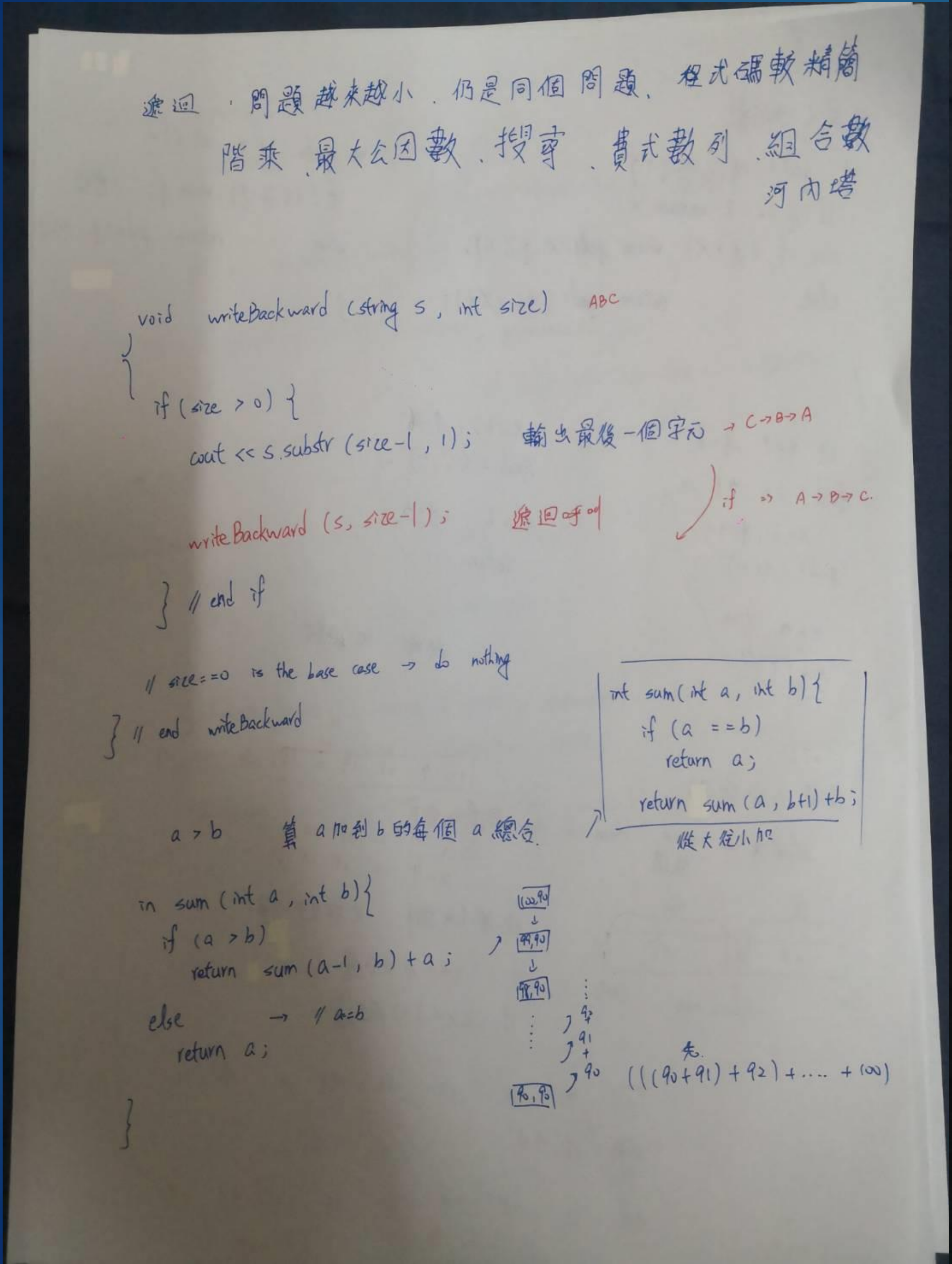


遞迴 ．問題越來越小．仍是同個問題．程式碼較精簡

階乘．最大公因數．搜尋．費式數列．組合數
河內塔

```
void writeBackward (string s, int size)   ABC
{
  if (size > 0) {
    cout << s.substr (size-1, 1);      輸出最後一個字元 → C→B→A
    writeBackward (s, size-1);         遞迴呼叫     if => A→B→C.
  } // end if
  // size==0 is the base case → do nothing
} // end writeBackward
```

a > b　算 a 加到 b 的每個 a 總合.

```
in sum (int a, int b){
  if (a > b)
    return sum (a-1, b) + a;
  else        → // a=b
    return a;
}
```

