Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Made by 培訓團隊群

Week 2
Basic Programing

STL 和 Coding 小知識

Outline

- Coding 小知識
- STL
  - Vector
  - String
  - STL 可以在型態中宣告 STL
- sort
- 題目賞析
  - CodeForces 1130 B
  - CodeForces 1137 A

Coding 小知識

- cin and cout
  - 輸入加速方式
  - 如何讀取包含空白的資訊
- long long int
- 陣列
  - memset
  - sizeof

cin and cout

- 有很好的型態支援度
- 速度比 scanf 和 printf 慢？
  - 原因是出在要配合 scanf 和 printf ，如果我們把它取消來看看會發生什麼事情。
  - ios::sync_with_stdio(0)
  - cin.tie(0)

a002. sort speed up	AC (0.9s, 19.4MB)	CPP	2019-02-25 17:37
a002. sort speed up	AC (1.7s, 19.4MB)	CPP	2019-02-25 17:18

cin and scanf and getline

- 讀取到空白停止(終止字元是空白)
  - 如果遇到需要讀取空白的情況，用 getline 解決(可設定終止字元)
  - 使用頻率少，但是必須知道它的存在。

long long

- 有比較大的數值處理能力
  - 使用時機：處理 int 範圍的測試資料時，如果過程中會有 加法 與 乘法 的操作
  - 大約是 -4e18 ~ 4e18

- PS：4e18 這個表示法相當於科學記號中的 4×10^{18}
- PS：如果連 long long 都沒辦法處理，那就是 大數 這個又更麻煩了。

陣列

- 初始化：memset
- 靜態宣告 與之後介紹的 vector<any_type> 比較
  - 建議陣列宣告根據題目給定的“Max N”宣告大小
  - 如果是想要根據題目給定的 N 宣告大小的話，建議使用動態的 vector<any_type>

STL

- vector<any_type>
  - 動態特性
- Cplusplus.com 說明文件簡介
- string
- STL 可以套在 STL 裡面
  - 然而 STLSTL 又可以套在 STL 裡面
    - 然而 STLSTLSTL 又可以套在 STL 裡面
      - 然而 STLSTLSTLST L又可以套在 STL 裡面
        - 然而 STLSTLSTLSTLSTL 又可以套在 STL 裡面

vector<any_type>

- vector 擁有很多方便的自帶函數可以使用，可以加快不少開發速度
  - vector::operator[]
  - vector::size()
  - vector::resize()
  - vector::assign()
  - vector::push_back()
  - 說明文件： www.cplusplus.com

C plus plus 說明文件簡介

- 以 `vector::push_back()`為例
  - 如果覺得文件量太大，可以先專注在定義、範例、複雜度，三個地方。

C plus plus 說明文件簡介

public member function

std::vector::push_back

<vector>

C++98 C++11

```
void push_back (const value_type& val);
```

Add element at the end

Adds a new element at the end of the vector, after its current last element. The content of *val* is copied (or moved) to the new element.

This effectively increases the container size by one, which causes an automatic reallocation of the allocated storage space if -and only if- the new vector size surpasses the current vector capacity.

Parameters

`val`

Value to be copied (or moved) to the new element.
Member type `value_type` is the type of the elements in the container, defined in vector as an alias of its first template parameter (`T`).

C plus plus 說明文件簡介

Example

```cpp
// vector::push_back
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  int myint;

  std::cout << "Please enter some integers (enter 0 to end):\n";

  do {
    std::cin >> myint;
    myvector.push_back (myint);
  } while (myint);

  std::cout << "myvector stores " << int(myvector.size()) << " numbers.\n";

  return 0;
}
```

The example uses `push_back` to add a new element to the vector each time a new integer is read.

C plus plus 說明文件簡介

Complexity

Constant (amortized time, reallocation may happen).

If a reallocation happens, the reallocation is itself up to linear in the entire size.

string

- vector<any_type> 擁有的自帶函數 string 也有，如：
  - string::push_back()
  - string::assign()

