C/C++ 進階班
資結演算法

分治法
(Divide and Conquer)

李耕銘

課程大綱

- 分治法簡介
- 分治法常見應用
  - ➢ 河內塔 (Hanoi Tower)
  - ➢ 合併排序 (Merge Sort)
  - ➢ 快速排序 (Quick Sort)
  - ➢ 最大子數列問題 (Maximum Subarray)
  - ➢ 矩陣相乘 (Matrix Multiplication)
  - ➢ 選擇問題 (Selection Problem)
  - ➢ 最近點對問題 (Closest Pair of Points Problem)
- Master theorem
- 實戰練習

分治法簡介

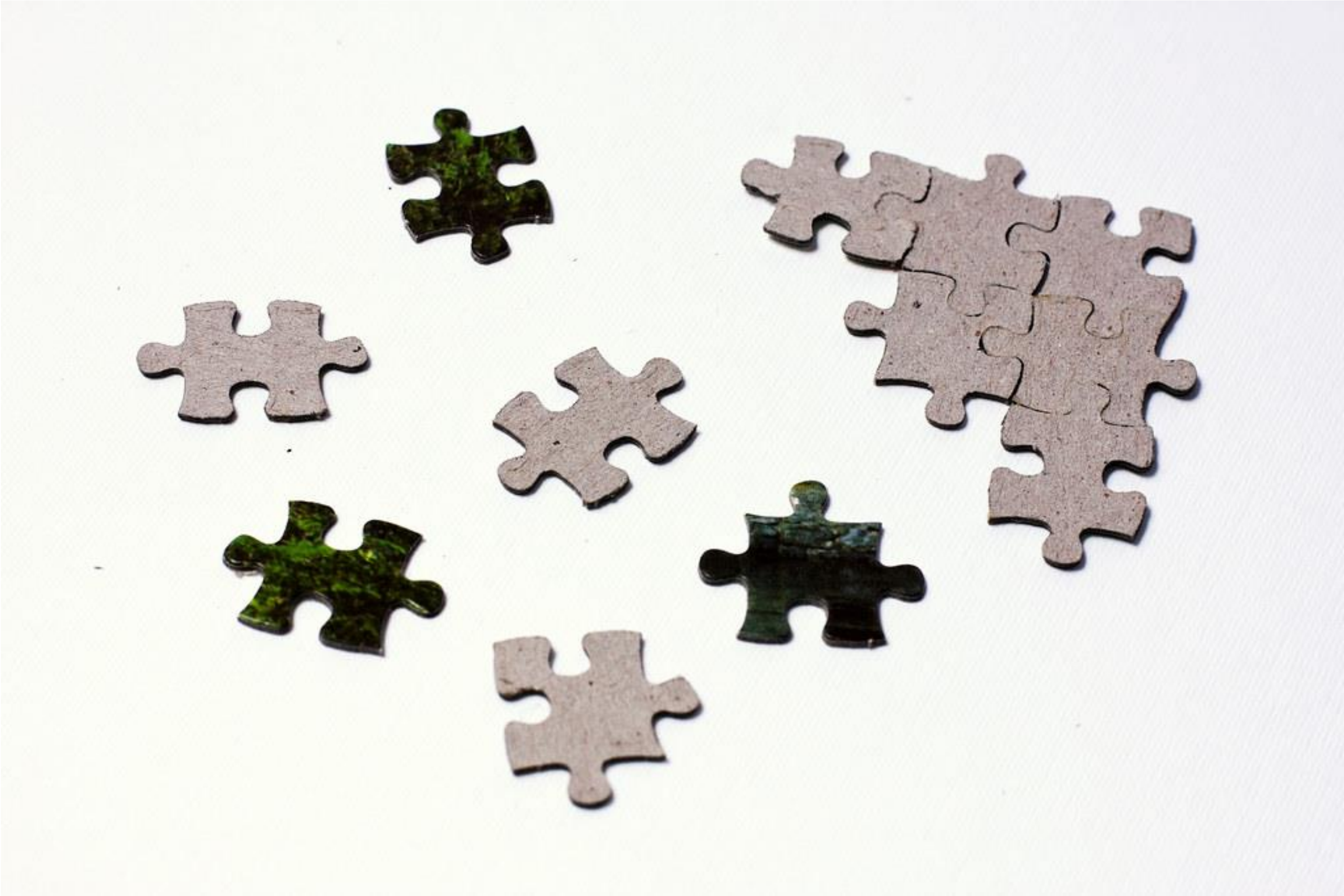
分治法簡介

- 分治法 (Divide and Conquer)
  1. 是一種程式設計的策略 (Strategy)
     - ✓ 並沒有固定的 Pseudocode
     - ✓ 貪婪演算法、動態規劃亦同
  2. 沒有共同定義的算法
     - ✓ 學習這些演算法背後的目的精神是很重要的
       - ◘ 為什麼要這樣做？
       - ◘ 這樣做可以包含所有可能或答案嗎？
       - ◘ 這樣做的複雜度如何？

分治法簡介

- 分治法 (Divide and Conquer)
  1. 顧名思義，「切割」與「征服」
  2. 把原(大)問題切割成幾個小問題後再分別征服之
  3. 大小問題的解法是一樣的，只有輸入資料不同
     - ✓ 利用遞迴來解決這些問題
     - ✓ 把問題縮小到一定規模後就可以直接求解
  4. 最後再把小問題的答案合併成原問題的答案
- *“利而誘之，亂而取之，實而備之，強而避之，怒而撓之，卑而驕之，佚而勞之，親而離之。攻其無備，出其不意。此兵家之勝，不可先傳也。” -孫子兵法*

分治法簡介



遞迴 (Recursion)

疊代：通常使用 for 迴圈跑遍所有範

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

```
int factorial(int n) {
    int sum = 1;
    for(int i = 1; i <= n; i++)
        sum *= i;
    return sum;
}
```

遞迴 (Recursion)

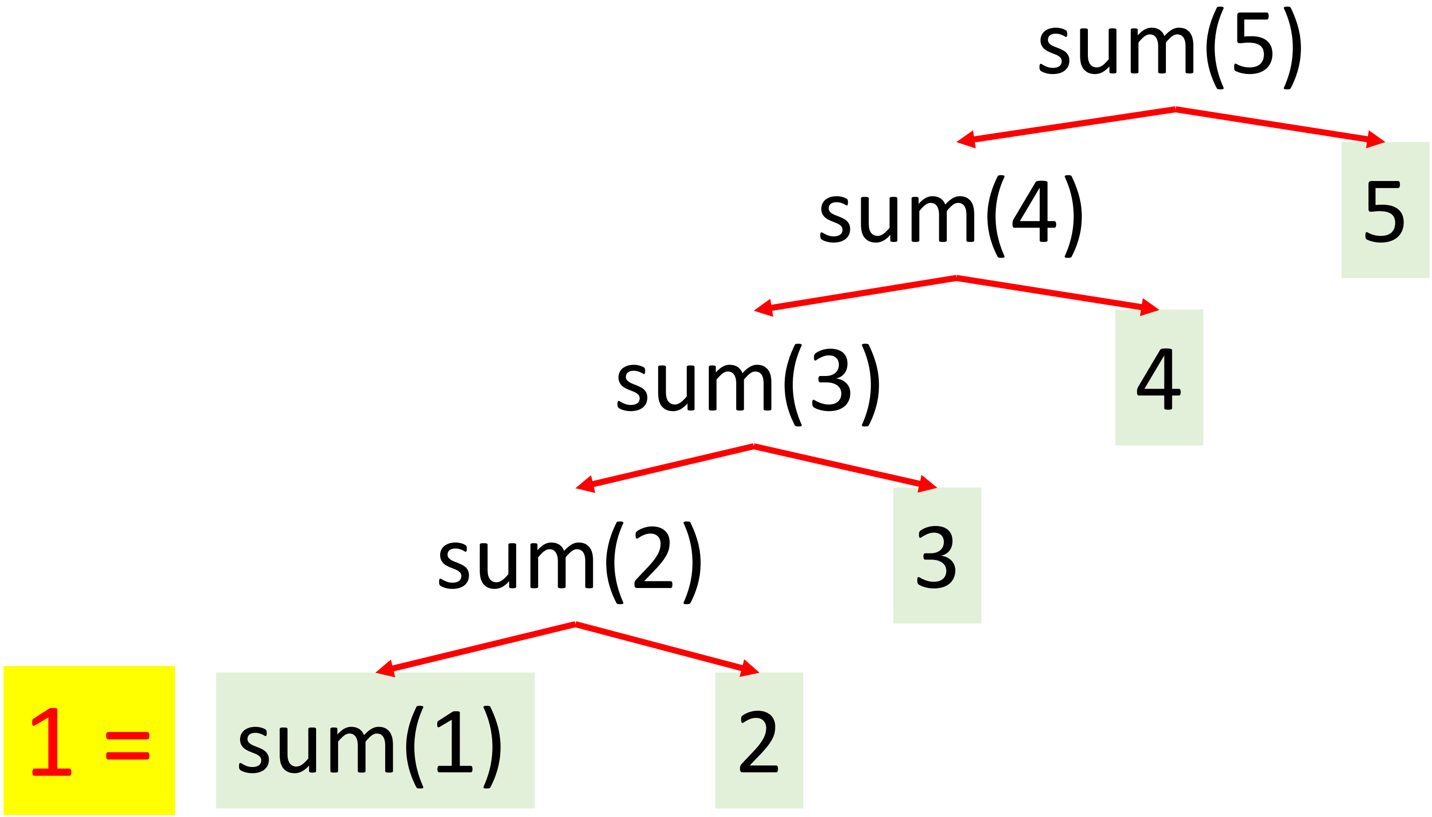
遞迴(Recursive)：把步驟進一步簡化後重新呼叫函式

```
sum(n) = 1 + 2 + 3 + 4 ... + n
sum(n) = 1 + 2 + 3 + 4 ... + n-1 + n
sum(n) = sum(n-1) + n
```

```
int sum(int n) {
    if(n == 1)
        return 1;
    else
        return sum(n-1) + n;
}
```

遞迴 (Recursion)

```
sum(n) = 1 + 2 + 3 + 4 ... + n
sum(n) = 1 + 2 + 3 + 4 ... + n-1 + n
sum(n) = sum(n-1) + n
```



遞迴 (Recursion)

- 化簡到哪裡可以結束
- 如何化簡問題

```
int sum(int n) {
    if(n == 1)
        return 1;
    else
        return sum(n-1) + n;
}
```