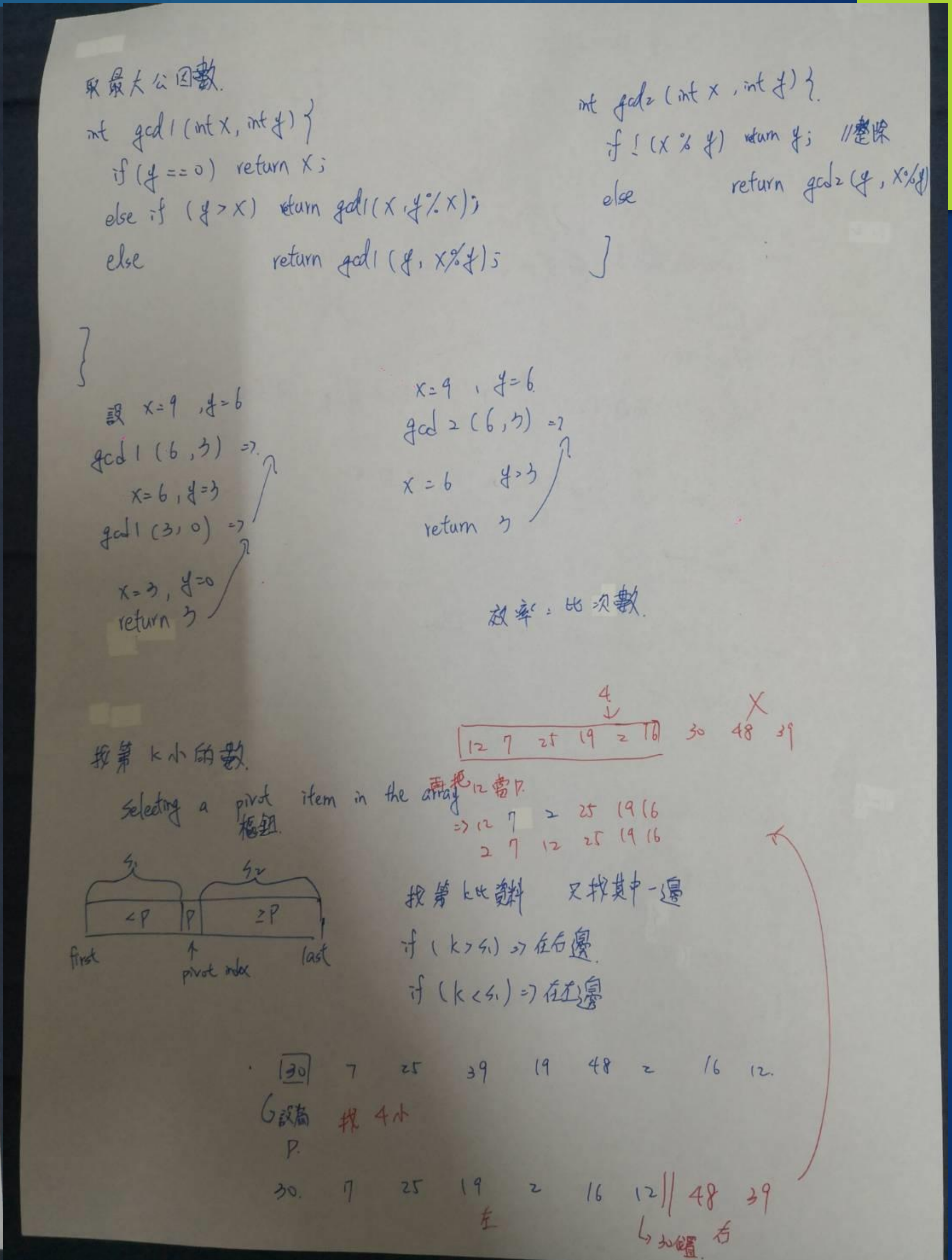
```
int sum(int a, int b){
  if (a ==b)
    return a;
  return sum (a, b+1) +b;
```

從大往小加

100,90
99,90
98,90
⋮
90,90

90 + 91 + 92 ⋮

先.
$(((90+91)+92)+\dots+100)$



取最大公因數.

```
int gcd1 (int x, int y) {
  if (y ==0) return x;
  else if (y > x) return gcd1(x, y%x);
  else            return gcd1 (y, x%y);
}
```

```
int gcd2 (int x , int y) {
  if !(x % y) return y;   //整除
  else        return gcd2 (y, x%y)
}
```

設 x=9 ,y=6
gcd1 (6 ,3) =>
x=6 , y=3
gcd1 (3, 0) =>
x=3, y=0
return 3

x=9 , y=6
gcd 2 (6, 3) =>
x = 6　y=3
return 3

效率：比次數.

找第 k 小的數.

Selecting a pivot item in the array
樞紐.

S1　S2
<P　P　≥P
first　pivot index　last

4
12 7 25 19 2 16 30 48 39
X

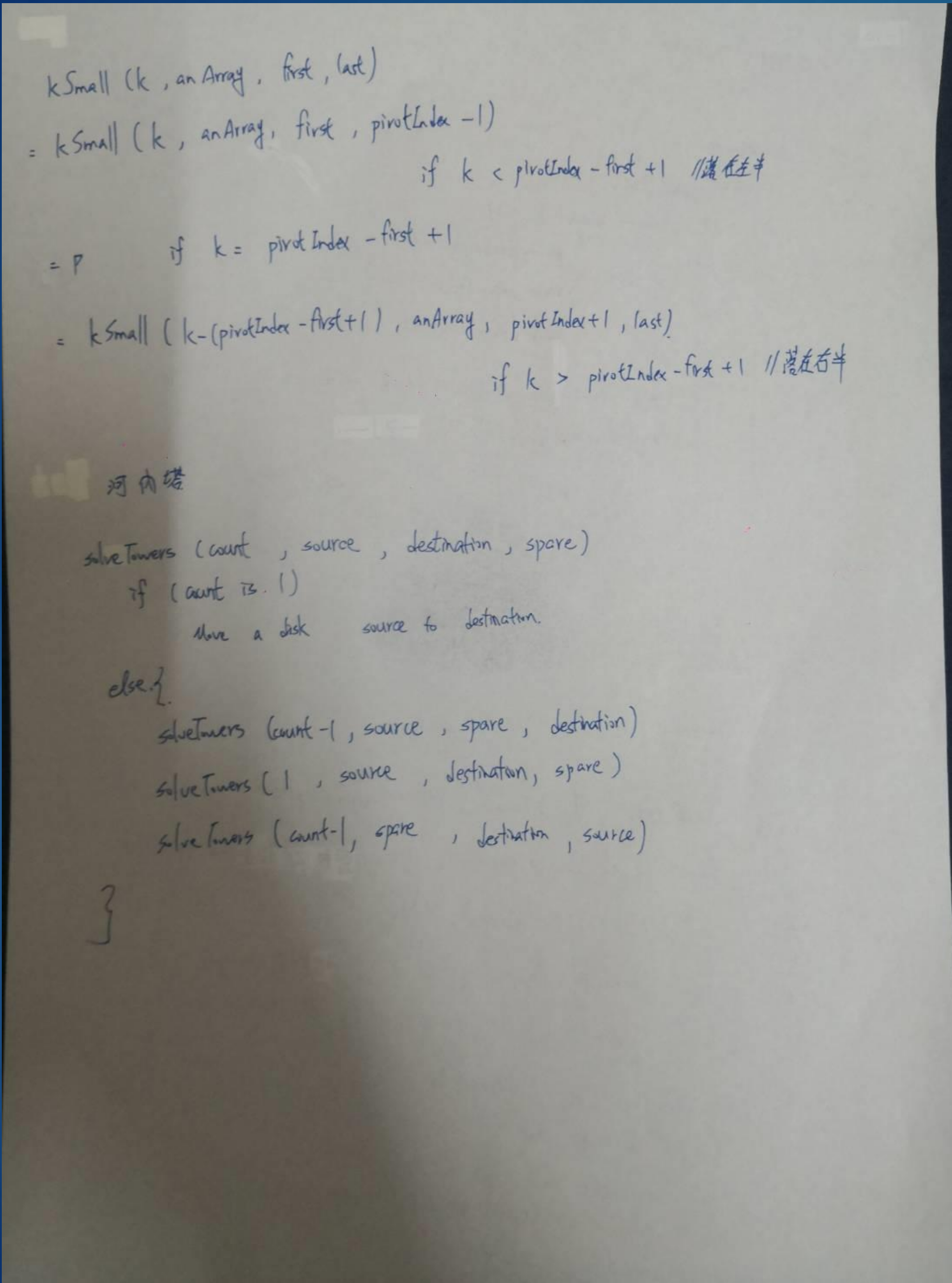
再把 12 當 P.
=> 12 7 2 25 19 16
2 7 12 25 19 16

