CH1、Recursive & Complexity

基本概念：遞迴與時間複雜度

目錄：

Algorithm (演算法)

Def：為了完成一特定任務之有限個指令所組成之集合

⇨ 良好的演算法須滿足：

1. Input ≥ 0 個
2. Output ≥ 1 個
3. Definiteness (明確性)：指令是清楚、且不模糊的
4. Finiteness(有限性)：在有限的執行次數後可完成
5. Effectiveness(有效性)：利用紙、筆即可追蹤

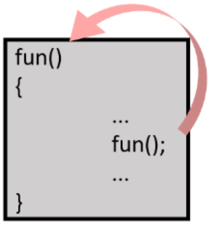
Note：演算法和程式差別在第 4 點(因為程式可以有無窮迴圈)

Recursive/Recursion(遞迴)

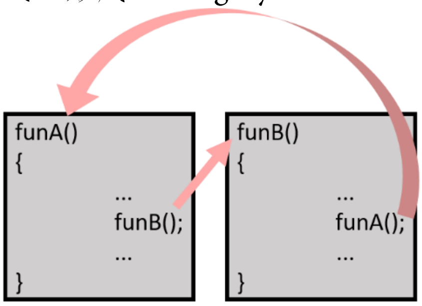
Def：函式在執行的過程會再度呼叫自身(Self-Calling)

Types:

1. 直接 Recursive

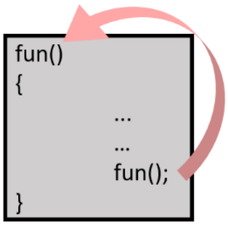


2. 間接 Recursive：在執行中呼叫另一函式，而另一函式又再度呼叫原本函式，形成 Calling Cycle



⇨ 非必要不建議使用

3. Tail(尾端)Recursive：可採用非遞迴改寫，目的：



⇨ 提高"系統效能" ，使用 loop(for, while, do ...while)

比較表

(可互相轉換)	Recursive	Non-recursive
特性	1. 程式精簡 2. 易理解 3. 表達性更佳	1. 較冗長 2. 不易理解 3. 表達性較差
時間	執行效率較差	執行效率較高(Speed up)
空間	1. 程式短=>省 Code Section 2. 需要系統的 Stack 支援 3. 需要的暫存變數少	1. 較耗 Space 2. 不需要系統的 Stack 支援 3. 需要的暫存變數多

函式定義(c/c++/java)

```
回傳型別        函式名稱        參數串列=>接收外部給予的資料
return types    function name   parameter list
void()不回傳/main()主程式
{
        Function
        或
        Body
}
```

常見的 Recursive 演算法：

1. 數學(7 種)：Non-Recursive
2. DS 各章談
3. 其他
   (1) Hanoi(河內塔)
   (2) Permutation(排列組合)

數學(7 種)：Non-Recursive

1. n!

   提示：

   $$n! = \begin{cases} 1 & , \text{if } n=0 \\ n*(n-1)! & , \text{otherwise} \end{cases}$$

   例：

```
int F(int n)
{
        if (n=0)
                return 1;
        else
                return n*F(n-1);
}
```

   呈上，問 F(3)=?　6

   又呼叫 F()幾次？　4*(含 F(3)本身)*

2. Σ (n)=1+...+n

提示：

Σ (n)=	{0	,if n=0
	Sum(n-1)+n	,otherwise}

例：

```
Sum(n)
{
	if(n=0)
		return 0;
	else
		return Sum(n-1)+n;
}
```

呈上，問Σ (3)=?　　6

又呼叫Σ ()幾次？　4*(含 Sum(3)本身)*

採 Non-recursive 寫Σ (n)=1+...+n

例：連加

```
int Sum(int n)
{
	int sum=0, i;
	for(i=1;i<=n;i++)
		sum=sum+i;
	return sum;
}
```

連乘

```
int Total(int n)
{
	int total=1, i;
	for(i=1;i<=n;i++)
		total=total*i;
	return total;
}
```

*(1) (10+1)*10/2，數據固定，複雜度：O(1)*

(2) 可代入 n，彈性比速度來得更重要

3. 費氏數列(Fibonacci number)

提示：

```
Fn=  {0                ,if n=0
      1                ,if n=1
      Fn-2+Fn-1  ,otherwise}
```

例：

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

概念：

n	0	1	2	3	4	5	6	7	8	9	10
Fn	0	1	1	2	3	5	8	13	21	34	55

呈上，問 Fib(5)=? 5

又呼叫 Fib()幾次？ 15*(含 Fib(5)本身)*

⇨ $O(2^n)$

Note:變形題(redefine)

```
Fn=  {1                ,if n=0
      1                ,if n=1
      Fn-2+Fn-1  ,otherwise}
```

概念：

n	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Fn	*1*	*1*	*2*	*3*	*5*	*8*	*13*	*21*	*34*	*55*	*89*

將上述改成 Iterative 的寫法

```
int Fib(int n)
{
        if(n=0) return 0;
        else if(n=1) return 1;
        else{
                int a=0,b=1,c,i,j
                for(i=2;i<=n;i++)
                {
                        c=a+b;
                        a=b;
                        b=c;
                }
        }
}
```

⇨ O(n)

4. Binomial Coefficient(二項式系數，較少考)

(n, m) = n! / m!(n-m)!

