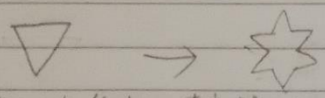
Date: NO: Check

recursion (遞迴)

定義：用一個小程式，重複呼叫，處理得資料大→小

缺：速度慢　　優：容易看懂

ex.

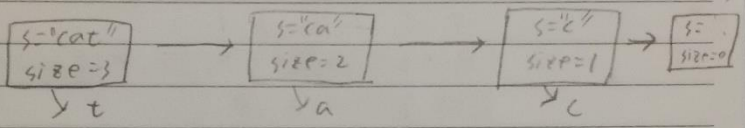


(fractal) (碎形)

特色：大化小、寫法單純

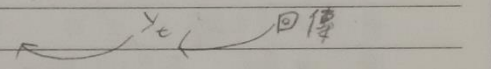
ex. 反轉字串：cat → tac　　回傳

過程：



=) tac

或 s="cat" → s="at" → s="t" → s=""



重點：一定要有終點　→ 沒有終點 = 崩潰

ex: 加法 =)例 5個1相加

迴圈 =) 1+1=2 =) 2+1=3 =) —— =) 4+1=5

遞迴 =) 1+1 =) 1+1+1 =) 1+1+1+1 =) 1+1+1+1+1

5 <= 1+4 <= 1+1+3 <= 1+1+1+2

ex: 最大公因數 =) 展轉相除法

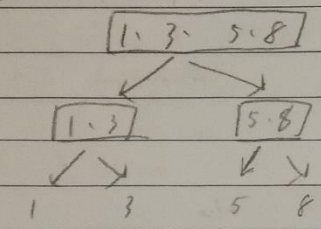
if x>y ; $x_1=9$、$y_1=6$ =) $x_2=6$、$y_2=3$ =) $x_3=3$、$y_3=0$

y_1　$x_1\%y_1$　y_2　$x_2\%y_2$

回傳

Date: NO: Check

ex: 2分法 對半



因不同狀況可能只取一邊 or 取兩邊

or可能不是對半 → 例：取第K小值

3 7 1 5 6 4 2 ：3為基點

=> 1 5 6 4 3 1 2

3為正確排序、可判斷兩邊的大小來

尋找第N小/大值

經典 ex: 河內塔

最少移動解 = 2^n-1 n=盤子數

遞迴解：有5個盤子 => 求4個解 => 求3個解

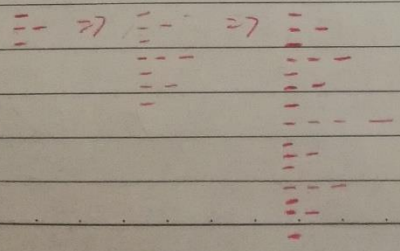
=> ... 求1個解

改變相對的起點和終點來達到目地

ex: 二元遞迴

將問題二元化，以呼叫兩次的方式完成遞迴

例：做一把尺

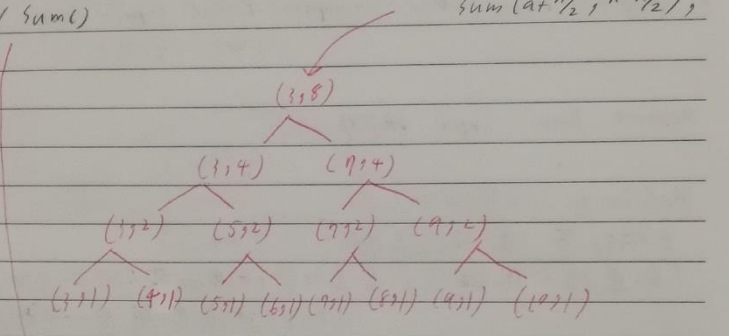


前提 a=3 、 b=10

Date: NO: 解：3+4+5+...+10 二元 Check

```
// Liner recursion
int Sum (int a, int b) {
  // assume a<b
  if (a==b)
    return a;
  else
    return a+Sum(a+1,b);
} // Sum()
```

```
// Binary recursion
int Sum (int a, int n) {
  // assume n=b-a+1
  if (n==1)
    return a;
  else
    return Sum(a,n/2) +
           Sum(a+n/2, n-n/2);
```



(3,10) → (4,10) → (5,10) → (6,10) → (7,10)
→ (8,10) → (9,10) → (10,10)

費式數列：1、2、3、5、8、13、21 ...

$n_1=1$ 、 $n_2=2$ 、 $n_3=n_1+n_2$ 、 $n_4=n_2+n_3$

$\Rightarrow n_k=n_{k-1}+n_{k-2}$, $k>2$

Date: NO: Check

Data abstraction (資料抽象化)

Object-Oriented (物件導向)

Encapsulation (封裝) Inheritance (繼承) Polymorphism (多型)
(封包、效率) (可用於類似屬性) (統合)

規定好 Purpose (目的)、Assumption (假設)、Input、Output

Abstract Data Types (ADT)

Modularity (模組化)
系統化管理 或處理較大的資料
(好處理、低錯誤)、需高內聚 (每個函式只做單一事件)
低耦合 (傳遞數據量一次越少越好)

Funtional abstraction (功能性的抽象化)
分區分塊，清楚描述且其為獨立運作
封裝並隱藏實作 (處理部分模塊時不干擾其程式運作)
(維護時好處理、好除錯)
使用者只需清楚該輸入什麼、不需呈現實作

描述：
ex: 貨物清單 (List)
基本：建構 List、解構 List、歸為空、計數、插入、刪除、
搜索
基本上這些就是夠

延申：清單反轉 (搜索 → 指入 → 刪除)
12345 → 51234 → 54123 → 54312 → 54321

Date: NO: Check

ex: 日期 (是否為假日)

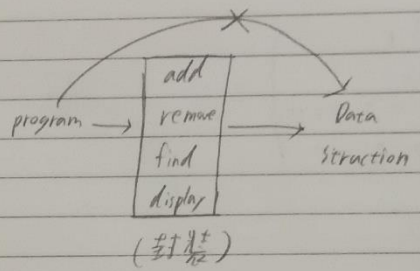
基本：元旦日期、是否今年、是否假日、下一天日期

注意：取名很重要(風格)

ex: 行事曆

基本：日期、時間、目的 (新增、取消)
是否有事、目的、類型
(可在因情況更細分)

實作：



多態：保留原本、在子程式加上特有的部分

多載：名字一樣但處理的數值不同、並動態
選擇需要的函式

ADT實例：

0	1	2	3	底層實作：陣列索引
2	5	3	4	
1	2	3	4	高階描述：第幾位置

Date: NO: Check

Pointer (指標)

(不需搬動資料，只需搬動指針)

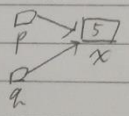
只需知道門牌，不需知道資料 (配給)

宣告時是還沒配給，不是 NULL

```
int *p; → 空
p = new int;        ) 完整宣告(無出差錯)
delete p; → 歸還
p = NULL; 遺忘      ) (保險)
```

```
ex: int *p, *q;
    int x = 5;
    p = &x;   => p = 5
    q = p     => q = 5
```

=>



動態(配置)陣列

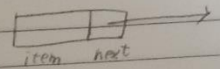
```
int arraySize = 50;
double * anArray = new double[ArraySize];
```

(空間不夠可擴大，但需搬資料)

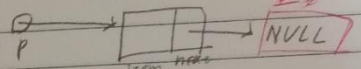
Date: NO: Check

指標實作

```
struct Node {
    int item;
    Node *next;
};
```




```
Node *p;
p = new Node;
p->next = NULL;
```



需畫圖(容易打結混)