找第 k 比資料　只找其中一邊

if (k > S1) => 在右邊.
if (k < S1) => 在左邊

30 7 25 39 19 48 2 16 12.
設為 P.　找 4 小

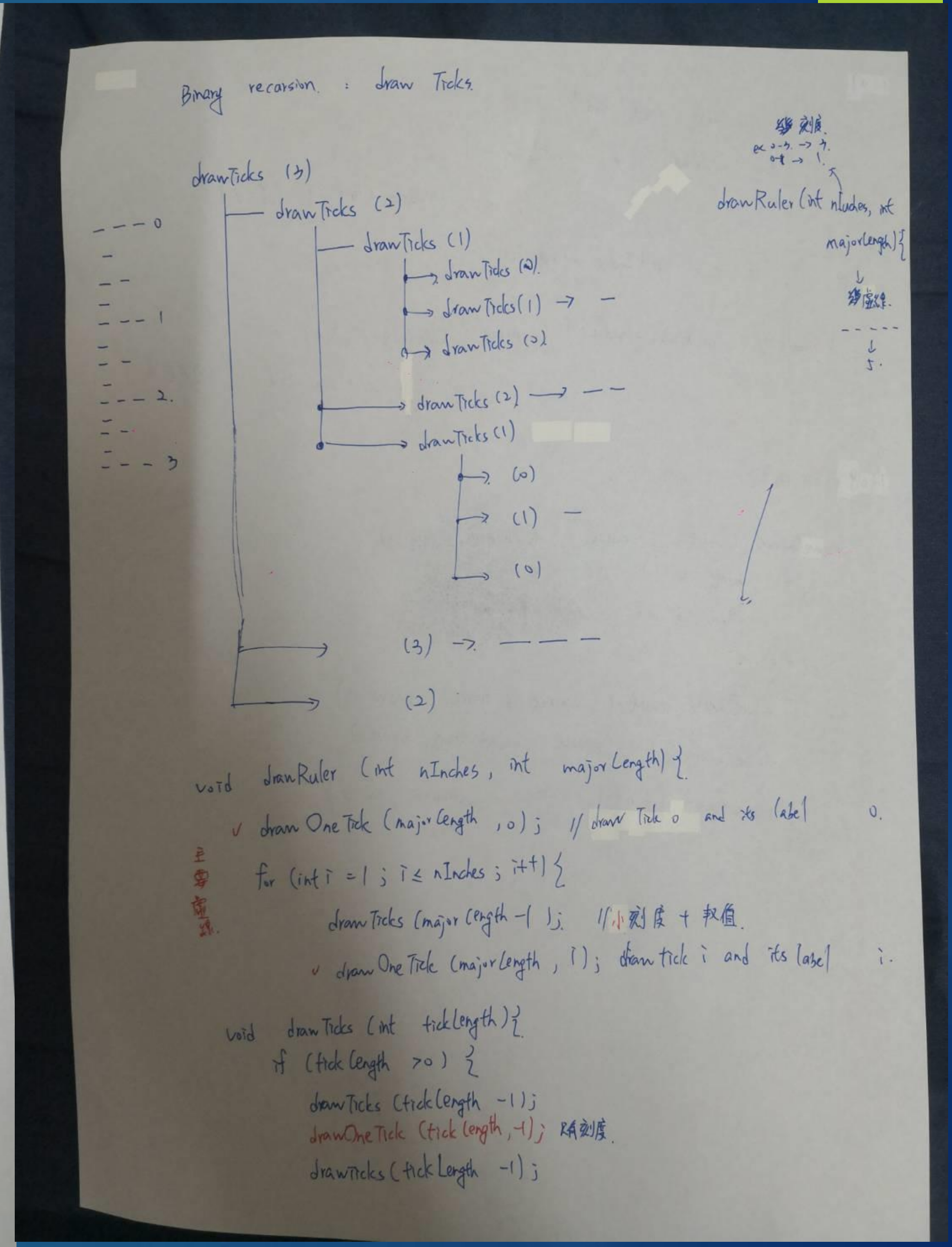
30. 7 25 19 2 16 12 ‖ 48 39
左　右
↳ 位置



```
kSmall (k , an Array , first , last)
= kSmall ( k , anArray , first , pivotIndex -1)
                        if k < pivotIndex - first +1   //落在左半

= P       if k = pivot Index - first +1

= kSmall ( k-(pivotIndex -first+1) , anArray , pivot Index+1 , last)
                        if k > pivotIndex - first +1   //落在右半
```