提示：

(n, m) =	{1	,if m=0 or n=m
	(n-1, m)+(n-1, m-1)	,otherwise}

例：

```
int bin(int n, int m)
{
        if(m==0 || n==m)
                return 1;
        else
                return bin(n-1, m)+bin(n-1, m-1);
}
```

呈上，問 bin(5,2)=?　　10

又呼叫 bin()幾次？　　19*(含 bin(5,2)本身)*

將上述改成 Iterative 的寫法

n! / m!(n-m)! 易造成數值過大而有 overflow 之問題，改善：先對消掉

```
int bin(int n, int m)
{
        int k=max(n-m, m);
        int a=1,b=1,i;
        for(int n;i>k;i--)
                a=a*i;
        for(int i;i<=n-k;i++)
                b=b*i;
        return a/b;
}
```

5. Ackerman's

提示：

A(m, n)=	{n+1	,if m=0
	A(m-1, 1)	,if n= 0
	A(m-1,A(m,n-1))	,otherwise}
A(2, 2)=7		
A(1, 2)=4		
A(2, 1)=5		

Recursive:

```
int A(int n, int m)
{
        if(m==0)
                return n+1;
        else if (n==0)
                return A(m-1, 1);
        else
                return A(m-1, A(m, n-1);
}
```

問 A(2, 2)=? 7

呼叫 A()幾次？ 27 次*(含 A(2, 2)本身)*

提示目錄：7 種數學題目

1. *n!*
2. *1+2+...+n-1*
3. *Fibonacci number*
4. *Binomial Coefficient*
5. *Ackerman's*
6. *GCD*
7. x^y

6. GCD(Greatest Common Divisor)

提示：

GCD(A, B)=	{B	,if(A mod B)=0
	GCD(B, A mod B)	,otherwise}

例：

```
int GCD(int A, int B)
{
	if(A%B==0)
		return B;
	else
		return GCD(B, A%B);
}
```

呈上利用 GCD，求 LCM(最小公倍數)

```
int LCM(int A, int B)
{
	return A*B/GCD(A, B);
}
```

改 Iterative 改寫 GCD

```
int GCD(int A, int B)
{
	while(A!=0 && B!=0)
	{
		if(A>B)	A=A%B;
		else	B=B%A;
	}
	if(A==0)	return B;
	else		return A;
}
```

7. x^y

提示：

y=	{1	,y=0
	x^y-1*x	,if y>0}

例：

```
int exp(int x, int y)
{
        if(y==0)
                return 1;
        else
                return exp(x, y-1)*x;
}
```

(1) exp(2, 10)= ?　　2 的 10 次方

(2) 此演算法的用途?　求 x 的 y 次方，即 x^y 為何

Iterative

```
int exp(int x, int y)
{
  int count=1;
  for(int i=0;i<y;i++)
        count = count *x;
  return count;
}
```

其他題目

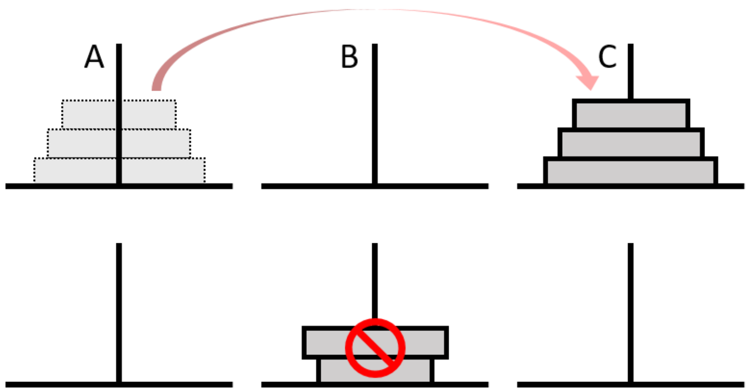
1. Tower of Hanoi(河內塔)

概念：

將 A 根的 Disc 搬到 C 柱，滿足：

(1) 一次搬一個

(2) 過程中，小的必在大的 Disc 之上



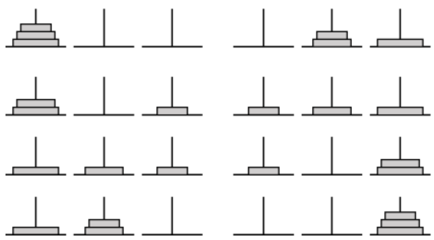
程式：(in Recursive)

```
void Hanoi(n: disc, A, B, C:pegs)
{
        if(n==1)
                move the disc from A to C;
        else
        {
                Hanoi(n-1, A, C, B);
                move disc n from A to C;
                Hanoi(n-1, B, A, C);
        }
}
```