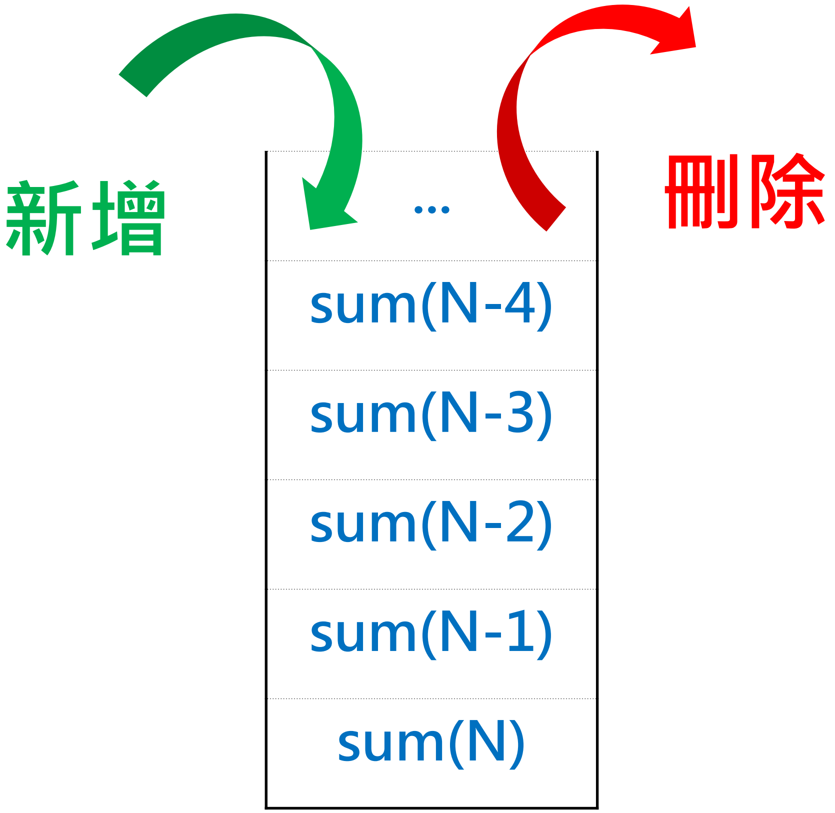
分治法簡介

遞迴

- 優點：容易設計、簡潔、易懂、直觀
  1. 如何化簡問題
  2. 化簡到哪裡可以結束
- 缺點：效率通常較差
  1. 每次呼叫就必須為函式配置記憶體
  2. 可能有重複運算與記憶體消耗的問題
  3. Stackoverflow

stackoverflow

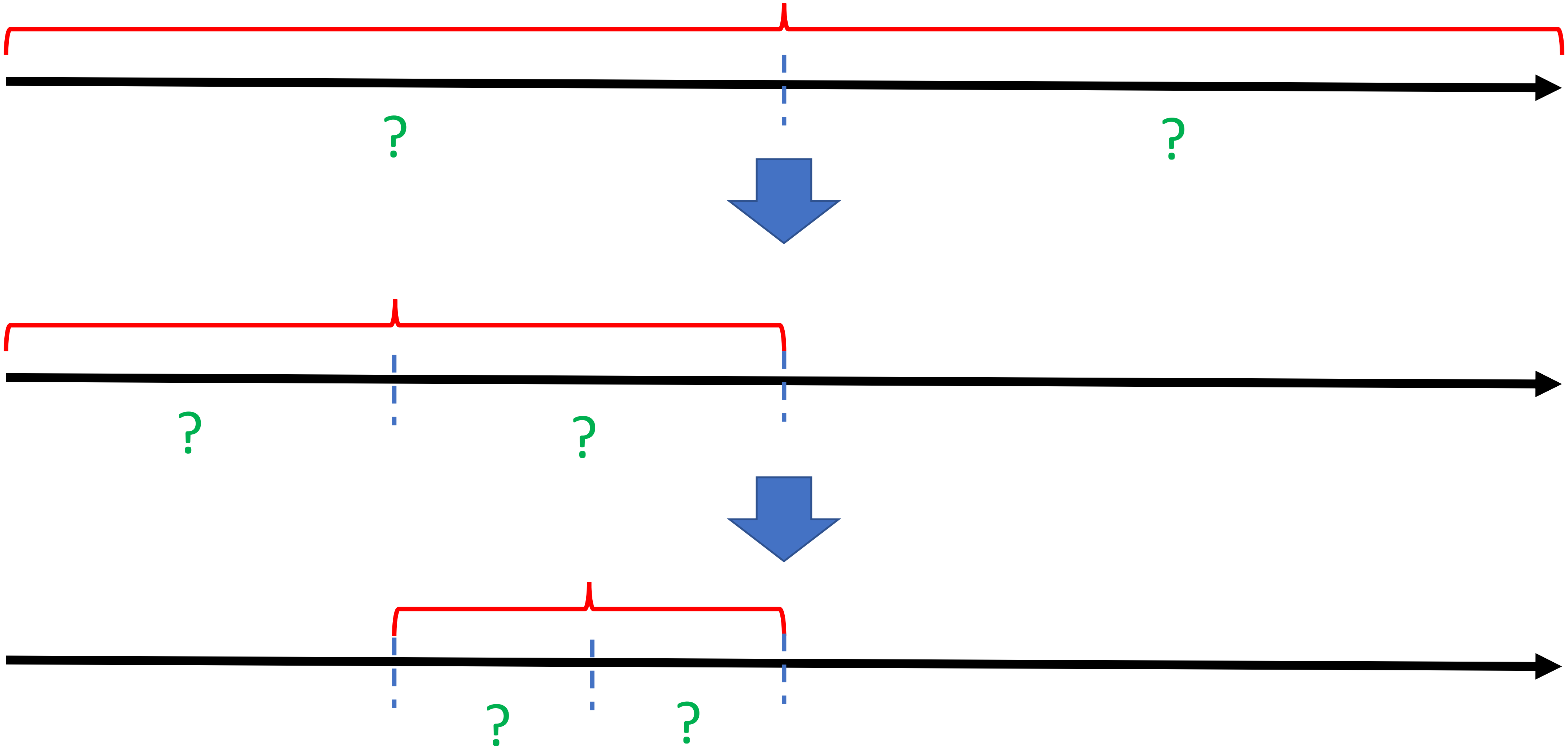
```
int sum(int N){
    return sum(N-1) + N;
}
```



二分搜尋法

- 每次搜尋時，用**刪去法刪去一半**的可能
  - ➢ 資料必須事先排序好，才知道要刪哪一半
  - ➢ 需要支援隨機存取(索引值)，否則效能低落
  - ➢ 每次搜尋會分成三種狀況
    1. 確定找或找不到資料！
    2. 沒有找到資料，但它會出現在陣列的前半部
    3. 沒有找到資料，但它會出現在陣列的後半部

二分搜尋法



二分搜尋法

- 二分搜尋法每一輪比較後都會有一半的資料區間被刪去
  - ➢ 可視為特化的分治法
    - ✓ 分治法把母問題切割成子問題 (Divide)
    - ✓ 再使用同樣的算法或函式分別處理 (Conquer)
    - ✓ 最後再把每個小問題的答案合併成母問題的 (Combine)
  - ➢ 但二分搜尋法將切割後直接將答案不存在的區間削減掉
    - ✓ 再繼續往下搜尋

遞迴 (Recursion)

斐波那契數列

1. 前兩項定義為 1
2. 之後每一項都是前面兩個的和

Ex：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89......

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

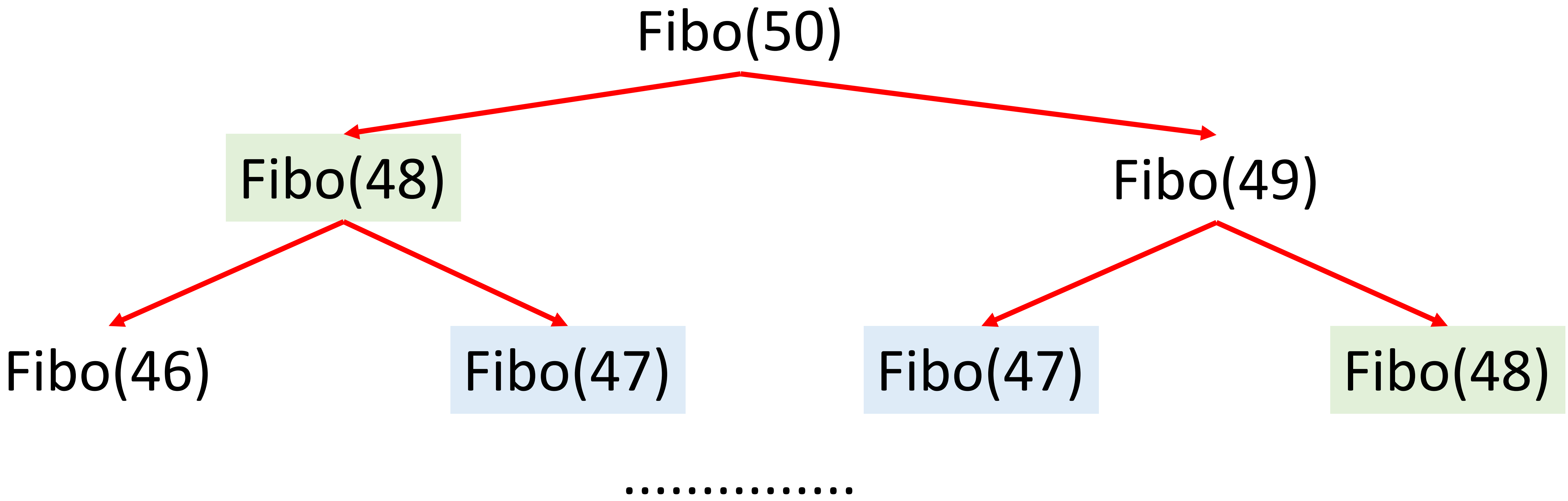
Example Code

Mission

利用分治法與遞迴求費波那契數列。

遞迴 (Recursion)

```
fibo(1) = 1
fibo(2) = 1
fibo(n) = fibo(n-1) + fibo(n-2)
```



遞迴 (Recursion)

```
int fibo(int n) {
    int *data = (int*) malloc(sizeof(int)*n);
    data[0] = 1;
    data[1] = 1;
    for(int i=2;i<n;i++){
       data[i] = data[i-1] + data[i-2];
    }
    int result = data[n-1];
    free(data);
    return result;
}
```

疊代：效率好、架構不清楚

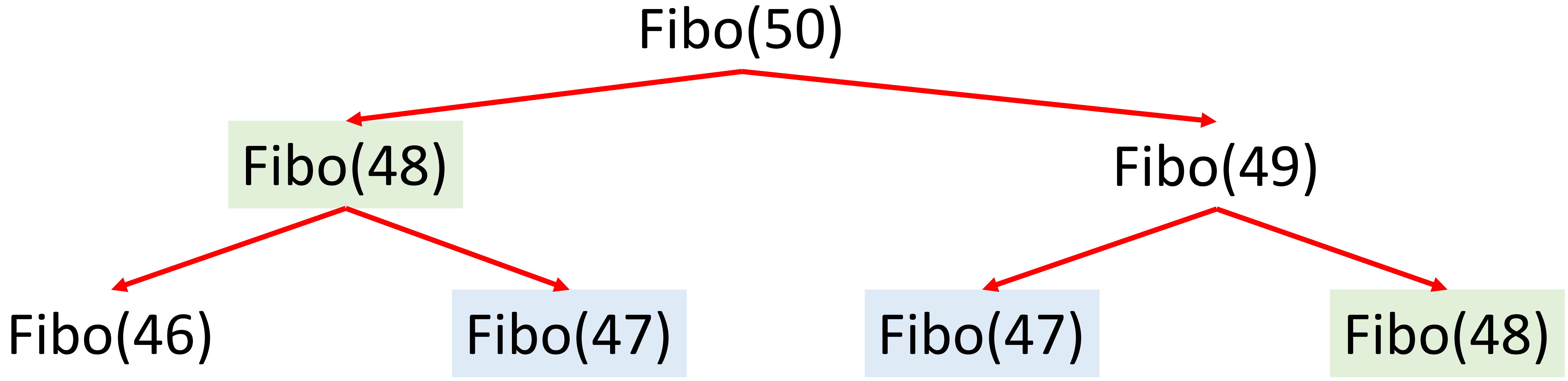
(這個方法就是動態規劃)