- 還有很多針對字串的自帶函數，如：
  - string::substr()
  - string::operator+=

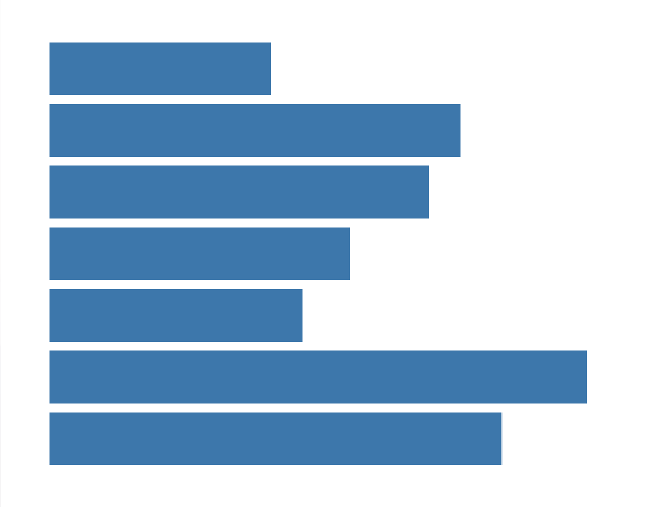
- 也可以把 string 轉型態成 char[] 的字串型態：
  - string::c_str()

string 字典序

- 簡單理解
  - 設想一本英語字典里的單詞，哪個在前哪個在後？

STL 可以套在 STL 裡面

- vector<any_type>，的範例：
  - vector<int>：整數vector
  - vector<long long int>：長整數vector
  - vector< vector<int> >：整數vector的vector



自學清單

- pair
  - first, second
  - sort
- set
  - insert
  - set::iterator 的 operator++ 的時間複雜度
  - 把整個 set 依序顯示
- map
  - map::operator[] 的時間複雜度
  - iterator 的 operator++ 的時間複雜度
  - 把整個 map 依序顯示

sort

- 將一堆元素由"給定規則"排成一順序
- 對於*整數* 預設是定義 *是否小於* 的規則
- 對於*字串* 預設是定義 *是否字典序小於* 的規則

sort

- 5, 6, 9, 8, 2 這五個元素由小到大排為 2, 5, 6, 8, 9
- a, bc, ay, aa 這四個元素由字典順序排為 a, aa, ay, bc

回顧字典序

- 簡單理解
  - 設想一本英語字典里的單詞，哪個在前哪個在後？

- `bool operator<  (cont string& lhs, const string& rhs);`

string 字典序

Example

```cpp
// string comparisons
#include <iostream>
#include <vector>

int main ()
{
  std::string foo = "alpha";
  std::string bar = "beta";

  if (foo==bar) std::cout << "foo and bar are equal\n";
  if (foo!=bar) std::cout << "foo and bar are not equal\n";
  if (foo< bar) std::cout << "foo is less than bar\n";
  if (foo> bar) std::cout << "foo is greater than bar\n";
  if (foo<=bar) std::cout << "foo is less than or equal to bar\n";
  if (foo>=bar) std::cout << "foo is greater than or equal to bar\n";

  return 0;
}
```

Output:

```
foo and bar are not equal
foo is less than bar
foo is less than or equal to bar
```

vector sort 使用練習

- 請各位打開 judge.cp.ccns.io
  - 特別感謝 ccns 的網管大大鼎力相助
- 練習題目 a002
- 練習&下課
- 15min鐘後繼續上課。

```cpp
#include <bits/stdc++.h>

vector<int> a;
int n;

int main() {
    cin >> n;
    a.resize(n);
    for(int i = 0; i < n; i++) cin >> a[i];
    sort(a.begin(), a.end());
    for(int i = 0; i < n; i++) cout << a[i] << " ";
    cout << endl;
    return 0;
}

```

sort

- 將一堆元素由"給定規則"排成一順序
- 對於 *vector<int>* 這種型態呢？
- 對於 *自定義struct* 的型態呢？
- 對於 *any_type* 呢？

自定義 sort

- 參考 cplusplus 網站上 sort 的定義
  - 可以在第三個欄位放入自定義小於

function template

std::sort

default (1)	`template <class RandomAccessIterator> void sort (RandomAccessIterator first, RandomAccessIterator last);`
custom (2)	`template <class RandomAccessIterator, class Compare> void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);`

自定義 sort

- 那我們是不是可以對 any_type 定義小於
- 那我們是不是可以對 vector<int> 定義小於

function template

std::sort

default (1)
```
template <class RandomAccessIterator>
  void sort (RandomAccessIterator first, RandomAccessIterator last);
```

custom (2)
```
template <class RandomAccessIterator, class Compare>
  void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);
```

題目賞析 – CodeForces 1130 B

- 現在有兩個人，小藍和小紅他們在位置1。
- 現在有一數列中有 2 個 1、2 個 2 ...、2 個 N，亂序
- 他們要各自依序拜訪 1 ~ N
- 且被拜訪過的位置不能被另一個人拜訪
- 請問兩人最小移動步數和為？