⇨ $T(n) = T(n-1)+1+T(n-1) \ = 2T(n-1) +1 = 2^n -1 \Rightarrow O(2^n)$

例：n=3 將過程列出？

1　move 1 from A to C
2　move 2 from A to B
3　move 1 from C to B
4　move 4 from A to C
5　move 1 from B to A
6　move 2 from B to C
7　move 1 from A to C



程式：

```
void perm(char list[], int i, int n)
{
        if(i==n)
        {
                for(int j=1;j<=n;j++)           //印出當下的組合
                printf("%c,"list[j]);
        }
        else
        {
                for(int j=i;j<=n;j++)
                {
                        swap(list[i],list[j]);      //輪流當頭
                        perm(list,i+1,n);           //剩餘的繼續做排列組合
                        swap(list[i],list[j]);      //還原
                }
        }
}
```

考試方式：
(1) 演算法(結構化英文)
(2) 虛擬碼(結構化英文)
(3) 程式(最嚴緊的)

Recursive **演算法定義**

Data Type(資料型別) & Abstract Data Type *(ADT，可協助在物件導向語言中"定義類別")*
Def：由一組 Data Objects 反作用，在這些 Objects 之上的 Operation 所組成
例：int 為 Data Type，Objects:0~9，Operation：+-*/

總結：ADT 只定義，不對 data 的呈現或製作進一步描述(only what ,but no how)

ADT (**抽象化資料型別**)

ADT is a spec(規範), of:
1. a set of data
2. the set of operations perform on the data

例：Stack 是 ADT，具 top(指向頂端)、push(加入)、pop(取出)…只"定義"，不實作

Note:
1. *ADT 亦是 data type*
2. *提供操作介面和製作的細節分開*

(以軟工角度而言：可達到 loosely couple)耦合力下降

從程式或系統開發想達到的：
1. *內聚力高(同一模組下的關聯性高)*
2. *耦合力低(兩個模組之間的關聯性越低越好)*

3. *ADT 如同只定義運作規範，實作留由後面進行。以 java 而言 ADT 如同"介面 interface"*
   例：Public abstrack void walk();　　//只定義不實作

Animal	Dog	…
Walk()	Walk()	…

提示目錄：
演算法
Recursive
ADT
Analysis
　　Space(空間)
　　Time(時間，較重要)

Algorithm / Program 之效能上的評估

1. Space Complexity:
   在 Input 數量不同之下所須的"空間"變化量
   越快提升表示："複雜度越高"

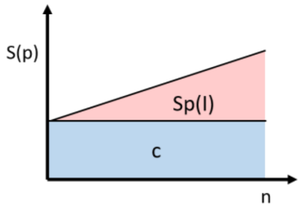
2. Time Complexity:
   在 Input 數量不同之下所須的"時間"變化量
   越快提升表示："複雜度越高"

Space Complexity(空間複雜度)

Def: 令 s(p)為 algo/prog p 之 space 需求，則：

s(p)=c + Sp(I)

固定+ 變動



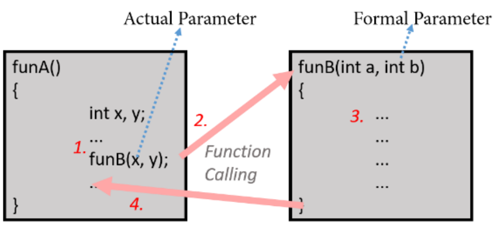
C(固定)包含：

1. Instruction Space(指令)
2. Variable Space(變數)
3. Constant(常數)
4. Fixed size structure variable(結構變數：array, struct, record...)

Sp(變動)包含：

1. 參數是結構型，且採用"call-by value"方式
2. Recursive Algorithm 執行所須的 Stack

[補充]：參數傳遞？



Function Calling 會做：

1. 將 funA 相關資訊 save (into Stack)
2. 將參數給予被呼叫函式*(Accept Parameter(AP)：接受參數；Formal Parameter(FP)：顯示參數)*
3. 被呼叫函式執行
4. 回傳結果，並繼續執行之前的函式 (pop from Stack，若 Stack 空，則停止)