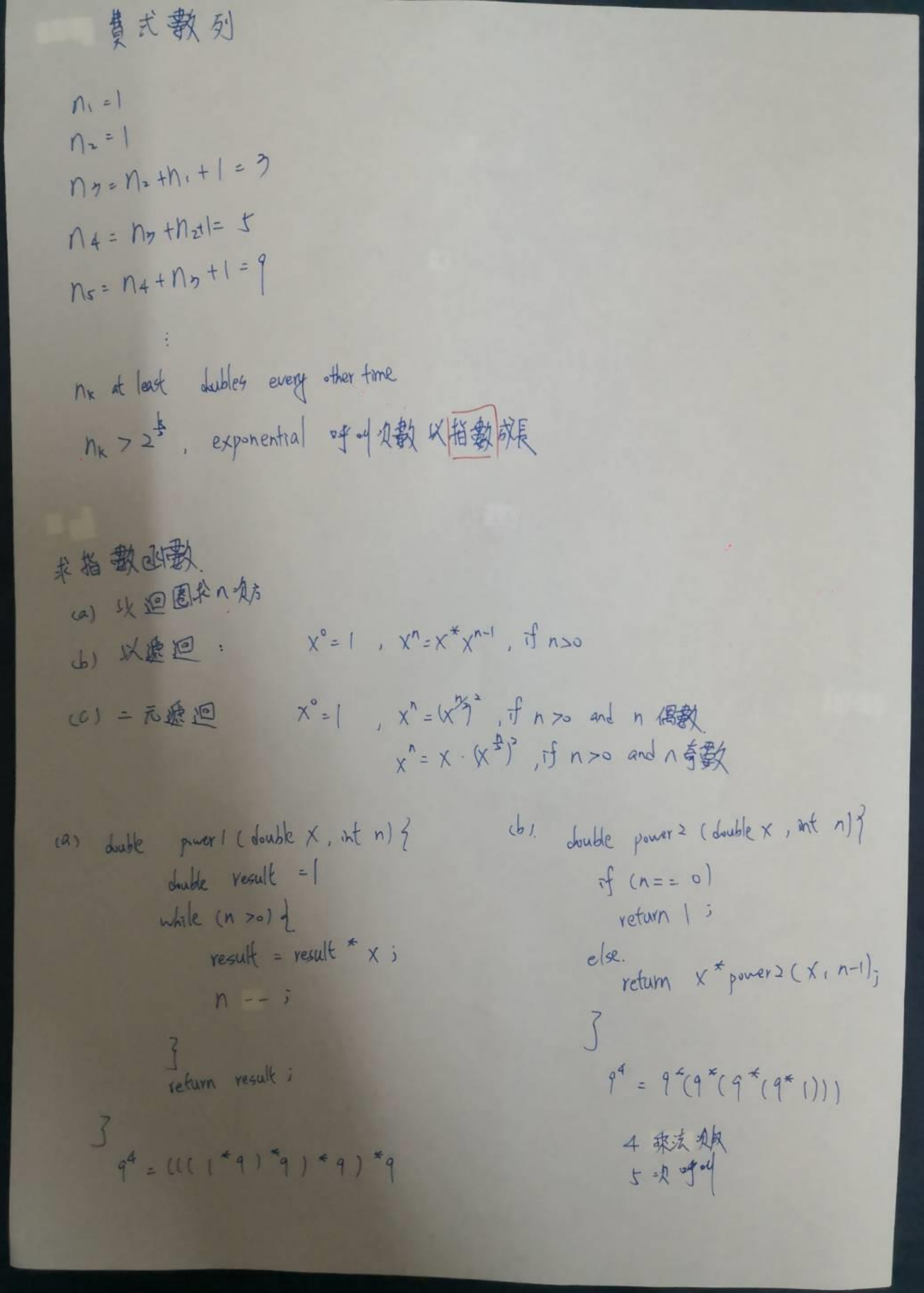
河内塔

```
solveTowers (count , source , destination , spare)
  if (count is 1)
      Move a disk source to destination.
  else {
      solveTowers (count -1 , source , spare , destination)
      solveTowers (1 , source , destination , spare )
      solveTowers (count-1, spare , destination , source)
  }
```



Binary recursion : draw Ticks.

```
void drawRuler (int nInches, int majorLength) {
  ✓ drawOneTick (majorLength , 0) ;  // draw Tick 0 and its label    0.
主要流程.
  for (int i = 1 ; i ≤ nInches ; i++) {
      drawTicks (majorLength -1 ); //小刻度 + 权值.
    ✓ drawOneTick (majorLength , i) ; draw tick i and its label    i.

void drawTicks (int tickLength) {
  if (tickLength >0 ) {
     drawTicks (tickLength -1) ;
     drawOneTick (tick length, -1) ; 所有刻度.
     drawTicks (tick Length -1) ;
```



費式數列

$n_1 = 1$

$n_2 = 1$

$n_3 = n_2 + n_1 + 1 = 3$

$n_4 = n_3 + n_2 + 1 = 5$

$n_5 = n_4 + n_3 + 1 = 9$

⋮

n_k at least doubles every other time

$n_k > 2^{\frac{k}{2}}$, exponential 呼叫次數以指數成長

求指數函數

(a) 以迴圈求n次方

(b) 以遞迴 : $x^0 = 1$, $x^n = x * x^{n-1}$, if $n > 0$

(c) 二元遞迴 $x^0 = 1$, $x^n = (x^{\frac{n}{2}})^2$, if $n > 0$ and n 偶數

$x^n = x \cdot (x^{\frac{n}{2}})^2$, if $n > 0$ and n 奇數

```
(a) double power1 (double x, int n) {
        double result = 1
        while (n > 0) {
            result = result * x ;
            n -- ;
        }
        return result ;
    }
```