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

遞迴：效率不好、架構清楚

分治法簡介

- 不適合使用分治法的情境
  - ➢ 問題會被切割成兩個以上的子問題
    - ✓ 假設是 n 個子問題，拆解 k 次後變成：n^k
  - ➢ 但特定狀況下指數成長無法避免的
    - ✓ 河內塔：每呼叫一次只能搬動一個圓盤，無法更快



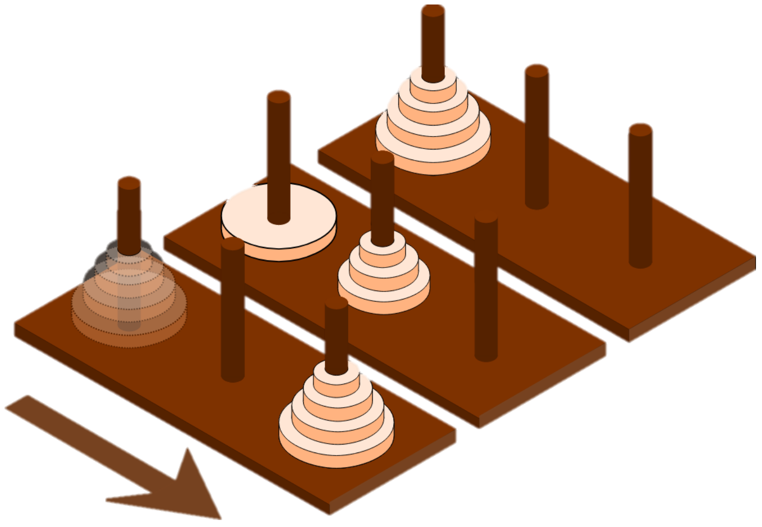
分治法簡介

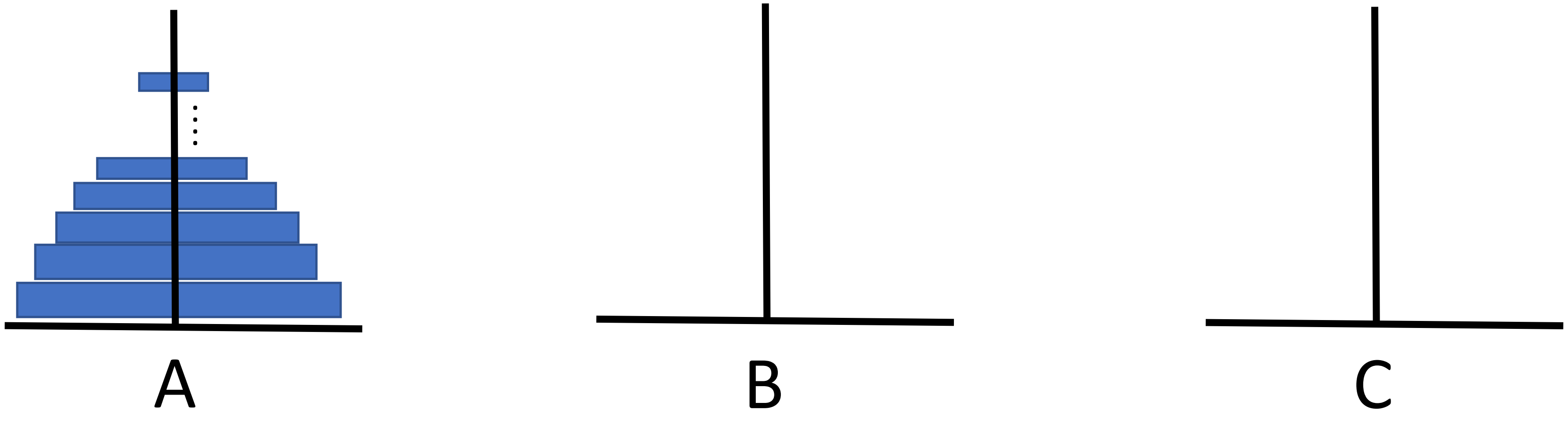
- 分治法總結
  - 可能適用：問題可以被切割成單一、且同樣的的子問題
  - 可能不適用：問題會被切割成兩個以上的子問題 (視情況而定)
    - ✓ 嘗試使用動態規劃？
  - 步驟：
    1. (Divide) 把原問題切割成許多同樣的子問題
       - ✓ 利用遞迴切割這些子問題
    2. (Conquer) 當切割到夠小的時候，就直接解決它
    3. (Combine) 把這些小問題的答案合併成原問題的答案

分治法常見應用

河內塔

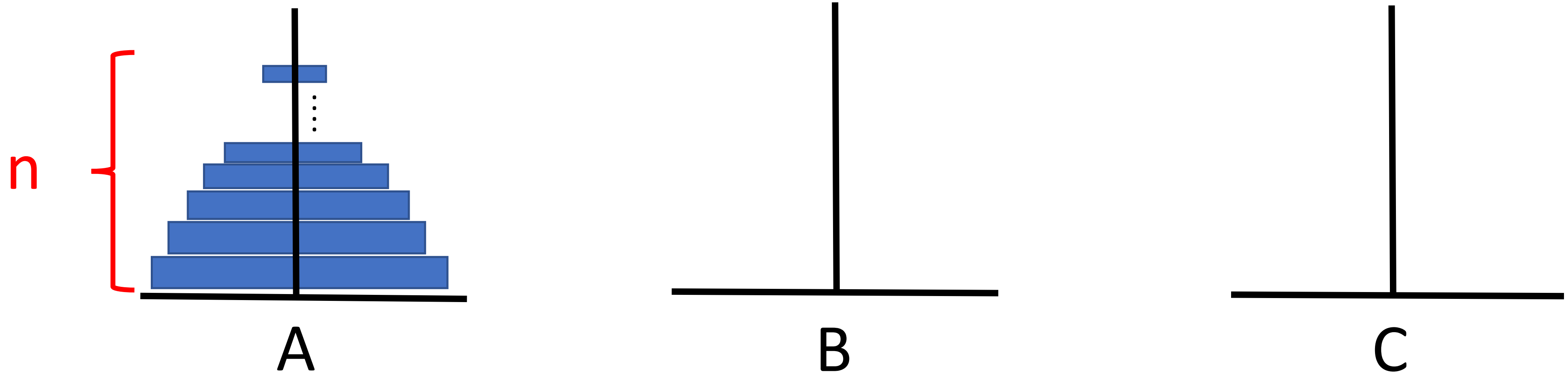
- 河內塔
  1. 有三根可以置放圓盤的棍子
  2. 圓盤依序由小到大，不重複
  3. 圓盤只能依照大小插在圓棍上 (小的放在大的上)
- 把 n 個圓盤從一根棍子移到另一根棍子的過程