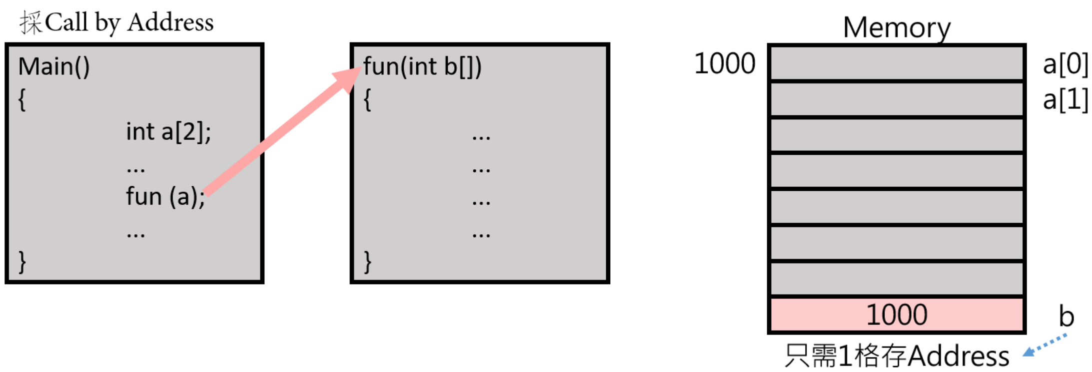
參數傳遞：

Call-by-Value(傳值呼叫)	Call-by-Address(傳址呼叫)
Actual Parameter 跟 Formal Parameter 會佔用不同的記憶體空間	Actual Parameter 跟 Formal Parameter 操作相同的記憶體空間
沒有副作用	有副作用
C/C++：一般變數預設為此種方式，int、floot、double	C/C++：指 Formal Parameter 的更動會影響到 Actual Parameter 的數值：結構型(Array)用此方式

例：

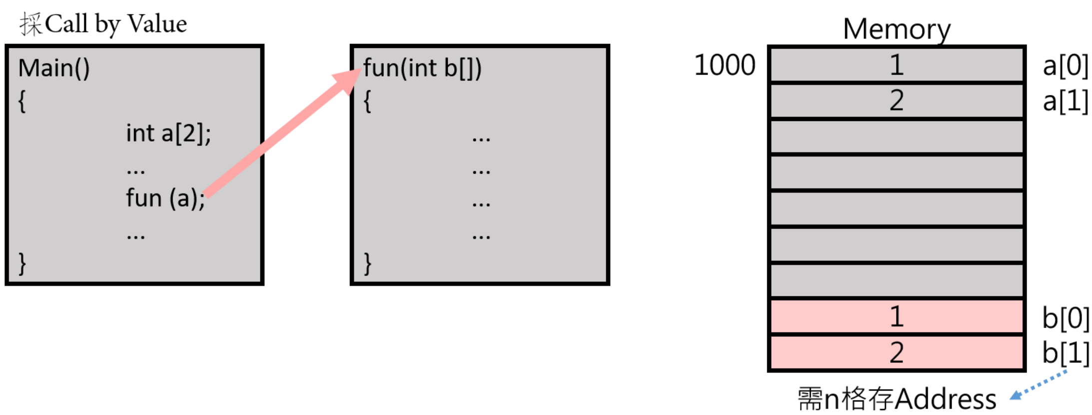
1. Call-by-Address

   b 只需一格，並與 a 操作同一格，須 n+1 格



2. Call-by-Value

   兩者無關係，自己找出自己需要的格子，須 2n 格



例 1：

```
float abc(float a, float b)
{
        rerturn a+b*c+(a+c)/4;
}
```

Sp(I)：0

例 2：

```
float sum(float list[], int n)
{
	float temp=0;
	int i;
	for(i=0;i<=n;i++)
	{
		temp+=list[i];
	}
	return list[0];
}
```

1. *若 Call-by-Value:* *Sp(I)=n*
2. *若 Call-by-Address:* *Sp(I)=0*

例 3：

```
float rsum(float list[], int n)
{
	if(n!=0)
		return rsum(list, n-1)+list[n-1];
}
```

採 Call-by-Address 且：int=2 bytes、float=4 bytes、address=2 bytes

Recursive 呼叫會將：

1. *Parameter(目前參數)*
2. *Local variable(區域變數)*
3. *Return address(之後的返回位址)*

(以上三點為：活動記錄(activation record)，在副程式呼叫的時候，會產生一個活動記錄，存於 Stack 中)

目的：是為了後續可順利回到原主程式中

由上可得知：呼叫一次 Recursive 須：

1. *Parameter:* *2(address)+2(int)*
2. *Local variable:* *NO*
3. *Return address:* *2*

⇨ *6 bytes*

*因了 n+1 次，故需 Sp(I)=(n+1)*6 bytes*

Time complexity(時間複雜度)

Def: 令 T(p)為 Algorithm/Program p 之所須時間，則:

T(p)=	coding	+	execution time
	development		runtime
	cost		benefit

Execution Time 衡量方式:

1. 實際執行，算 CPU 時間(cost high…不用)
2. 分析/預估方式

⇨ 分析 what?

