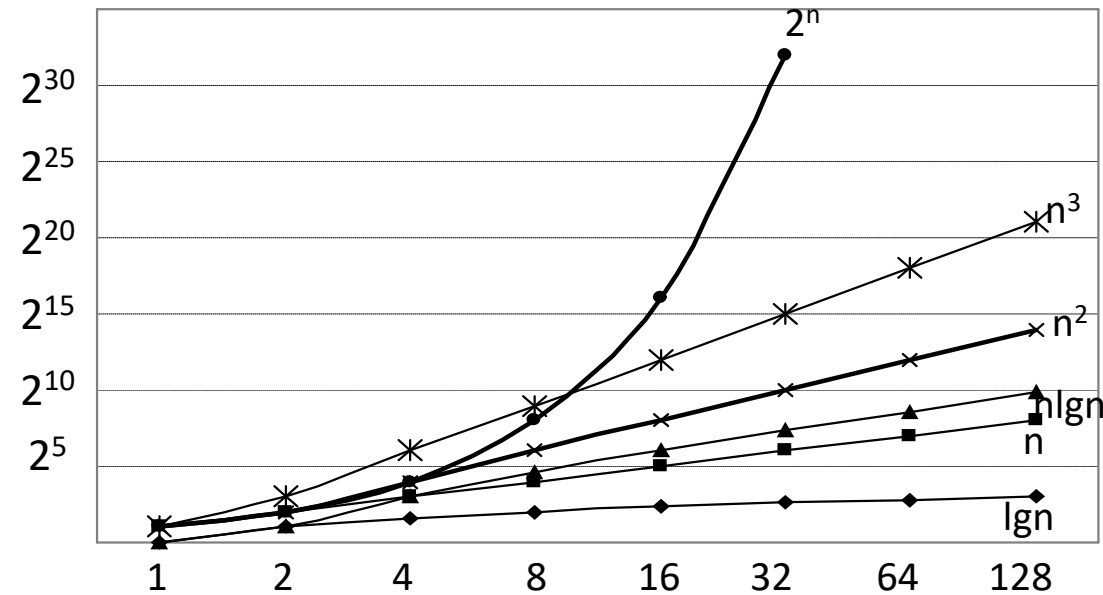
假設有兩個演算法都可解決問題 P ，其輸入資料量為 n ；演算法 A 的估算執行次數為 $4n^2+174$ ，演算法 B 的估算執行次數為 n^3+5n+6 。(見下圖)



在 $n>7$ 後 演算法 A 比 B 好

一般常見的Big-O

BigO	類別
$O(1)$	常數時間 (constant)
$O(\log n)$	對數時間 (logarithmic)
$O(n)$	線性時間 (linear)
$O(n \log n)$	對數線性時間 (log linear)
$O(n^2)$	平方時間 (quadratic)
$O(n^3)$	立方時間 (cubic)
$O(2^n)$	指數時間 (exponential)
$O(n!)$	階層時間 (factorial)
$O(n^n)$	n 的 n 次方時間



一般而言，這幾種類別由 $O(1)$ ， $O(\log n)$ ， \dots ， $O(n!)$ ， $O(n^n)$ 之效率按照排列的順序愈來愈差，也可以下一種方式表示。

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

作業01-04

- 決定以下的時間複雜度，以Big-O表示
 - $f(n) = n^2 \log n + \log n$
 - $f(n) = 8 \log \log n$
 - $f(n) = \log n^2$
 - $f(n) = n/100 + 10000/n^2$
 - $f(n) = \log n!$

循序搜尋 (sequential search)

```
public int SequentialSearch(int a[], int target) {  
    int targetIndex = -1;  
    for (int i = 0; i < a.length; i++) {  
        if (a[i] == target) {  
            targetIndex = i;  
            break;  
        }  
    }  
    return targetIndex;  
}
```

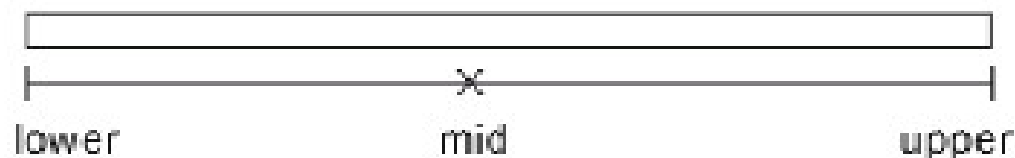
- 循序搜尋 (sequential search) 的情形可分，其平均搜尋到的次數為

$$\sum_{k=1}^n \left(k \times \frac{1}{n} \right) = \frac{1}{n} \times \sum_{k=1}^n k = \frac{1}{n} (1 + 2 + \dots + n) = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