題目賞析 – CodeForces 1130 B

- Input

3
1 1 2 2 3 3

- Output

9

題目賞析 – CodeForces 1130 B

- Input

3

1 1 2 2 3 3

- 累計移動步數：0
- Output

9

題目賞析 – CodeForces 1130 B

- Input

3

1 1 2 2 3 3

- 累計移動步數：2
- Output

9

題目賞析 – CodeForces 1130 B

- Input

3

1 1 2 2 3 3

- 累計移動步數：4
- Output

9

題目賞析 – CodeForces 1130 B

- Input

3

1 1 2 2 3 3

- 累計移動步數：4+1
- Output

9

題目賞析 – CodeForces 1130 B

- Input

3

1 1 2 2 3 3

- 累計移動步數：4+3
- Output

9

題目賞析 – CodeForces 1130 B

- Input

3

1 1 2 2 3 3

- 累計移動步數：4+5
- Output

9

題目賞析 – CodeForces 1130 B

- Input

4

4 1 3 3 1 2 2 4

- Output

23

位置	拜訪次序
1	4
2	1
3	3
4	3
5	1
6	2
7	2
8	4

題目賞析 – CodeForces 1130 B

- 分析：
- 為了避免靠位置小的人走到下個拜訪中位置大的，造成步數的浪費
- 小紅都去拜訪 i 中位置小的那家
- 小藍都去拜訪 i 中位置小的那家

4 1 3 3 1 2 2 4

- 我想很多人都可以想到這裡。

題目賞析 – CodeForces 1130 B

- 怎麼實現這個想法？

題目賞析 – CodeForces 1130 B

- 每個點有位置跟拜訪次序兩個特徵

位置	拜訪次序
1	4
2	1
3	3
4	3
5	1
6	2
7	2
8	4

題目賞析 – CodeForces 1130 B

- 每個點有位置跟拜訪次序兩個特徵
- 如果我們按照 *拜訪次序* 排序

位置	拜訪次序
2	1
5	1
6	2
7	2
3	3
4	3
1	4
8	4

題目賞析 – CodeForces 1130 B

- 第奇數個row是拜訪順序中位置小的
  - 小紅
- 第偶數個row是拜訪順序中位置大的
  - 小藍

位置	拜訪次序
2	1
5	1
6	2
7	2
3	3
4	3
1	4
8	4

題目賞析 – CodeForces 1130 B

- 實現這個想法？

位置	拜訪次序
2	1
5	1
6	2
7	2
3	3
4	3
1	4
8	4

感受一下 excel 操作 – 自定義排序

位置	拜訪次序
1	4
2	1
3	3
4	3
5	1
6	2
7	2
8	4

位置	拜訪次序
2	1
5	1
6	2
7	2
3	3
4	3
1	4
8	4

題解說明

- 等等會示範用 vector 套 vector<int> 來存取資料
  - 因為 vector< vector<int> > 的適用範圍比較廣，方便同學適應更多題目
- 但是這題可以使用 vector 套 pair 來存取資料比較方便
  - 不用自定義 sort，因為 pair 有 *是否小於* 的定義，熟悉的同學可以更快速的實作算法

vector 套 vector<int> - 示範

- 宣告

```
vector< vector<int> > a;
```

vector 套 vector<int> - 示範

- 初始化

```
int main() {
: cin >> n;
: a.assign(2*n, vector<int>(2, 0));
:
```

- Hint 若要動態宣告大小，一定要先獲得 題目規定 的數值。

vector 套 vector<int> - 示範

- 完整讀取資料示範

```cpp
int main() {
    cin >> n;
    a.assign(2*n, vector<int>(2, 0));

    for(int i = 0; i < 2*n; i++) {
        int thing, addr;
        addr = i+1;
        cin >> thing;
        a[i][0] = addr;
        a[i][1] = thing;
    }
```

題目賞析 – CodeForces 1130 B

- 開始模擬走路吧！
  - Hint 走路的步數會大於 int 範圍
- 以上這題全部所需要使用到的基本操作就教給各位了！
- 有了這些技巧之後
  - 就可以試試看思考題目
  - 更方便的快速實作腦袋中的思路

題目賞析 – CodeForces 1107 A

- 現在有 Q 筆問題
- 每筆問題中有一個 N 代表之後的數字 S 是幾位數
- 現在想要把數字 S 拆成兩個數字以上，並且由小到大排序，請問可能辦到嗎？
- 可以的話輸出可能的拆法

題目賞析 – CodeForces 1107 A

- Input

2

6

654321

2

33

- Output

YES

3

6 54 321

NO

題目賞析 – CodeForces 1107 A

- 貼心小提示：
  - 可以使用 string 儲存
  - 可以使用 str.substr() 輸出

練習&下課時間

Take a break!