河內塔

最大的盤子要先從 A 移到 C

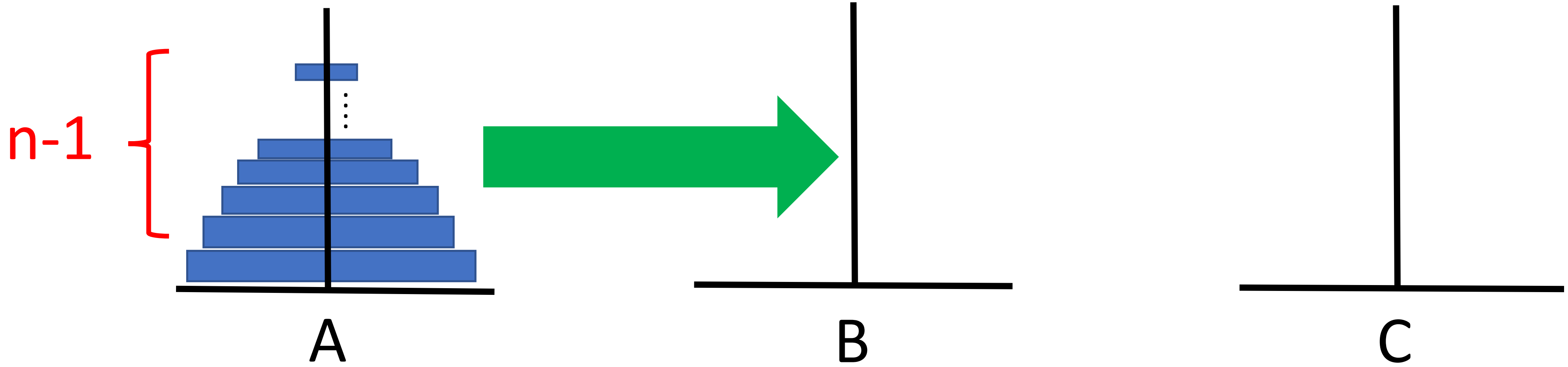


$$Hanoi(n) =$$

河內塔

最大的盤子要先從 A 移到 C

→須淨空最大盤子上的所有盤子

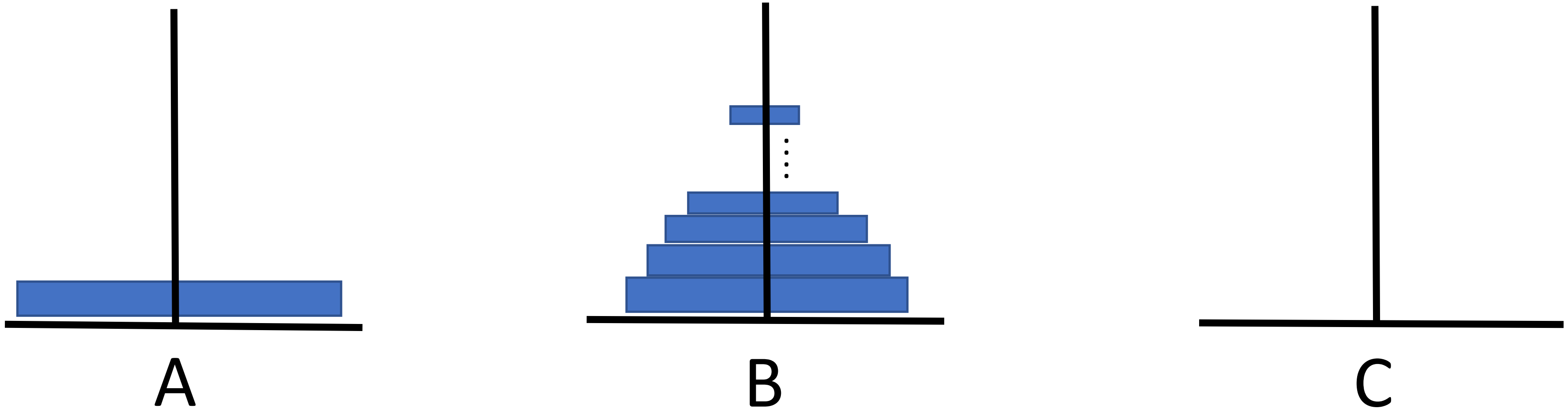


$$Hanoi(n) =$$

河內塔

最大的盤子要先從 A 移到 C

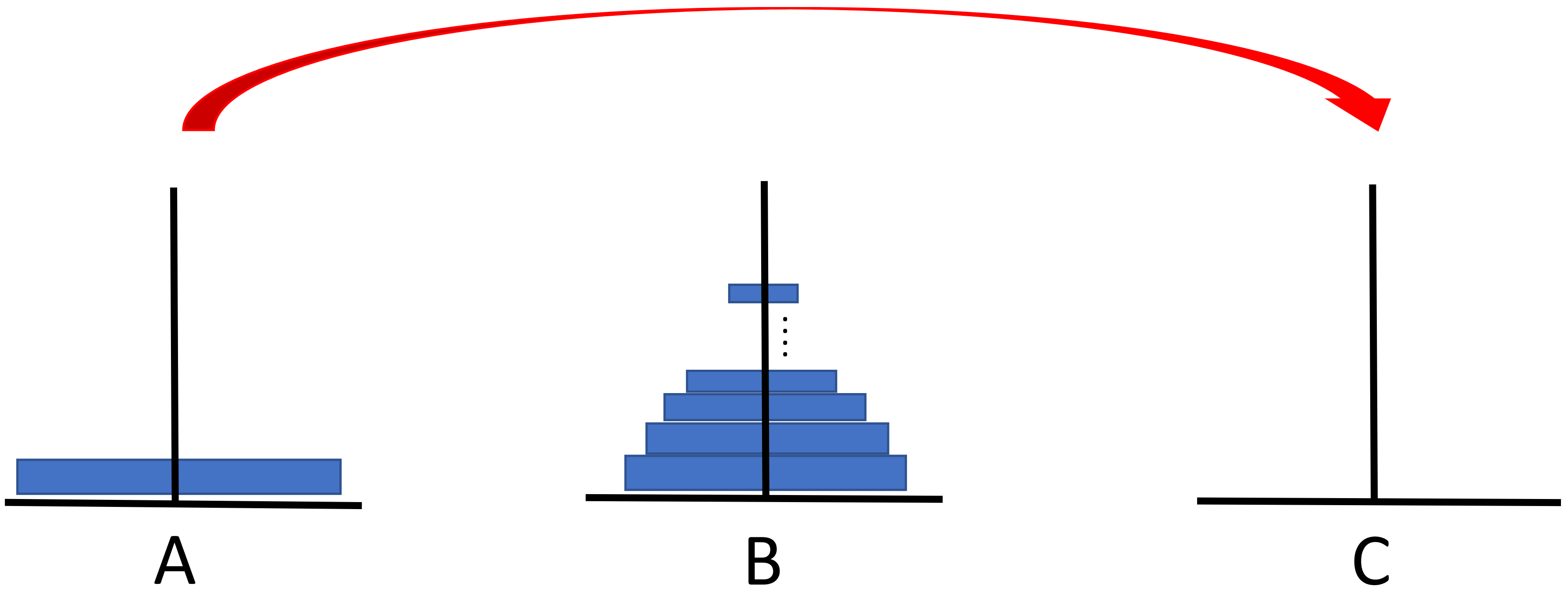
→須淨空最大盤子上的所有盤子



$$Hanoi(n) = Hanoi(n-1)$$

河內塔

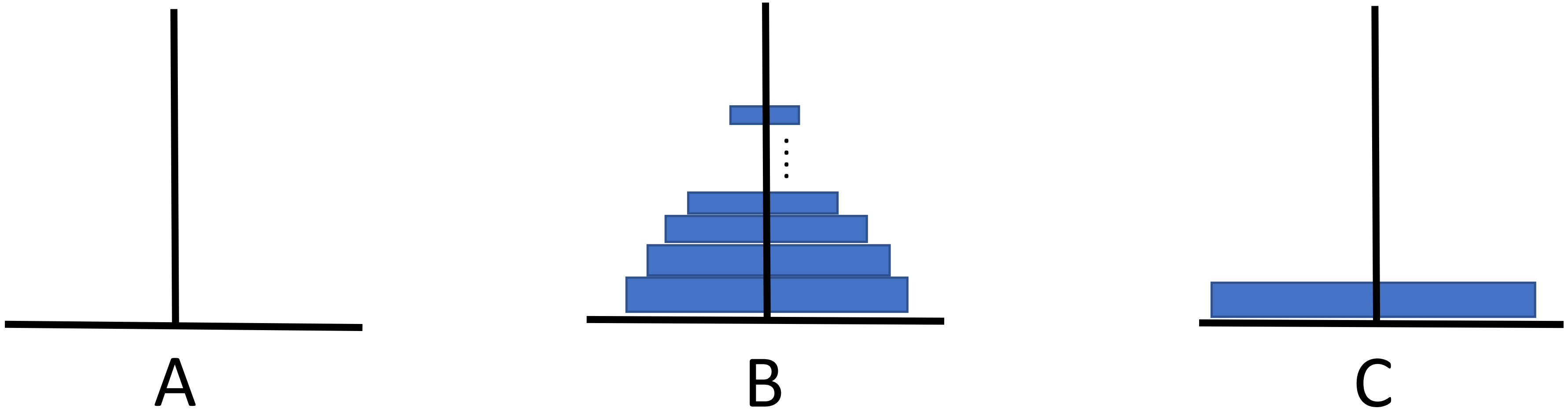
最大的盤子移動到目的地



$$Hanoi(n) = Hanoi(n-1)$$

河內塔

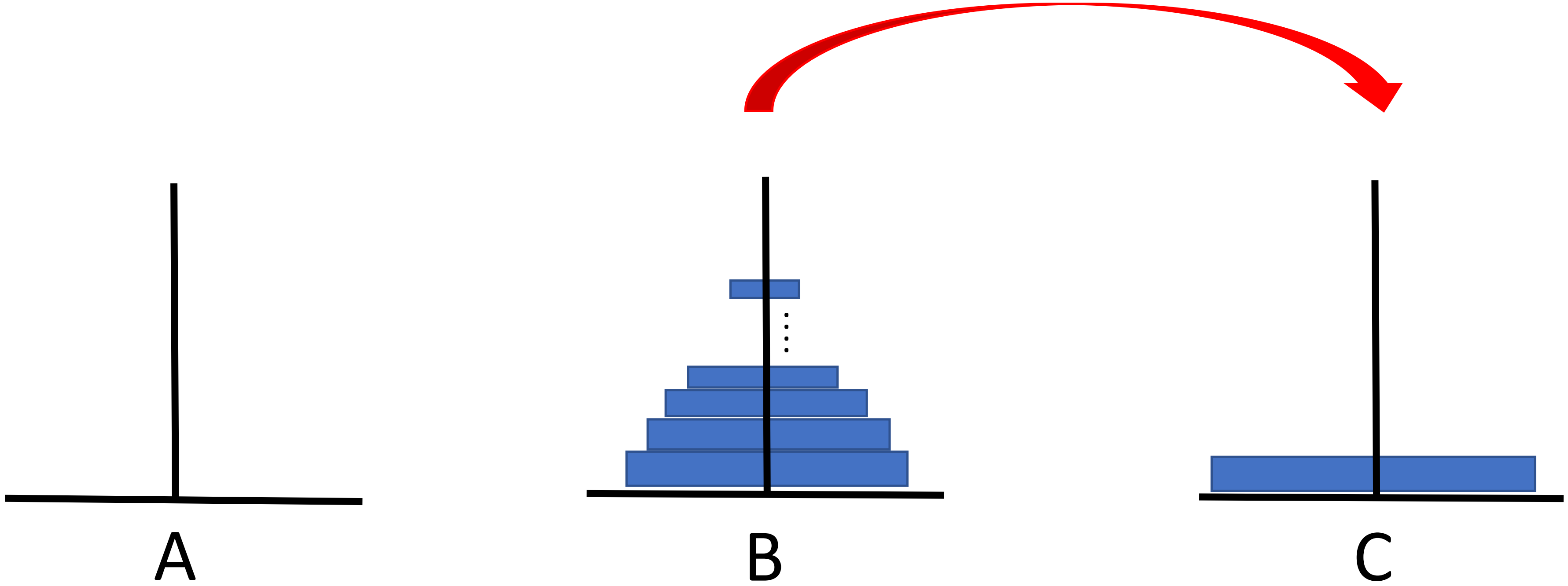
最大的盤子移動到目的地



$$Hanoi(n) = Hanoi(n-1) + Hanoi(1)$$

河內塔

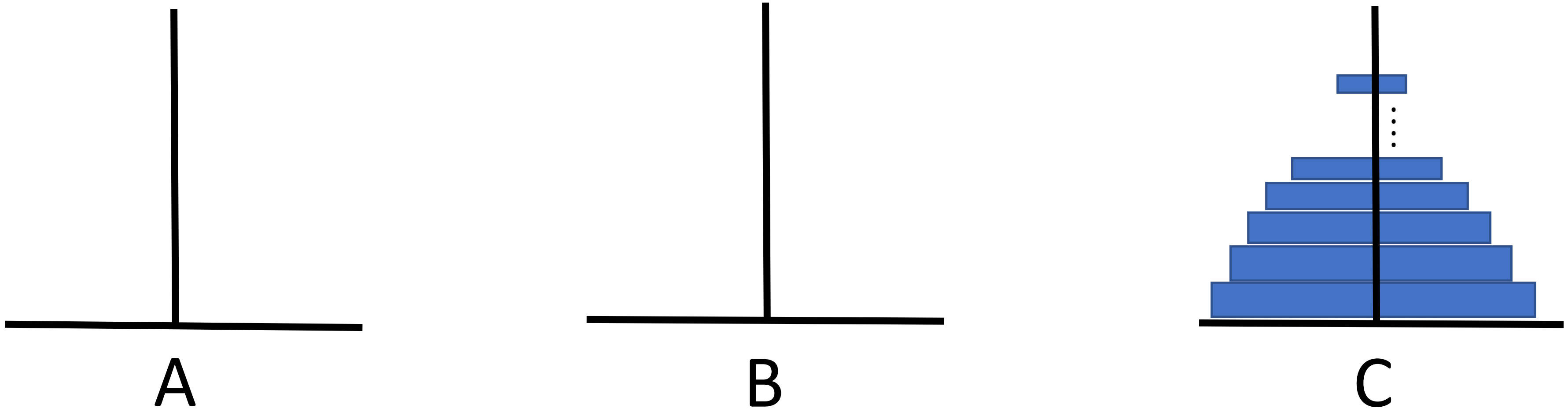
移動 n-1 個盤子到目的地



$$Hanoi(n) = Hanoi(n-1) + Hanoi(1)$$

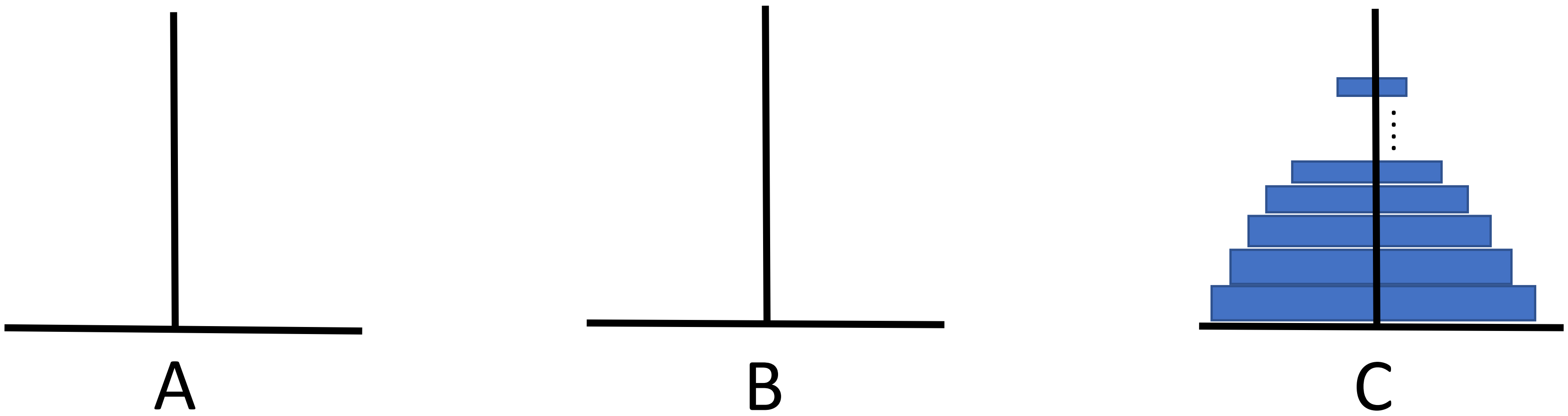
河內塔

移動 n-1 個盤子到目的地



$$Hanoi(n) = Hanoi(n-1) + Hanoi(1) + Hanoi(n-1)$$

河內塔