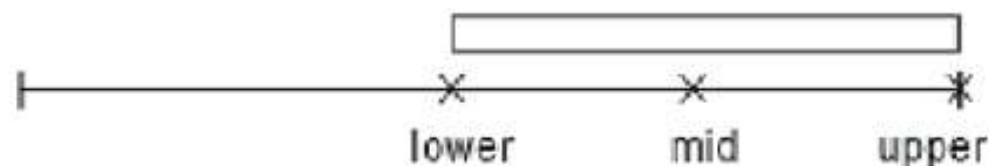
因此循序搜尋的 Big-O 為 $O(n)$ 。

二元搜尋法

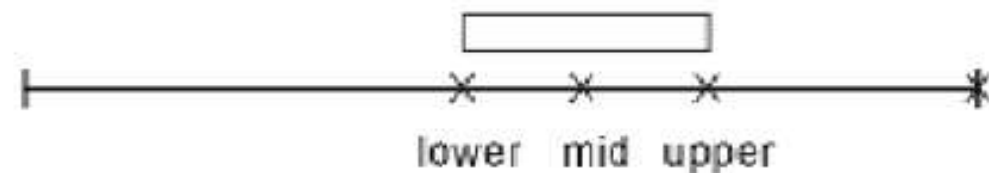
資料已經皆排序好，因此由中間（mid）開始比較，便可知欲搜尋的資料（key）落在mid的左邊還是右邊



當 $key > data[mid]$ 時，則 $lower = mid + 1$ ，upper不變，新的 $mid = (lower + upper) / 2$



當 $key < data[mid]$ 時，則 $upper = mid - 1$ ，lower不變，新的 $mid = (lower + upper) / 2$



二元搜尋法(3)

```
public int binarySearch(int a[],int target){
    int targetIndex=-1;
    int mid=-1;
    int lower=0;
    int upper=a.length-1;
    while(lower<=upper){
        mid=(lower+upper)/2;
        if(target>a[mid])
            lower=mid+1;
        else if(target<a[mid])
            upper=mid-1;
        else {
            targetIndex=mid;
            break;
        }
    }
    return targetIndex;
}
```

$O(\log n)$

作業01-05

- 請完成二元搜尋法的程式碼

循序搜尋 v.s 二元搜尋法

- 搜尋的次數為 $\log 32 + 1 = 6$ ，此處的 \log 表示 \log_2 。資料量為128個時，其搜尋的次數為 $\log 128 + 1$ ，因此當資料量為 n 時，其執行的次數為 $\log n + 1$ 。

陣列大小	二元搜尋	循序搜尋
128	8	128
1,024	11	1,024
1,048,576	21	1,048,576
4,294,967,296	33	4,294,967,296

費氏數列

- 費氏數列（Fibonacci number），其定義如下：

$$f_0=0$$

$$f_1=1$$

$$f_n=f_{n-1}+f_{n-2} \quad \text{for } n \geq 2$$

因此

$$f_2=f_1+f_0=1$$

$$f_3=f_2+f_1=1+1=2$$

$$f_4=f_3+f_2=2+1=3$$

$$f_5=f_4+f_3=3+2=5$$

⋮

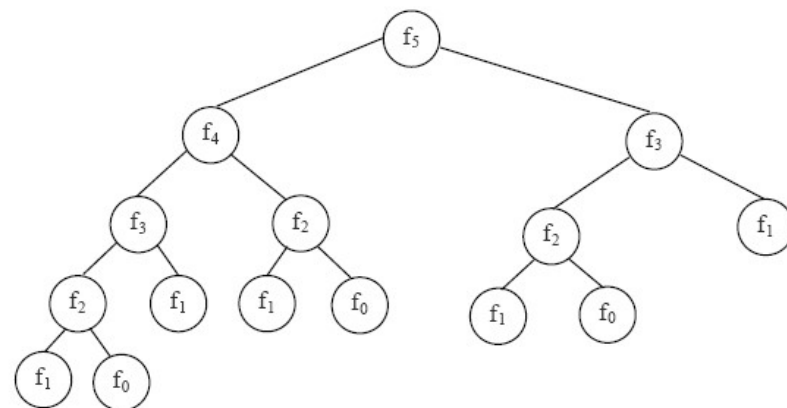
依此類推

費式數列 - 遞迴演算法

Java 片段程式：以遞迴方式計算費氏數列

```
public static int Fibonacci(int n)
{
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return (Fibonacci(n-1) + Fibonacci(n-2));
}
```