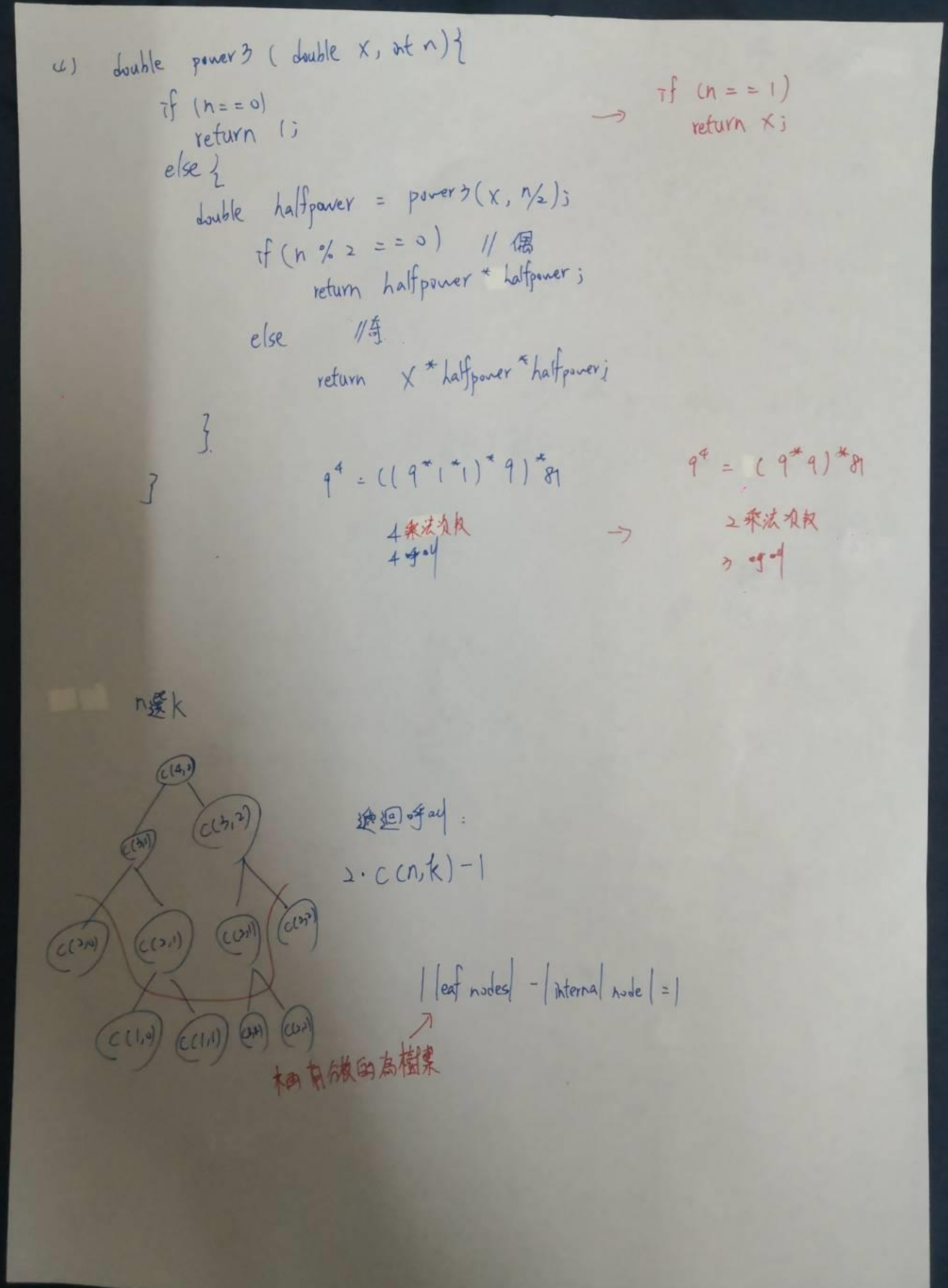
$9^4 = (((1 * 9) * 9) * 9) * 9$

```
(b) double power2 (double x, int n) {
        if (n == 0)
            return 1 ;
        else
            return x * power2 (x, n-1);
    }
```

$9^4 = 9*(9*(9*(9*1)))$

4 乘法次數

5 次呼叫



```
(c) double power3 ( double x, int n) {
        if (n == 0)                          →  if (n == 1)
            return 1;                               return x;
        else {
            double halfpower = power3(x, n/2);
                if (n % 2 == 0)   // 偶
                    return halfpower * halfpower;
                else      // 奇
                    return x * halfpower * halfpower;
        }
    }
```