$$Hanoi(n) = Hanoi(n-1) + Hanoi(1) + Hanoi(n-1)$$

$$T(n) = \begin{cases} 1, if\ n = 1 \\ 2T(n-1) + 1, if\ n \geq 2 \end{cases}$$

河內塔

$$T(n) = \begin{cases} 1, if\ n = 1 \\ 2T(n-1) + 1, if\ n \geq 2 \end{cases}$$

$T(n)$

$\leq 2T(n-1) + 1$

$\leq 2[2T(n-2) + 1] + 1 = 4T(n-2) + 1 + 2$

$\leq 4[2T(n-3) + 1] + 1 + 2 = 8T(n-3) + 1 + 2 + 4$

....

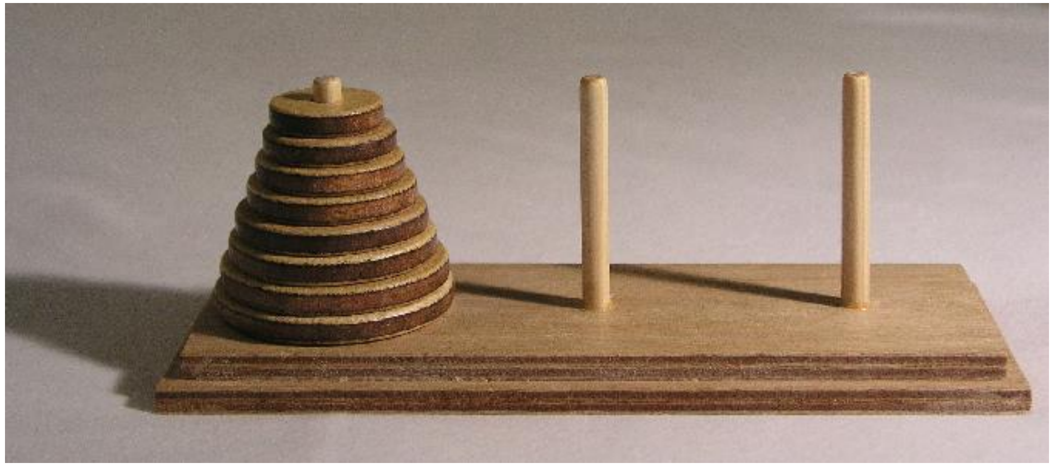
$\leq 2^{n-1}T(1) + 2^{n-1} - 1 = 2^n - 1$

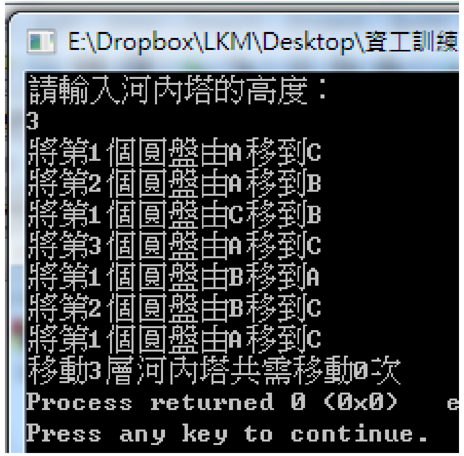
$$T(n) = O(2^n)$$

Practice

Mission

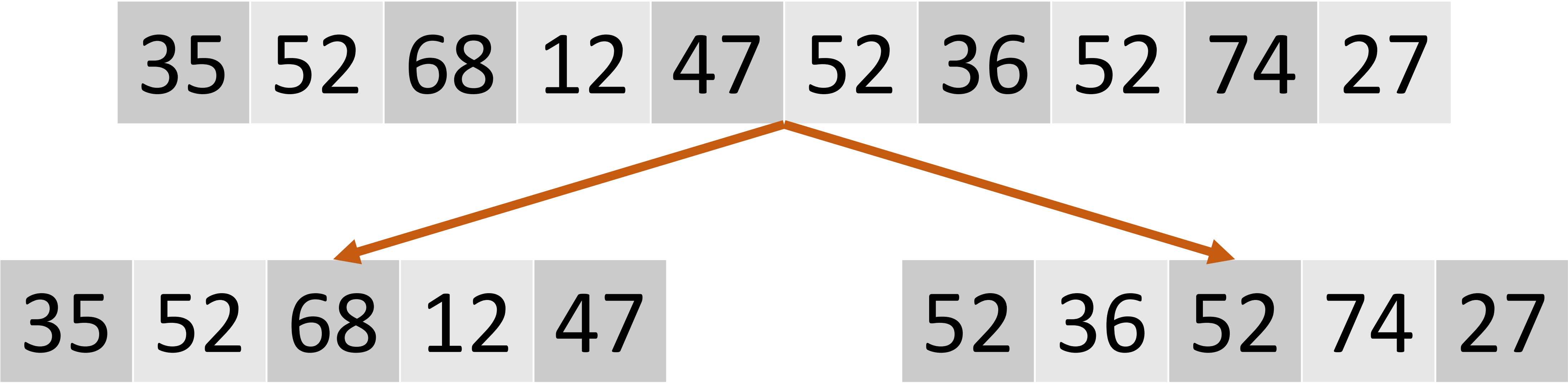
輸入一整數 n，請輸出搬動 n 層的所有過程。





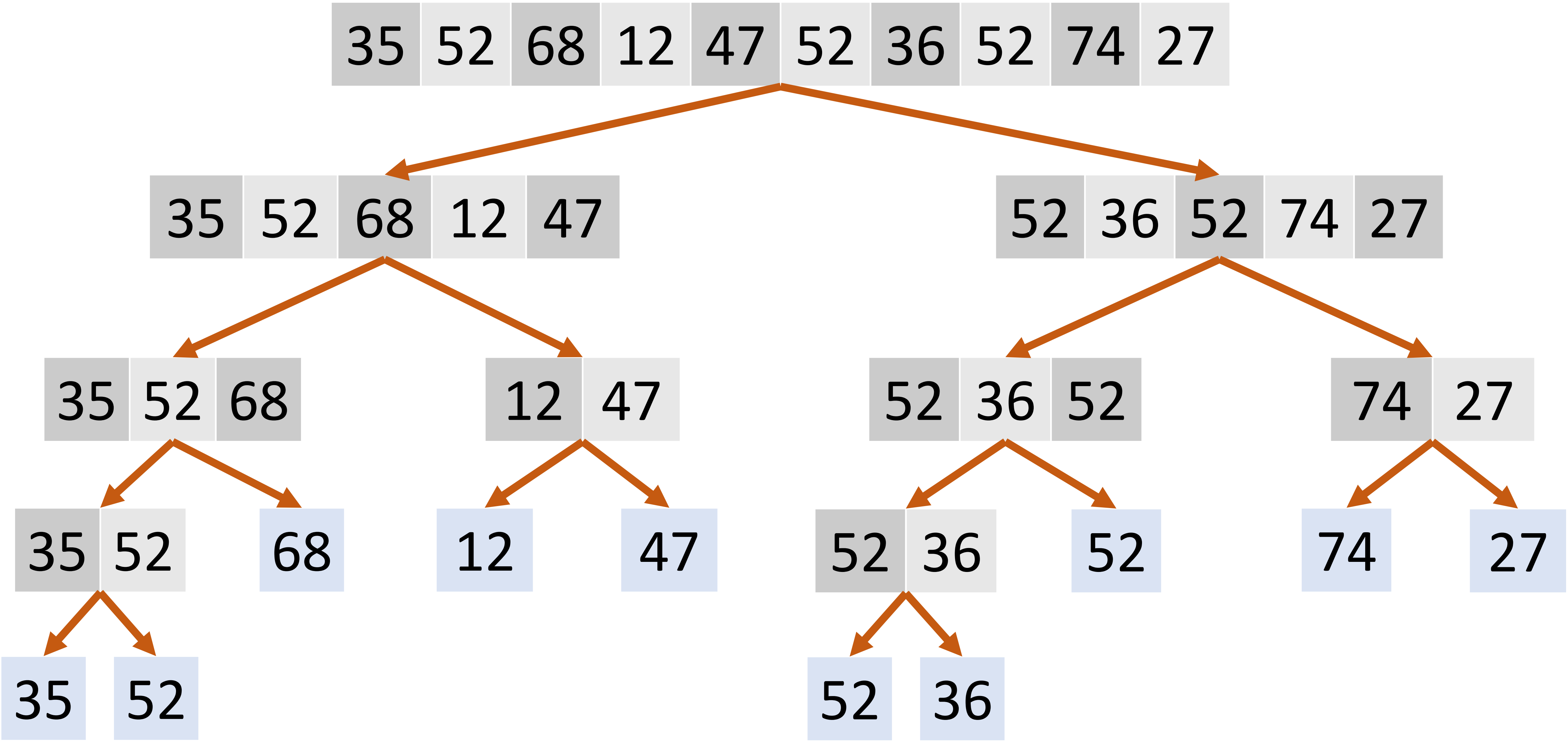
合併排序

- 切割資料後再融合
  - ➢ 把資料切成兩組，分別排序
  - ➢ 再把已排序好的兩組資料融合在一起
  - ➢ 分治法 (Divide and Conquer) 的應用