Outline

1. 演算法的效率
2. 設計演算法的思考方法

演算法的效率

Competitive Programming

Made by 培訓團隊群

Big O

2 倍、3 倍、甚至 10 倍的常數倍優化不是競賽時考慮的要點。

我們所設計的演算法必須根據輸入規模 N 而定。

Big O

Big O 表示法

$f(N) = O(g(N)) \Leftrightarrow \exists M, c > 0.\ \forall N > M.\ |f(N)| \leq c \cdot |g(N)|$

意思是說在 N 足夠大的時候，已經存在正數 c 使得 $c \cdot |g(N)|$ 大於等於 $|f(N)|$

Big O

例如估計的時間函數: $f(x)=x^2+x+1$

在 x 很大的時候，
主要影響整個函數值的大小是平方項

這時我們可以說 $f(x) = O(x^2)$

Big O

設輸入規模為 N，常見的複雜度有：

$O(1) \leq O(\log N) \leq O(N) \leq O(N\log N)$
$\leq O(N^k) \leq O(k^N) \leq O(N!) \leq O(N^N)$

其中 k 為常數 (不隨輸入規模成長)

競賽規範

記憶體**空間**的規範各競賽都不相同
通常得考慮:
- 遞迴深度
- 使用的變數多寡
- 程式碼長度

競賽規範

而競賽都以秒為單位去做**時間**限制
- 例如 1 秒、 3 秒、10 秒

合理的複雜度

通常會直接考慮資料的規模與計算出來的複雜度

有個傳統(?)的限制： 10^7

合理的複雜度

假設題目:

規模為 N

而你:

設計出的演算法複雜度為 $O(N^2 \log N)$

合理的複雜度

$x = N^2 \log N$ 得落在 $x \leq 10^7$ 左右
這樣的複雜度才不容易超時

- 也就是說如果 $N = 10^5$
- 那就得重新設計演算法
- 因為此時 $x = 10^{10} * \log(10^5)$ 超大

常見思考方法

演算法的設計思維

-枚舉
-動態規劃
-分治法
-貪心法

最大連續和問題

- 考慮整數數列: a(1), a(2), ..., a(N)
- 讓 a(L), a(L+1), ..., a(R) 盡量大
- 其中 1 <= L <= R <= N

- 例如 -4, 2, 3, -1, 0, 4, -5, 6, -7
  的最大連續和為 9
- [2, 3, -1, 0, 4, -5, 6]

演算法的設計思維

-枚舉
-動態規劃
-分治法
-貪心法

枚舉

所謂枚舉，
就是數出部份給定的集合中元素。

應用在問題中
將每個 L 與 R 配對舉出來
接著`for(int k = L; k <= R; k++) sum += A[k];`
就能找出最大的 sum

枚舉: 最大連續和問題

```
int best = A[1];

for (int L = 1; L <= N; L++) {
  for (int R = L; R <= N; R++) {
    int sum = 0;
    for (int k = L; k <= R; k++) sum += A[k];
    best = max(best, sum);
  }
}
```

其時間複雜度為 $O(N^3)$

演算法的設計思維

-枚舉
-動態規劃
-分治法
-貪心法

動態規劃: 最大連續和問題

```
for (int L = 1; L <= N; L++) {
   sum[L][L-1] = 0;
   for (int R = L; R <= N; R++) {
      sum[L][R] = sum[L][R-1] + A[R];
      best = max(best, sum[L][R]);
   }
}
```

時間複雜度為 $O(N^2)$

動態規劃

```
sum[L][L-1] = 0;
for (int R = L; R <= N; R++)
   sum[L][R] = sum[L][R-1] + A[R];
```

從邊界遞推地紀錄所有問題的解，且一個項用到前一項的最佳結果

這就是在計算前綴和

動態規劃

好處是將會重複使用到的解都保存下來了，就能省下不少時間

不用像枚舉一樣重新計算

```
for(int k = L; k <= R; k++) sum += A[k];
```

演算法的設計思維

-枚舉
-動態規劃
-分治法
-貪心法

分治法

- 分治 (divide & conquer) 簡稱 D&C
- 將一個大的問題
- 分成幾個**互相獨立**的子問題
- 然後再將子問題分成子子問題
- 一直重複分割的動作直到最小問題 (邊界)
- 接著讓子問題合併求出父問題。