$9^4 = ((9 * 1 * 1) * 9) * 81$ → $9^4 = (9 * 9) * 81$

4 乘法次數 → 2 乘法次數

4 呼叫 → 3 呼叫

n選k

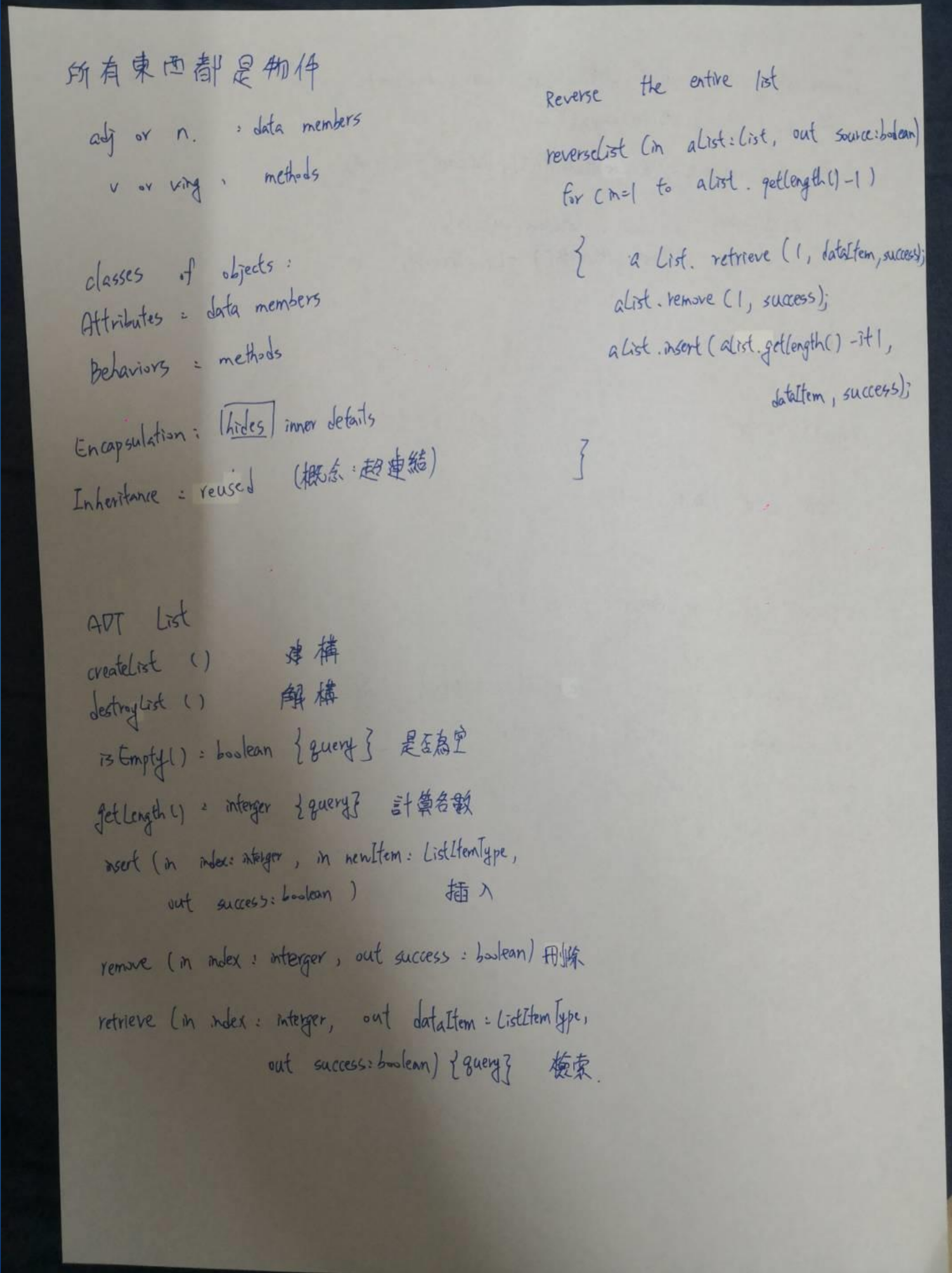
C(4,2)
C(3,1)
C(3,2)
C(2,0)
C(2,1)
C(2,1)
C(2,2)
C(1,0)
C(1,1)
C(1,0)
C(1,1)

遞迴呼叫 :

$2 \cdot C(n,k) - 1$

| leaf nodes | - | internal node | = 1

橘色圈的為樹葉



所有東西都是物件

adj or n. : data members

v or ving , methods

classes of objects :

Attributes = data members

Behaviors = methods

Encapsulation : hides inner details

Inheritance : reused (概念：超連結)

ADT List

createList () 建構

destroyList () 解構

isEmpty() : boolean {query} 是否為空

getLength() : interger {query} 計算各數

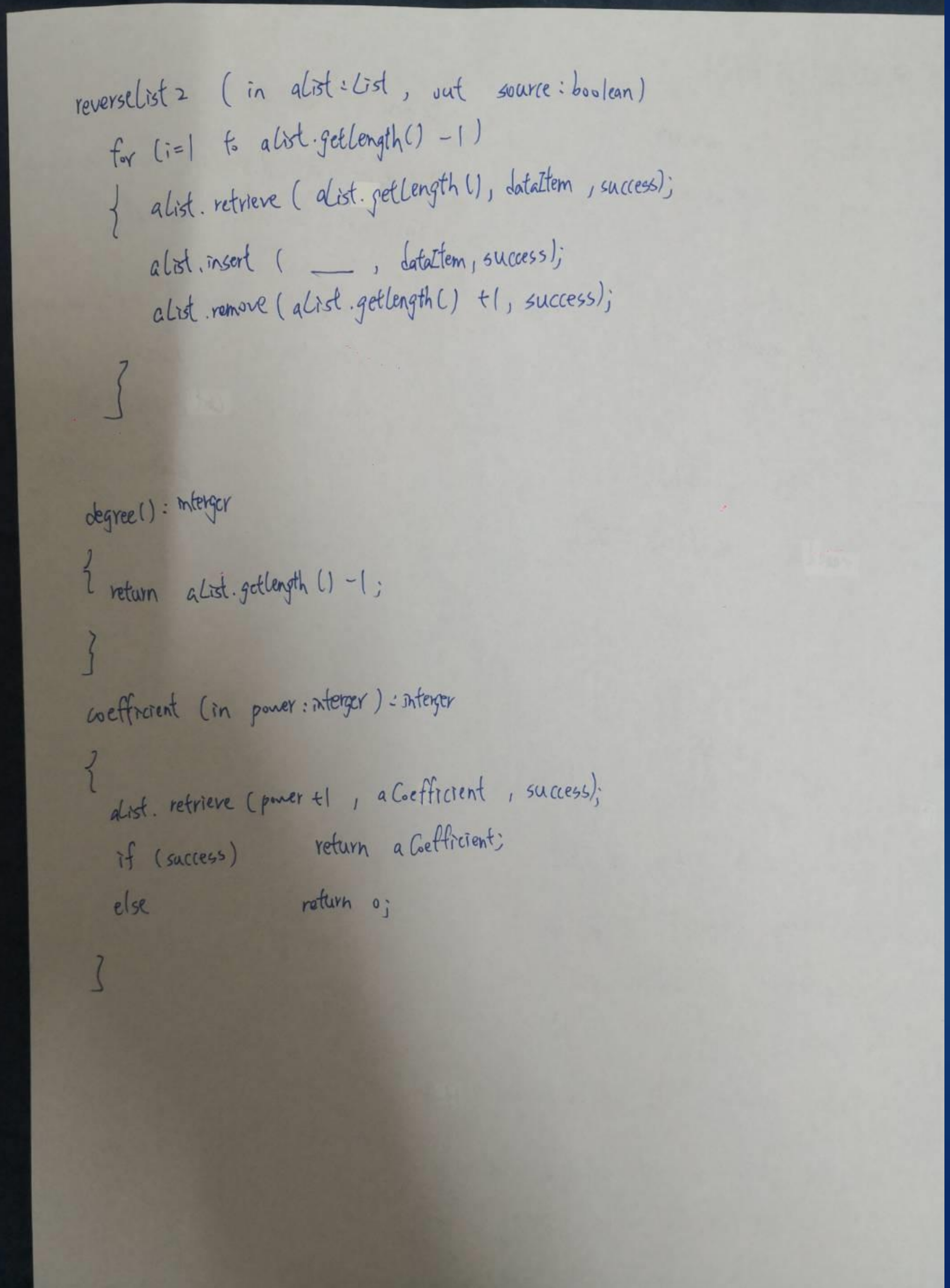
insert (in index: interger , in newItem : ListItemType,
out success: boolean) 插入