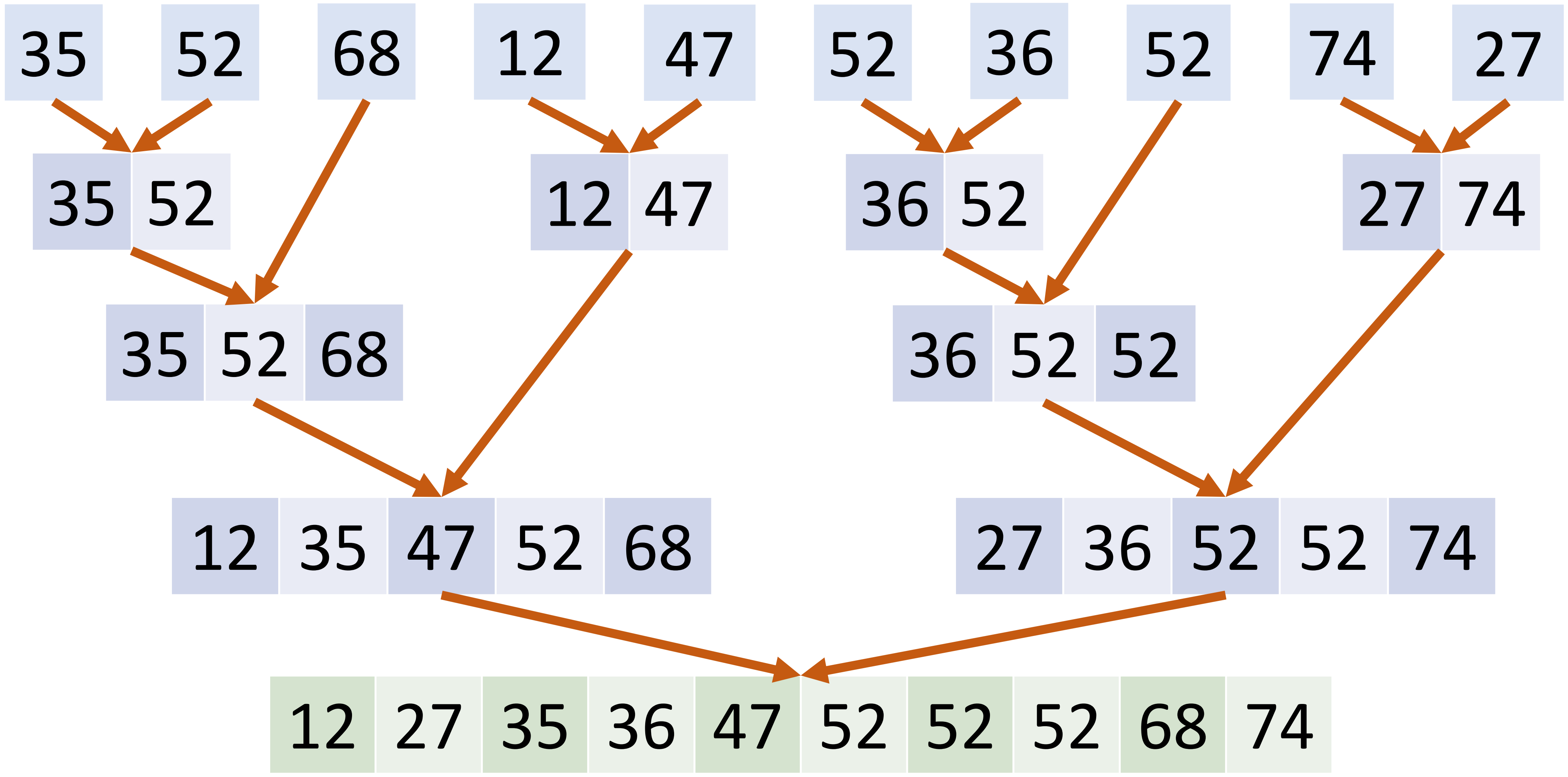
Divide

合併排序


35 52 68 12 47 52 36 52 74 27
35 52 68 12 47
52 36 52 74 27
35 52 68
12 47
52 36 52
74 27
35 52
68
12
47
52 36
52
74
27
35
52
52
36

Conquer

合併排序



合併排序

```
void Merge_Sort(int data[],int start,int finish){
    if(finish>start){
        int middle = (finish+start)/ 2;
        Merge_Sort(data, start, middle);
        Merge_Sort(data, middle+1, finish);
        Merge(data,start,finish,middle);
    }
}
```

Divide&Conquer

Combine

$$T(n)=\begin{cases} O(1), if\ n=1 \\ T\left(\left\lceil\frac{n}{2}\right\rceil\right)+T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)+O(n), if\ n\geq 2 \end{cases}$$

合併排序

$$T(n) = \begin{cases} O(1), if\ n = 1 \\ 2T(\frac{n}{2}) + O(n), if\ n \geq 2 \end{cases}$$

$$T(n)$$
$$\leq 2T\left(\frac{n}{2}\right) + cn$$
$$\leq 2\left[2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right] + cn = 4T\left(\frac{n}{4}\right) + 2cn$$
$$\leq 4\left[2T\left(\frac{n}{8}\right) + c\frac{n}{4}\right] + 2cn = 8T\left(\frac{n}{8}\right) + 3cn$$

....

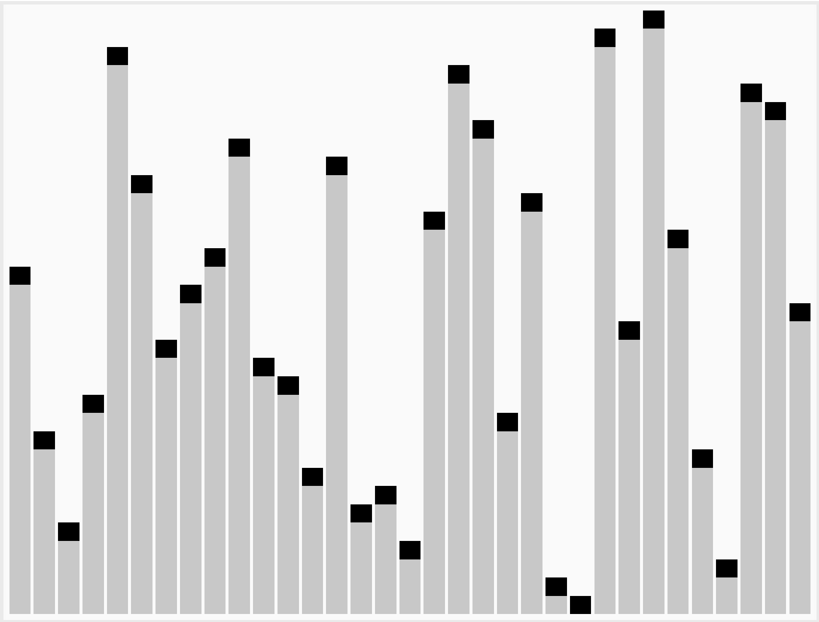
$$\leq 2^k T\left(\frac{n}{2^k}\right) + kcn, let\ k = log_2 n$$

$$T(n) \leq nT(1) + cnlog_2 n = O(n) + O(nlog_2 n)$$

$$T(n) = O(nlog_2 n)$$

快速排序

- 隨機選出一筆資料當基準點
  - ➢ 比該筆資料小的放左邊
  - ➢ 比該筆資料大的放右邊
  - ➢ 依序做至資料數目為1



快速排序

Divide & Conquer

```
void Quick_Sort(int data[], int start, int finish){
    if (start < finish) {
        int pivot = Partition(data, start, finish);
        Quick_Sort(data, start, pivot - 1);
        Quick_Sort(data, pivot + 1, finish);
    }
}
```

O(n)

$$T(n) = \begin{cases} O(1), if\ n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n), if\ n \geq 2 \end{cases}$$

最大子數列問題

給定一陣列，陣列的值有正有負，請找出一區間[a,b]可以使區間內的元素總和最大，並回傳該元素總和。

8	-5	-1	4	-3	6	2	-2	3	4

暴力解：

$O(n^3)$

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int n = nums.size();
        int sum[n][n]; // Should use malloc or vector instead
        int maximum = -2147483648;
        for(int start=0;start<n;start++){
            for(int finish=start;finish<n;finish++){
                sum[start][finish] = 0;
                for(int k=start;k<=finish;k++){
                    sum[start][finish] += nums[k];
                }
                if(sum[start][finish]>maximum){
                    maximum = sum[start][finish];
                }
            }
        }
        return maximum;
    }
};
```

$O(n^3)$

最大子數列問題

優化的暴力解：

$O(n^2)$

sum_1_n 存 1 到 n 元素和

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int n = nums.size();
        int sum_1_to_n[n]; // Should use malloc or vector instead
        int sum[n][n]; // Should use malloc or vector instead
        int maximum = -2147483648;
        for(int i=0;i<n;i++){
            if(i==0)
                sum_1_to_n[i] = nums[0];
            else
                sum_1_to_n[i] =  sum_1_to_n[i-1] + nums[i];
        }
        for(int start=0;start<n;start++){
            for(int finish=start;finish<n;finish++){
                if(start)
                    sum[start][finish] = sum_1_to_n[finish]-sum_1_to_n[start-1];
                else
                    sum[start][finish] = sum_1_to_n[finish];
                if(sum[start][finish]>maximum){
                    maximum = sum[start][finish];
                }
            }
        }
        return maximum;
    }
};
```

$O(n)$

$O(n^2)$

start 到 finish 的和變：O(1)

最大子數列問題

分治法：向二分搜尋法學習

最大值出現在左邊或右邊？

$max\_subarray(0, \left\lfloor \frac{n}{2} \right\rfloor)$ $max\_subarray(\left\lfloor \frac{n}{2} \right\rfloor, n)$

8	-5	-1	4	-3	6	2	-2	3	4

$$max\_subarray(0, n)$$
$$= \max\left(max\_subarray(0, \left\lfloor \frac{n}{2} \right\rfloor), max\_subarray(\left\lfloor \frac{n}{2} \right\rfloor, n)\right)$$

這樣分割問題，對嗎？

最大子數列問題

分治法：向二分搜尋法學習

最大值出現在左邊或右邊？

$max\_subarray(0, \lfloor \frac{n}{2} \rfloor)$

$max\_subarray(\lfloor \frac{n}{2} \rfloor, n)$

8	-5	-1	4	-3	6	2	-2	3	4

有可能出現在中間！

最大子數列問題

$max\_subarray(0, \lfloor \frac{n}{2} \rfloor)$ $max\_subarray(\lfloor \frac{n}{2} \rfloor, n)$