計算 algo 之中"指令"執行次數

考型：

1. *計算執行次數*
2. *Time Complexity*
3. *Recursive Function Time Complexity*

1. 計算執行次數

例 1:

```
for(i=0;i<rows;i++)
{
	for(j=0;j<cols;j++)
	{
		A[i][j]=0;
	}
	count++;	//j loop 失敗
}
```

*count++;		//i loop 失敗*

*rows(1+2*cols+1)+1 => 2rows + 2*roows*cols +1*

例 2:

```
float sum(float list[], int n)
{
	float temp=0;
	int i;
	for(i=0;i<n;i++)
	{
		temp+=list[i];
	}
	return temp;
}
```

1+2n+1+1=2n+3

另解：指一個"指令"執行一次須多少個步驟

statement	*S/E(Step/Execution)*	*Frequency*
temp=0;	*1*	*1*
for(i=0;i<n;i++)	*1*	*(n+1)*
	1+n+1 = n+2	

例 3:

```
float rsum(float list[], int n)
{
	if(n)
	{
		return rsum(list,n-1)+list[n-1];
	}
	return list[0];
}
```

n+1 +n +1 = 2n+2

例 4-1:

```
for i=1 to n do
	for j =1 to n do
		x=x+1
```

問 x=x+1 須做幾次？ $n*n=n^2$

例 4-2:

```
for i = 1 to n do
	for j = 1 to i do
		x=x+1
```

問 x=x+1 須做幾次？

1. *1+...+n = n(n+1)/2*
2. *Sum 1-n (Sum 1-i(1)) = Sum 1-n(i) = n(n+1)/2*

例 5:

```
for i = 1 to n do
	for j = 1 to i do
		for k = 1 to j do
			x=x+1
```

問 x=x+1 須做幾次？

n(2n+1)(n+1)/12 + n(n+1)/4 = n(n+1)(n+2)/6

例 6:

```
for k = 1 to n do
	for i = 1 to k do
		for j = 1 to k do
			if(i!=j) x=x+1;
```

問 x=x+1 須做幾次？

n(2n+1)(n+1)/6 - n(n+1)/2 = n(n+1)(n-1)/3

例 7-1:

```
i = n
while (i>=1) do
begin
        i=i/2;
        x=x+1;
end
```

問 x=x+1 須做幾次？

$log_2(n)+1$

例 7-2:

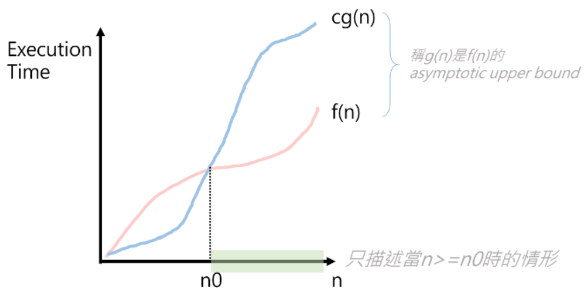
```
i=1
while (i<=n) do
begin
        i=i*2;
        x=x+1;
end
```

問 x=x+1 須做幾次？

$log_2(n)+1$

思考： O(n) => 利用一個衡量方式，可快速了解一個演算法/程式之複雜度 ⇨ 採用"漸近表示法"

2. Time Complexity 之"漸近表示法"符號
   (1) Big Oh(*$O()$*)：Upper Bound of f(n) *(上限)*
   (2) Omega(*$\Omega()$*)：Lower Bound of f(n) *(下限)*
   (3) Theta(*$\Theta()$*)：More precise than O and Ω *(上下限之間的區域)*



[補充]

```
x=2;
while(x<=n)
{
        x=x*x;
}
```

問 x=x*x 執行幾次？

$n=2^{(2^k)}$, $logn=2^{k*log2}$ => $log(logn)=k$ => $log(logn)+1$ 次

Big Oh

Def：若 f(n)為一時間函數，則 f(n)=O(g(n))，iff 存在 2 個正數 c 和 n_0，使得 $f(n) \le c*g(n)$, for all $n \ge n_0$

例 1：

f(n) = 3n+2

⇨ *O(n)*

c=4, n_0=2

例 2：

$f(n) = 2n^2+4n+3$

⇨ *$O(n^2)$*

c=3, n_0=5

例 3：

f(n) =3n+2 => O(1)

is error

定理： $a_m n^m + a_{m-1} n^{m-1} + \ldots + a_0 n^{a0}$ $f(n) = O(n^m)$

常用的 Big Oh：　小到大

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

常數 < 對數 < 線性 < 多項式　< 指數項

例 1：

$f(n) = 2^n + 100n + 8$

⇨ *$O(2^n)$*

例 2：

$f(n) = n^3+2000n^2+5$

⇨ *$O(n^3)$*

例 3-1：

$\sum i = n(n+1)/2$

⇨ *$O(n^2)$*

例 3-2：

$\sum i^2 = n(n+1)(n+2)/6$

⇨ *$O(n^3)$*

例 4：

f(n)= 99

⇨ *O(1)*

例 5：
$\sum 1/i = \sum i^{-1} = \log n$
⇨ *O(log n)*

例 6：
$O(\log n^2)$與 $O(n)$，誰大？
⇨ *O(n)*

例 7：
$f(n) = 3n^2+8$，下列何者正確？
1. $f(n)=O(n^2)$
2. $f(n)=O(n)$
3. $f(n)=O(n^3)$

1、3；註(3)：以 Tightly Upper Bound 而言為錯誤；但以定義而言正確

Omega(Ω)