remove (in index : interger , out success : boolean) 刪除

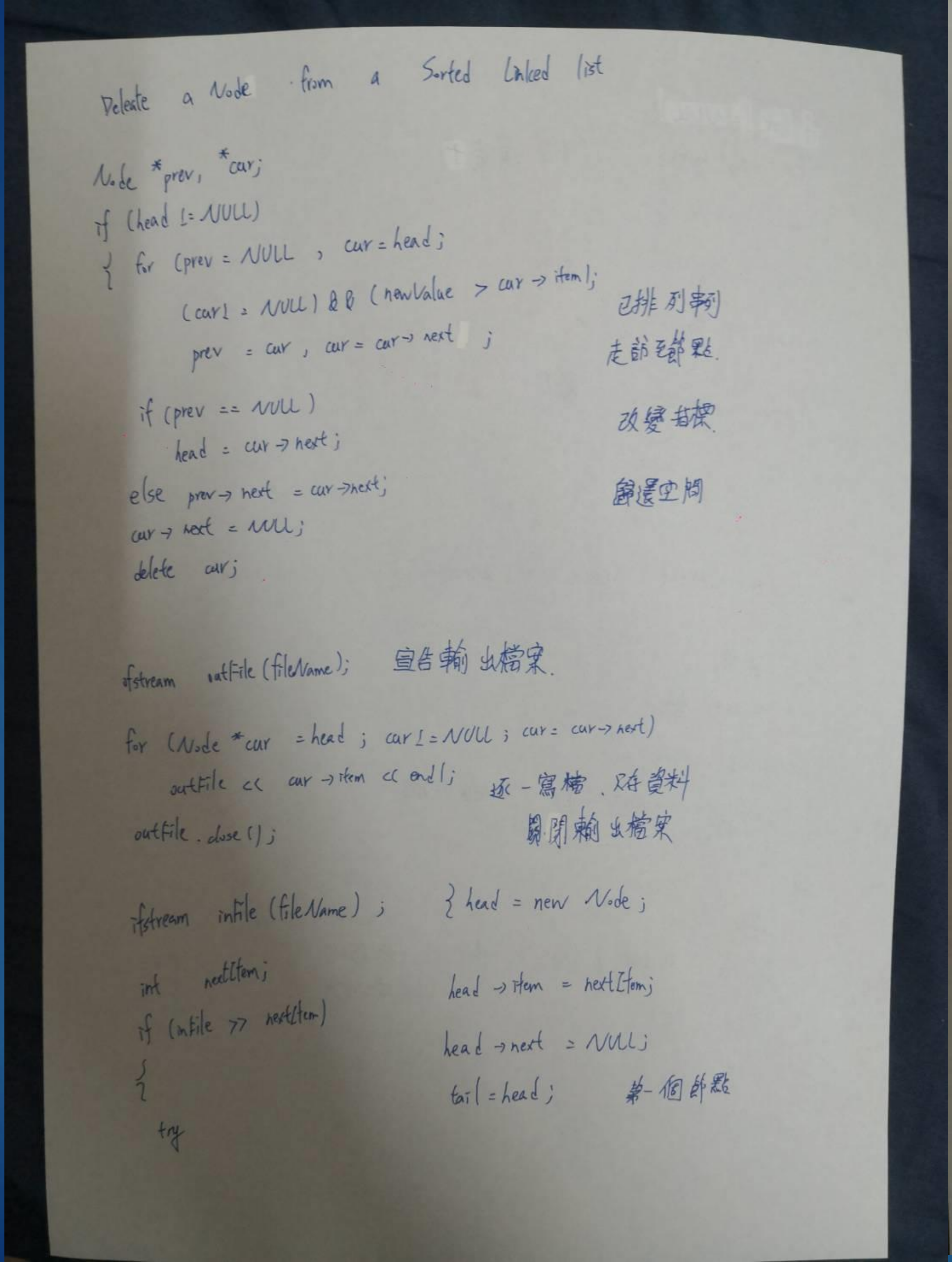
retrieve (in index : interger, out dataItem : ListItemType,
out success: boolean) {query} 檢索.