8	-5	-1	4	-3	6	2	-2	3	4

$max\_cross\_array(0, n)$

1. $max\_subarray(0, \lfloor \frac{n}{2} \rfloor)$
2. $max\_subarray(\lfloor \frac{n}{2} \rfloor, n)$
3. $max\_cross\_array(0, n)$

回傳其中的最大值！

最大子數列問題

$max\_cross\_array(0, n)$

8	-5	-1	4	-3	6	2	-2	3	4

從中間(-3)往左右長，分別往左右長到最大值

左
1. 4
2. 4 - 1= 3
3. 3 - 5 = -2
4. -2 + 8 = 6

max_left = +6

右
1. 6
2. 6 + 2 = 8
3. 8 - 2 = 6
4. 6 + 3 = 9
5. 9 + 4 =13

max_right = +13

最大子數列問題

$max\_cross\_array(0, n)$

8	-5	-1	4	-3	6	2	-2	3	4

從中間(-3)往左右長，分別往左右長到最大值

總和：

max = -3 + max_left + max_right = 16

最大子數列問題

Divide
&
Conquer

```
max_left = maxSubArray(data_left);
max_right = maxSubArray(data_right);
max_center = maxCrossArray(data);

if(max_left>=max_center && max_left>=max_right)
    return max_left;
else if(max_right>=max_center && max_right>=max_left)
    return max_right;
else
    return max_center;
```

Combine

最大子數列問題

$$max\_cross\_array(0, n)$$

8	-5	-1	4	-3	6	2	-2	3	4

```
int max_center = nums[(len-1)/2];
int index_left = -1, index_right = 1,left_sum = 0,right_sum = 0, max = 0;
while((len-1)/2+index_left>=0){
    left_sum += nums[(len-1)/2+index_left];
    if(left_sum > max)
        max = left_sum;
    index_left--;
}
max_center += max;

max = 0;
while((len-1)/2+index_right<len){
    right_sum += nums[(len-1)/2+index_right];
    if(right_sum > max)
        max = right_sum;
    index_right++;
}
max_center += max;
```

最大子數列問題

Divide & Conquer

```
max_left = maxSubArray(data_left);
max_right = maxSubArray(data_right);
max_center = maxCrossArray(data);
```

$\to T\left(\left\lceil\frac{n}{2}\right\rceil\right)$ (max_left, max_right)

$\to O(n)$ (max_center)

```
if(max_left>=max_center && max_left>=max_right)
    return max_left;
else if(max_right>=max_center && max_right>=max_left)
    return max_right;
else
    return max_center;
```

Combine

$$T(n) = \begin{cases} O(1), if\ n = 1 \\ T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + O(n), if\ n \geq 2 \end{cases}$$

$$T(n) = O(nlog_2 n)$$

證明方法同 merge sort

最大子數列問題

下下章節的動態規劃可以把最大子數列問題壓在 O(n)

8	-5	-1	4	-3	6	2	-2	3	4

Practice 3

Mission

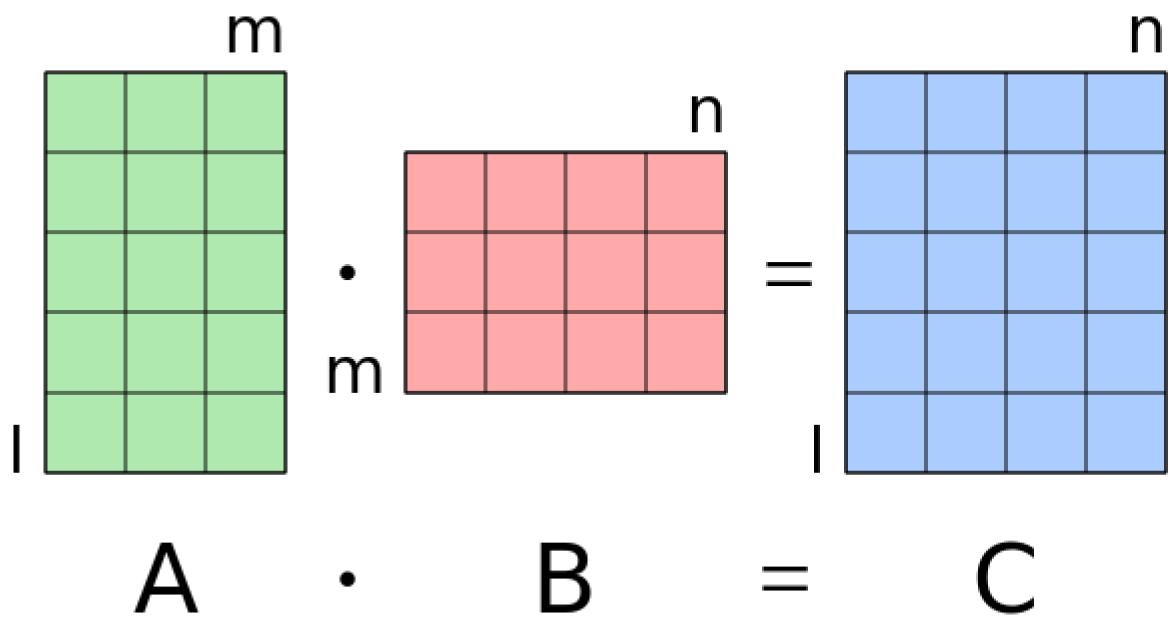
Try LeetCode #53. Maximum Subarray

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

- Example 1:
  - Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
  - Output: 6
  - Explanation: [4,-1,2,1] has the largest sum = 6.

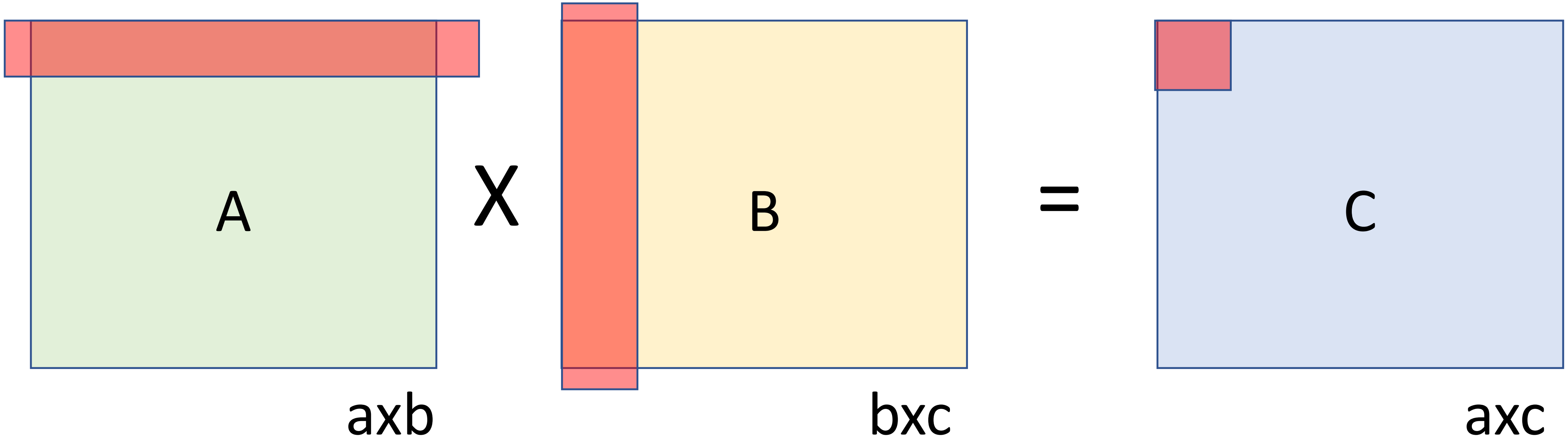
Ref： https://leetcode.com/problems/maximum-subarray/