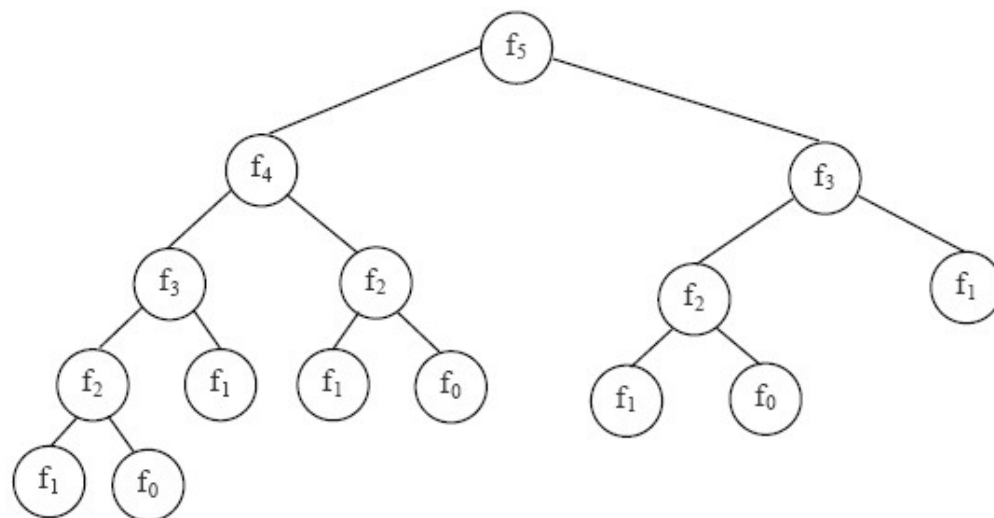
若以遞迴的方式進行計算的話，其圖形如下：



費式數列 - 遞迴演算法 $O(2^n)$

- $T(n) = T(n-1) + T(n-2)$
 $< T(n-1) + T(n-1)$
 $< 2^1 * T(n-1)$
 $< 2^2 * T(n-2)$
 ...
- $< 2^{n-1} * T(1)$
 $= 2^{n-1}$

若以遞迴的方式進行計算的話，其圖形如下：



費式數列 - 非遞迴演算法

Java 片段程式：以非遞迴方式計算費氏數列

```
public static int Fibonacci(int n)
{
    int prev1, prev2, item, i;
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else {
        prev2 = 0;
        prev1 = 1;
        for ( i = 2; i <= n; i++){
            item = prev1 + prev2;
            prev2 = prev1;
            prev1 = item;
        }
        return item;
    }
}
```

$O(n)$

作業01-06

- 請完成費式數列遞迴及非遞迴演算法的程式碼

Ω

Ω 的定義如下:

$f(n) = \Omega(g(n))$, 若且唯若, 存在正整數 c 和 n_0 , 使得 $f(n) \geq cg(n)$, 對所有的 n , $n \geq n_0$ 。

請看下面幾個範例:

(a) $3n+2 = \Omega(n)$, \because 我們可找到 $c=3$, $n_0=1$

使得 $3n+2 \geq 3n$

(b) $200n^2+4n+5 = \Omega(n^2)$, \because 我們可找到 $c=200$, $n_0=1$

使得 $200n^2+4n+5 \geq 200n^2$

Θ

Θ 的定義如下：

$f(n) = \Theta(g(n))$, 若且唯若，存在正整數 c_1 , c_2 及 n ，使得 $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ ，對所有的 n ， $n \geq n_0$ 。

我們以下面幾個範例加以說明：

(a) $3n+1 = \Theta(n)$ ， \because 我們可以找到 $c_1=3$ ， $c_2=4$ ，且 $n_0=2$ ，

使得 $3n \leq 3n+1 \leq 4n$

(b) $10n^2+4n+6 = \Theta(n^2)$ ， \because 只要 $c_1=10$ ， $c_2=11$ 且 $n_0=10$

便可得 $10n^2 \leq 10n^2+4n+6 \leq 11n^2$