Reverse the entire list

```
reverselist (in aList: List, out source:boolean)
 for (m=1 to alist . getlength() -1 )
  {  a List. retrieve (1, dataItem, success);
   aList . remove (1, success);
   aList . insert ( aList. getlength() -i+1,
                        dataItem , success);

  }
```

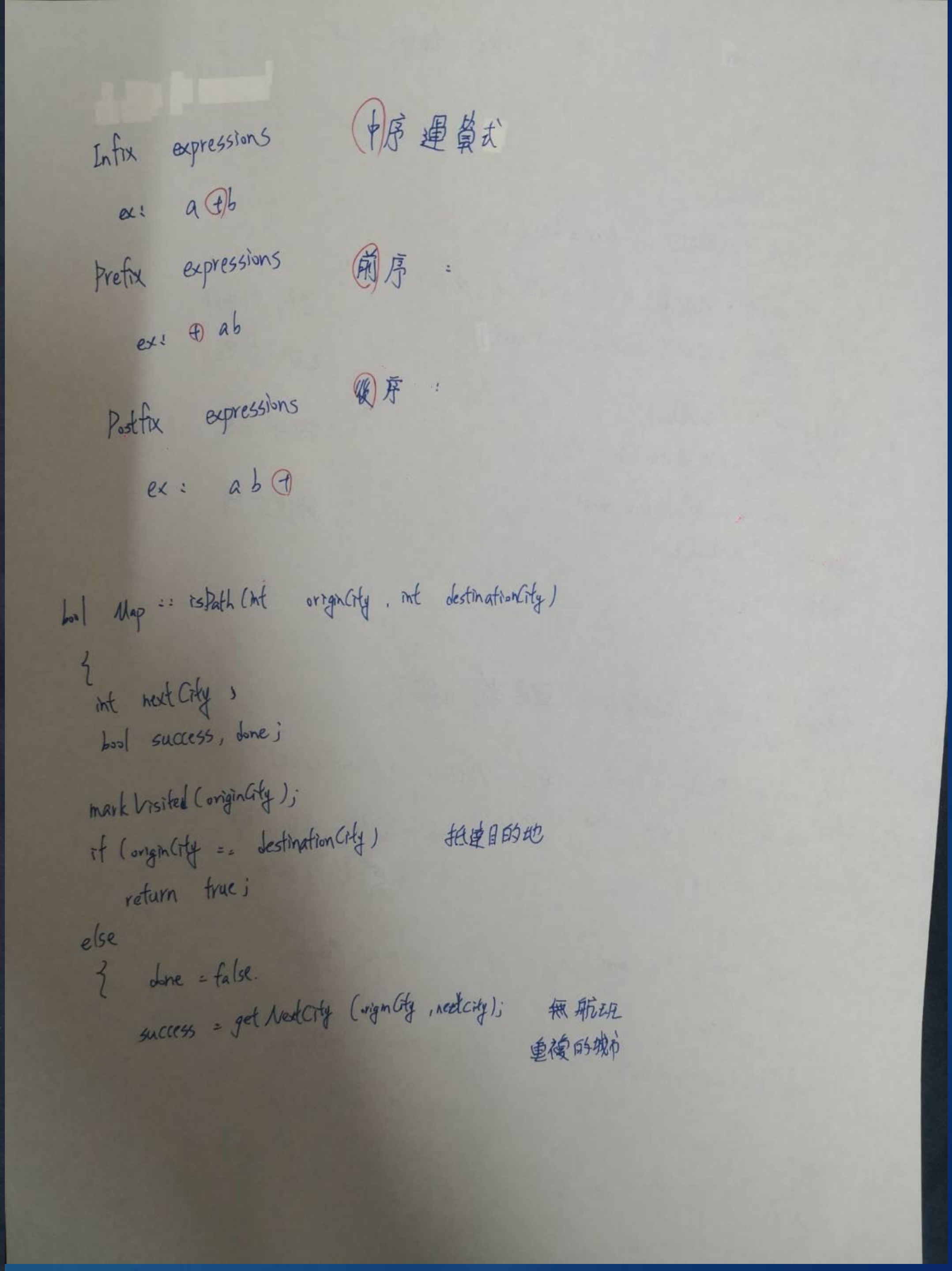


```
reverseList 2 ( in aList : List , out source : boolean)
  for (i=1 to aList.getLength() -1 )
  {  aList. retrieve ( aList. getLength (), dataItem , success);
     aList.insert ( ___ , dataItem, success);
     aList .remove ( aList .getlength() +1, success);

  }


degree() : interger
{ return aList.getlength () -1 ;
}

coeffrcient (in power : interger ) : interger
{
  aList. retrieve (power +1 , aCoefficient , success);
  if (success)       return aCoefficient;
  else               return 0;

}
```



Delete a Node from a Sorted Linked list

```
Node *prev, *cur;
if (head != NULL)
{ for (prev = NULL , cur = head;
      (cur != NULL) && (newValue > cur -> item);           已排列串列
      prev = cur , cur = cur -> next ;                      走訪至節點

  if (prev == NULL)                                         改變指標
     head = cur -> next;
  else prev -> next = cur -> next;                          歸還空間
  cur -> next = NULL;
  delete cur;


ofstream outFile (fileName);   宣告輸出檔案.

for (Node *cur = head ; cur != NULL ; cur = cur -> next)
   outFile << cur -> item << endl;        逐一寫檔 . 存資料
outFile . close ();                          關閉輸出檔案

ifstream inFile (fileName) ;        { head = new Node ;

int    nextItem;                      head -> item = nextItem;
if (inFile >> nextItem)
                                      head -> next = NULL;
{
                                      tail = head;      第一個節點
  try
```



Infix expressions 中序運算式

ex: a + b

Prefix expressions 前序 :

ex: + ab

Postfix expressions 後序 :

ex: ab +

```
bool Map :: isPath (int originCity , int destinationCity)
 {
  int nextCity ;
  bool success, done;

  markVisited (originCity);
  if (originCity == destinationCity)          抵達目的地
     return true;
 else
  {   done = false.
     success = getNextCity (originCity , nextcity);     無航班
                                                        重複的城市
```