Def: 若 f(n)為一時間函數，則 $f(n)=O(g(n))$，iff 存在 2 個正數 c 和 n_0，使得 $f(n) \geq c*g(n)$, for all $n \geq n_0$

例：n^2 與 $2^{\log n}$，何者較大？

雙邊取 log => $\log(n^2)$、$\log(2^{\log n})$ => $\log(n^2)$、$(\log n)(\log 2)$ => $\log(n^2)$、$\log(n)$，故 n^2 較大

例：$\log(n!) = \Omega(n\log n)$

$\log(n!) \geq \log(n/2^{n/2}) = n/2 \log n/2 = n/2[(\log n) - 1]$

Theta(Θ)

Def: $f(n) = \Theta(g(n))$, iff 存在 3 個正數，c_1,c_2,n_0，使得： $c_1*g(n) \leq f(n) \leq c_2*g(n)$

例 1：$\Omega(n) + \Theta(n^2) =?$
$[n, infinite] + [n^2, n^2] = [n^2, infinite] = \Omega(n^2)$

例 2：$\Omega(n) + O(n^2) = ?$
$[n, infinite] + [1, n^2] = [n, infinite] = \Omega(n)$

例 3：$\Theta(n) + O(n^2) = ?$
$[n, n] + [1, n^2] = [n, n^2] = O(n^2)$ or $\Omega(n)$

遞迴時間函式求解
展開法
Master Method
Extended M. M.
Recursive Tree
[離散]特性方呈式(費氏)
Guess+Proof
替換法

展開法

例 1：河內塔

$T(n) = 2T(n-1) + 1$, $T(0)= 0$
$= 4T(n-2)+1 +1$
...
$= 2T(0) + 2^n -1$
$=> O(2^n)$

例 2：Binary Search

$T(n)= T(n/2) + 1$,$T(0)=1$
$= T(n/4) + 1 +1$
...
$= T(n/n) + logn$
$=> O(log\ n)$

例 3-1：Quick Sort 之 Worst case:

$T(n)= T(n-1) + n$
$= T(n-2) + (n-1) + n$
...
$= T(n-n) + 1 + ... + n$
$= n(n+1)/2$
$=> O(n^2)$

例 3-2：Quick Sort 之 Best case:

$T(n)= 2T(n/2) + n$, $T(1)=1$, $T(0)=0$
$= 2[2T(n/4) + n/2] + n$
$= 4T(n/4) + n + n$
...
$= nT(n/n) + logn * n$
$= n + nlog\ n$
$=> O(nlog\ n)$

例 4-1：

$T(n)= T(sqrt(n)) + 1$, $T(2) = 1$

$= T(n^{1/2}) + 1$

$= T(n^{1/4}) +1 +1$

$= T(n^{1/8}) +1 +1 +1$

...

$= T(n^{(1/2)^i}) + i$

$= 1 + log(log\ n)$

$=> O(log(log\ n))$

例 4-2：

$T(n)= 2T(sqrt(n)) + log\ n$, $T(2) = 1$ $T(n^{1/2}) = 2T(n^{1/4}) + log(n^{1/2})$

$= 2T(n^{1/2}) + log\ n$

$= 2[2T(n^{1/4}) + log(n^{1/2})] + log\ n$

$= 4T(n^{1/4}) + 2log(n^{1/2}) + log\ n$

$= 4T(n^{1/4}) + log\ n + log\ n$

$= 4T(n^{1/4}) + 2log\ n$

...

$= 2^i\ T(n^{1/2^i}) + i*log\ n$ $n^{(1/2)^i} = 2$；$(1/2)^{i(log\ n)} = 1$；$2^i = log\ n$；$i = log(log\ n)$

$= log\ n*T(2) + log(log\ n)*log\ n$

$= log\ n + log(log\ n)*(log\ n)$

$=> O(log(log\ n)*(log\ n))$

Time Complexity (**時間複雜度**)

執行次數

漸近式曲線

遞迴函式求解

展開代入法

Master Method (支配理論)

Extended Master Method

Recursive Tree

Master Method

Def: 若 $T(n)=aT(n/b)+f(n)$

其中{$a\geq1, b>1$；$f(n)$為 Postive Function}，即可採 Master Method 求解

上述 $T(n)$為演算法中"Divide and Conquer (分割與克服)"