分治法: 最大連續和問題

- 將數列切一半 (分割)

- 左半的最大連續和為何 (子問題)
- 右半的最大連續和為何 (子問題)
- 包含”切開的分水嶺”的最大連續和 (子問題☆)

- 選出三者中最大值，就是整個數列的解 (合併)

分治法: 原大小的問題

P [a(1), a(2), a(3), a(4), a(5)]

假設 N = 5

分治法: 分割問題

[a(1), a(2), a(3), a(4), a(5)]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

P [a(1), a(2)]

P [a(3), a(4), a(5)]

分治法: 分割問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

P [a(1)]

P [a(2)]

分治法: 最小子問題(邊界)

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

[a(1)]

P [a(2)]

Return a(1)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

P [a(2)]

分治法: 最小子問題(邊界)

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

[a(2)]

Return a(2)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

P [..., a(1), a(2), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

maxSum [..., a(1), a(2), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

L=P [a(1)]

R=P [a(2)]

M=maxSum [..., a(1), a(2), ...]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

[a(1), a(2)]

P [a(3), a(4), a(5)]

Return max(L, M, R)

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

P [a(3), a(4), a(5)]

分治法: 分割問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

P [a(3)]

P [a(4), a(5)]

分治法: 最小子問題 (邊界)

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

[a(3)]

P [a(4), a(5)]

Return a(3)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

P [a(4), a(5)]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

[a(4), a(5)]

Return max(L, M, R)

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

P [..., a(3), a(4), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

maxSum [... a(3), a(4), ...]

分治法: 子子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

L=P [a(3)]

R=P [a(4), a(5)]

M=maxSum [... a(3), a(4), ...]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

[a(3), a(4), a(5)]

Return max(L, M, R)

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

P [..., a(2), a(3),...]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

maxSum [..., a(2), a(3),...]

分治法: 子問題

[a(1), a(2), a(3), a(4), a(5)]

L=P [a(1), a(2)]

R=P [a(3), a(4), a(5)]

M=maxSum [..., a(2), a(3),...]

分治法: 合併問題 (回傳解)

[a(1), a(2), a(3), a(4), a(5)]

Return max(L, M, R)

分治法: 原問題

P [a(1), a(2), a(3), a(4), a(5)]

分治法: 原問題

G=P [a(1), a(2), a(3), a(4), a(5)]

得到原問題的解了

分治法: 複雜度

假設原問題大小為: N

考慮實際時間花費

分治法: 時間花費

原問題時間花費為: T(N)

分割問題後為: T(N/2) + T(N/2)

maxSum: N

分治法: 時間花費

合併問題得 $T(N) = 2*T(N/2) + N$

並且最小子問題 $T(1) = 1$

分治法: 時間花費

$$T(N)$$

$$= 2^1 * T(N/2^1) + 1 * N = 2^1 * (\ 2 * T(N/2^2) + N/2^1\ ) + 1 * N$$

$$= 2^2 * T(N/2^2) + 2 * N = 2^2 * (\ 2 * T(N/2^3) + N/2^2\ ) + 2 * N$$

$$= 2^3 * T(N/2^3) + 3 * N = 2^3 * (\ 2 * T(N/2^4) + N/2^3\ ) + 3 * N$$

...

$$= 2^? * T(1) + ? * N$$

想想看 “問號” 為多少？

分治法: 複雜度

$T(N) = 2^{\lg N} * T(1) + (\lg N) * N$

$\wedge\, T(1) = 1$

$\Rightarrow T(N) = 2^{\lg N} + N\lg N$

$\Rightarrow T(N) = O(N\lg N)$

演算法的設計思維

-枚舉
-動態規劃
-分治法
-貪心法

貪心法

- 每次做一個在當下看起來最佳的決策
- 進而漸漸求出全局最佳解

- 貪心法是動態規劃的特例

貪心法: 最大連續和問題

```
int best = A[1], sum = 0;

for (int R = 1; R <= N; R++) {
  sum = max(A[R], sum + A[R]);
  best = max(best, sum);
}
```

複雜度為 O(N)

更優的複雜度?

枚舉、動態規劃、分治法、貪心法

這些思考方式能讓我們想出怎樣設計演算法
但可不能只滿足於此，要不斷的思考是否還存在別的演算法

Questions?