1. Divide：Sub Problem
2. Conquer：Sub-Solution
3. Combination：Total Solution

Master Method 跟上述之關係：

$T(n)=aT(n/b) + \quad f(n)$

2   1,3

1.a => Sub Problem 之數目

2.n/b => 各 Sub-Problem 之 Data size

3.T(n/m) => Sub-Problem 之處理時間

4.f(n) => 為 n 筆 Data 做 Divide, Combine 之時間

Master Method 說明：

步驟：

step1: 先求出 $n^{\log b(a)}$

step2: case1=> $n^{\log b(a)} > f(n) \Rightarrow \Theta(n^{\log b(a)})$

case2=> $n^{\log b(a)} = f(n) \Rightarrow \Theta(n^{\log b(a)} * \log n)$

case3=> $n^{\log b(a)} < f(n) \Rightarrow \Theta(f(n))$

前提：$af(n/b) \leq cf(n)$，其中常數 c 為<1

例 1：

$T(n)=4T(n/2)+n$

$a=4 \quad b=2 \quad f(n)=n$

$n^{\log 2(4)}=n^2$

∴為 Master Method 中的 case1

$\Rightarrow \quad \Theta(n^2)$

例 2：

$T(n)=2T(n/2)+n$

$a=2 \quad b=2 \quad f(n)=n$

$n^{\log 2(2)}=n$

∴為 Master Method 中的 case2

$\Rightarrow \quad \Theta\ (n\log n)$

例 3：

$T(n)=2T(n/4)+n$

$a=2 \quad b=4 \quad f(n)=n$

$n^{\log 4(2)}=n^{1/2}$

∴為 Master Method 中的 case3

又 $2f(n/4) \leq 1/2f(n) < 1$

$\Rightarrow \quad \Theta(f(n)) = \Theta(n)$

例 4-1：

$T(n)=T(9n/10)+n$

$a=1 \quad b=10/9 \quad f(n)=n$

$n^{\log 10/9(1)}=1$

∴為 Master Method 中的 case3

又 $f(9n/10) \leq 9/10f(n) < 1$

$\Rightarrow \quad \Theta(f(n)) = \Theta(n)$

例 4-2：

$T(n)=16T(n/4)+n^2$

$a=16$ $b=4$ $f(n)=n^2$

$n^{log4(16)}=n^2$

∴為 Master Method 中的 case2

=> $\Theta(n^2*log\ n)$

例 5-1：

$T(n)=7T(n/3)+n^2$

$a=7$ $b=3$ $f(n)=n^2$

$n^{log3(7)}=n^{1....}$

∴為 Master Method 中的 case3

又 $7f(n/3) \leq 7/9f(n) < 1$

=> $\Theta(f(n)) = \Theta(n^2)$

例 5-2：

$T(n)=7T(n/2)+n^2$

$a=7$ $b=2$ $f(n)=n^2$

$n^{log2(7)}=n^{2....}$

∴為 Master Method 中的 case1

=> $\Theta(n^{log2(7)})$

例 5-3：

$T(n)=7T(n/4)+n^2$

$a=7$ $b=4$ $f(n)=n^2$

$n^{log4(7)}=n^{1....}$

∴為 Master Method 中的 case3

又 $7f(n/4) \leq 7/16f(n) < 1$

=> $\Theta(f(n)) = \Theta(n^2)$

例 6：

$T(n)=4T(n/16)+n^{1/2}$

$a=4$ $b=16$ $f(n)=n^{1/2}$

$n^{log16(4)}=n^{1/2}$

∴為 Master Method 中的 case2

=> $\Theta(n^{1/2} * log\ n)$

Extended Master Method (E.M.M)

Def：若 $T(n)=aT(n/b)+ n^{logb(a)} * log^k(n)$

此時不適用 case 3，須採 Extended Master Method

即 $T(n)= \Theta(n^{logb(a)} * log^{k+1}(n))$

例 1：

$T(n)=2T(n/2) + n\log n$　求 $T(n)=\Theta(?)$

$a=2$　$b=2$　$f(n)= n\log n$

$n^{\log 2(2)}=n$

∴為 Master Method 中的 case3

又 $n^{\log 2(2)}=n < n\log n = f(n)$

且 $f(n)= n^{\log b(a)} * \log n = n \log n$

=> 為 Extended Master Method，即 $T(n) = \Theta(n \log^2(n))$

例 2-1：

$T(n)=4T(n/2) + n^2\log n$ 求 $T(n)=\Theta(?)$

$a=4$　$b=2$　$f(n)= n^2\log n$

$n^{\log 2(4)}=n^2$

∴為 Master Method 中的 case3

又 $n^{\log 2(4)}=n^2 < n^2\log n = f(n)$

且 $f(n)= n^{\log b(a)} * \log n = n^2 \log n$

=> 為 Extended Master Method，即 $T(n) =\Theta(n^2 \log^2(n))$

例 2-2：

$T(n)=16T(n/4) + n^2\log^2(n)$　求 $T(n)=\Theta(?)$

$a=16$ $b=4$　$f(n)= n^2\log^2(n)$

$n^{\log 4(16)}=n^2$

∴為 Master Method 中的 case3

又 $n^{\log 4(16)}=n^2 < n^2\log^2(n) = f(n)$

且 $f(n)= n^{\log b(a)} * \log n = n^2 \log^2(n)$

=> 為 Extended Master Method，即 $T(n) = \Theta(n^2 \log^3(n))$

Recursive Tree

試想：$T(n)=T(n/3) + T(2n/3) + n$，求 $T(n)=\Theta(?)$

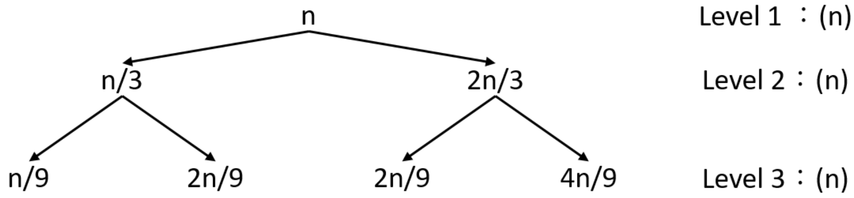
不符合 Master Method 型態，不可採用，故利用 Recursive Tree

case1：各 level 之 cost 相同　　=> Θ(level cost * log n)

case2：各 level 之 cost < first level cost　　=> Θ(first level cost)

Case 1 概念：

$T(n)=T(n/3) + T(2n/3) + n$：



$(2/3)^h * n = 1$

$n = 1/(2/3)^h = (3/2)^h$

$\log n = h \log (3/2)$

可知：Level cost=n，又有 log n 層，故 $T(n) = \Theta(n \log n) = O(n \log n)$

例 1：

T(n) = T(n/4) + T(3n/4) + n

$\Theta(n*\log n) = O(n*\log n)$

例 2：

$T(n) = T(n/7) + T(3n/4) + n^2$

$n^2/49 + 9n^2/16 < n^2$ => case 2

Θ(first level cost) = $T(n) = \Theta(n^2)$

[補充]

T(n) = T(n/4) + T(n/4) + T(n/4) + n = 3T(n/4) + n

a=3 b=4 f(n)=n

$\log_b(a) = \log_4(3) = 0... < n$

case3

3T(n/4) = 3/4T(n) < 1

=> $\Theta(f(n)) = \Theta(n)$

[離散]費氏數列之時間函式

例：T(n) = T(n-1) + T(n-2) , T(0) = 0 , T(1) = 1

利用特性方程式：

$x^2 = x + 1 \Rightarrow x^2 - x + 1 = 0$

得 2 根為 $x_1 = [1+\sqrt{(5)}] / 2$

$x_2 = [1-\sqrt{(5)}] / 2$

$T(n) = A x_1^n + B x_2^n$

$= A ([1+\sqrt{(5)}]/2)^n + B ([1-\sqrt{(5)}]/2)^n$

又 $T(0) \Rightarrow A*1 + B*1 = 0$

又 $T(1) \Rightarrow A*[1+\sqrt{(5)}]/2 + B*[1-\sqrt{(5)}]/2 = 0$

得 $A = 1/\sqrt{(5)}, B = -1/\sqrt{(5)}$

$\therefore T(n) = 1/\sqrt{(5)}*\{[1+\sqrt{(5)}]/2\}^n - 1/\sqrt{(5)}*\{[1-\sqrt{(5)}]/2\}^n$

$T(n) = O(([1+\sqrt{(5)}]/2)^n) \Rightarrow O(2^n)$

Guest + Proof

例：

T(n) = 2T(n/2 + 17) + n

當 n 很大時：n/2 + 17 => n/2

T(n) = 2T(n/2) + n

採 Master Method => case2

T(n) = O(nlogn)

替代法（函數轉型）

例：

$T(n) = T(\sqrt{}(n)) + 1$

$= T(n^{1/2}) + 1$

令 $n = 2^m$ (即 $m = \log n$)

$T(n) = T(\sqrt{}(n)) + 1 \Rightarrow T(2^m) = T(2^{m/2}) + 1$

令 $s(m) = T(2^m) \Rightarrow s(m) = s(m/2) + 1$

採 Master Method => case2

$s(m) = O(\log n)$

$T(n) = O(\log(\log n))$

補充：

Josephus 問題：

例(95 中央)：將 n 人圍成圓圈，並以下列方式一一淘汰(最終只選一人)：
2, 4, 6, …淘汰，即：選一人、淘汰一人…以此類推，最終將選出幾號？

設一開始有 2k 人，則執行一圈剩下：1, 3, 5, …, 2k-1

第二圈：1, 5, 9, …即前一輪加 1 除 2

$\Longrightarrow f(2k) = 2f(k) - 1$　　//前一輪=2×後一輪 -1

故得：

$f(1)=1$, $f(n)=2f(n/2)-1$, if n is even；$f(n)=2f((n-1)/2)$, if n is odd

由簡單的觀察可發現並猜測：$f(n) = 2k + 1$

再利用數學歸納法證明之

$\Longrightarrow f(n)=2^k+1$，其中 $n=2^m+k$, $m=[\log_2 n]$