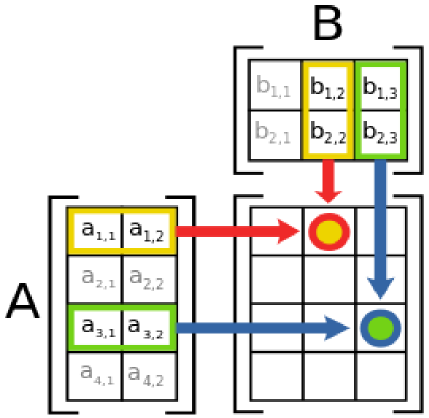
矩陣相乘



- 矩陣相乘
  - 輸入兩$n \times n$的矩陣
  - 輸出兩矩陣的乘積
  - 這裡只考慮正方形矩陣
  - 不常考，但電腦科學中很重要

矩陣相乘

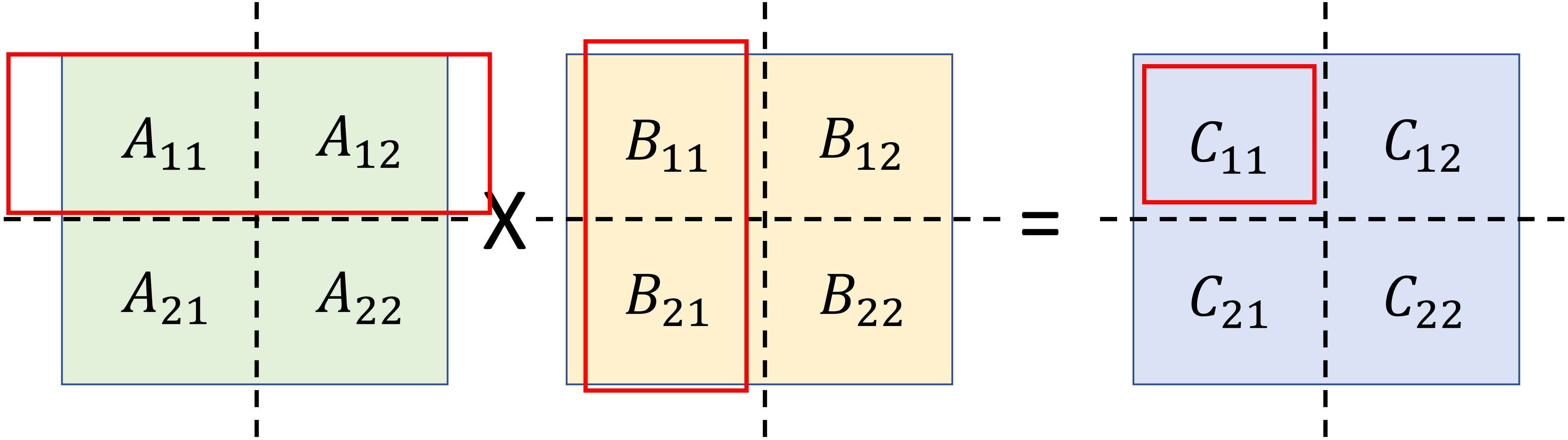




$$C[i,j] = \sum_{k=1}^{b} A[i,k] \times B[k,j]$$

$$T(n) = O(n^3)$$

矩陣相乘



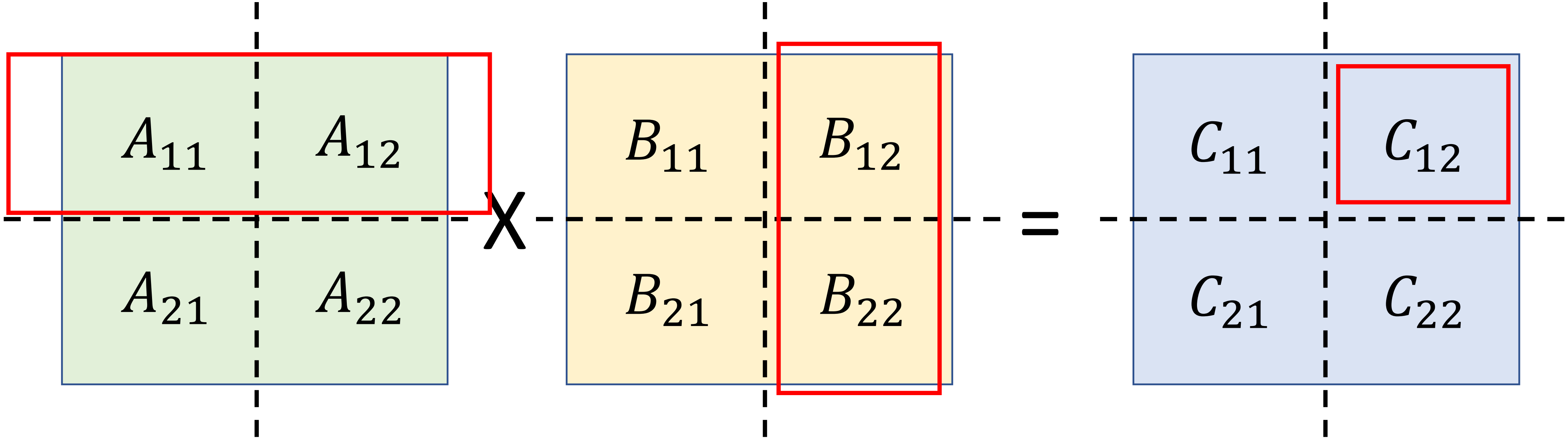
$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

矩陣相乘



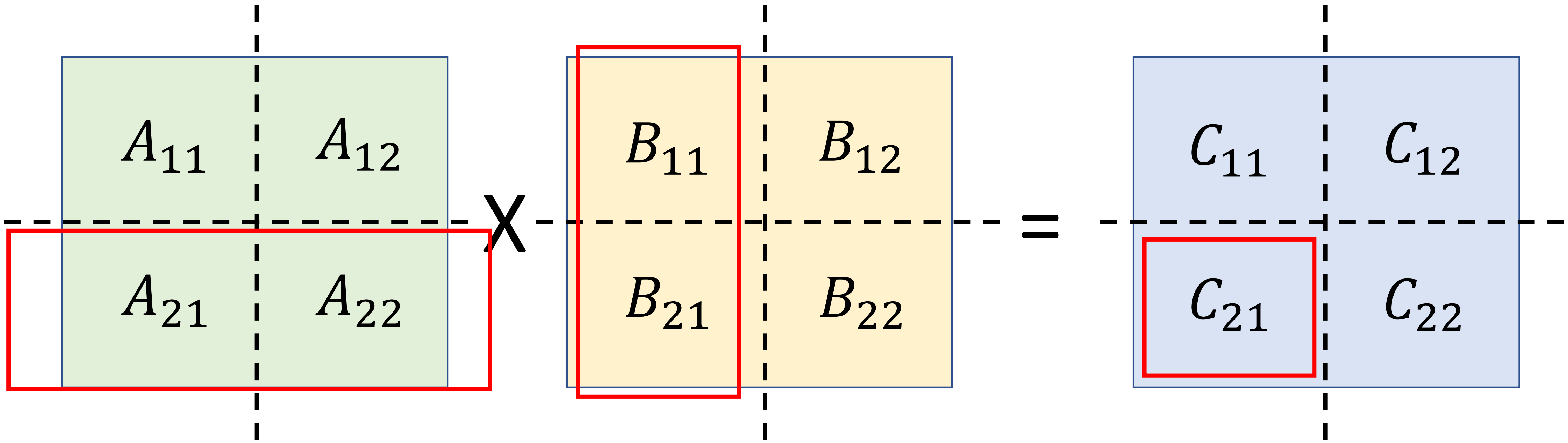
$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

矩陣相乘



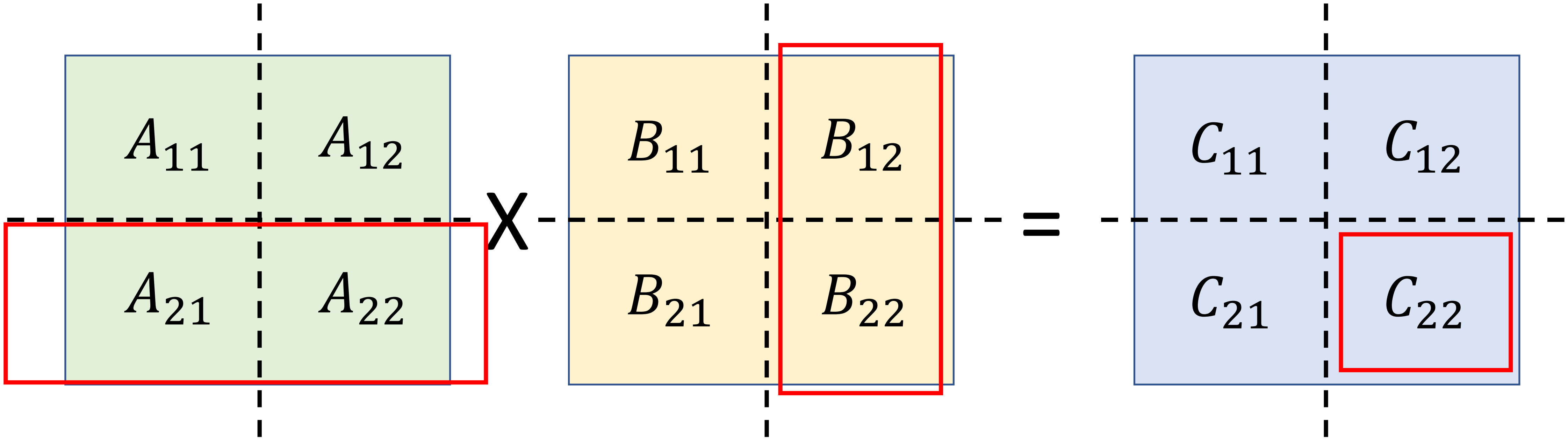
$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

矩陣相乘



$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

矩陣相乘

$$C_{11} = MatrixMultiply(\frac{n}{2}, A_{11}, B_{11}) + MatrixMultiply(\frac{n}{2}, A_{12}, B_{21})$$

$$C_{12} = MatrixMultiply(\frac{n}{2}, A_{11}, B_{12}) + MatrixMultiply(\frac{n}{2}, A_{12}, B_{22})$$

$$C_{21} = MatrixMultiply(\frac{n}{2}, A_{21}, B_{11}) + MatrixMultiply(\frac{n}{2}, A_{22}, B_{21})$$

$$C_{22} = MatrixMultiply(\frac{n}{2}, A_{21}, B_{12}) + MatrixMultiply(\frac{n}{2}, A_{22}, B_{22})$$

$Divide$：$\Theta(1)$

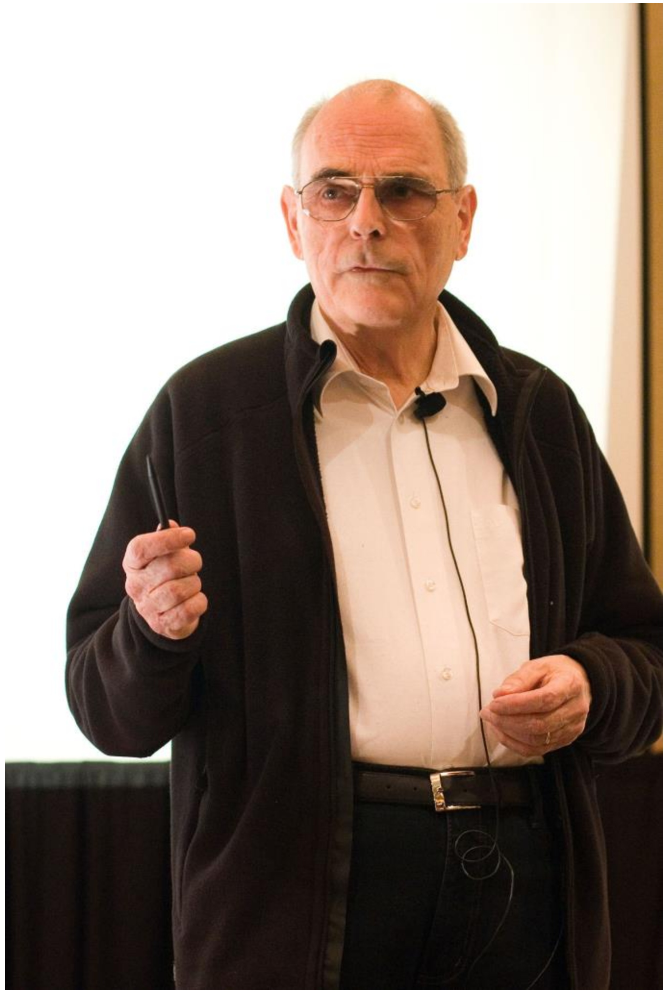
$Conquer$：$T(n) = 8T(n/2)$

$Combine$：$4\Theta\left((n/2)^2\right) = \Theta(n^2)$

$$T(n) = \begin{cases} O(1), if\ n = 1 \\ 8T(\frac{n}{2}) + \Theta(n^2), if\ n \geq 2 \end{cases}$$

$T(n) = \Theta(n^3)$,沒有比較好

矩陣相乘



- Strassen algorithm
  - ➢ 減少遞迴的呼叫次數
    - ✓ 8次→7次
  - ➢ Ex：
    1. ac + ad + bc +bd
       - ✓ ×: 4、+: 3
    2. (a+b)(c+d)
       - ✓ ×: 1、+: 2

矩陣相乘

$$C = A \times B$$
$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{12} + A_{11})B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

矩陣相乘

$$C = A \times B$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22} = M_3 + M_5$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21} = M_2 + M_4$$

$$C_{22} = A_{21}B_{11} + A_{22}B_{21} = M_1 - M_2 + M_3 + M_6$$

矩陣相乘

$Strassen(n, A, B)$

if (n == 1)

return AB;

$$\left.\begin{aligned}
M_1 &= Strassen(\frac{n}{2}, (A_{11} + A_{22}), (B_{11} + B_{22})) \\
M_2 &= Strassen(\frac{n}{2}, (A_{21} + A_{22}), B_{11}) \\
M_3 &= Strassen(\frac{n}{2}, A_{11}, (B_{12} - B_{22})) \\
M_4 &= Strassen(\frac{n}{2}, A_{22}, (B_{21} - B_{11})) \\
M_5 &= Strassen(\frac{n}{2}, (A_{12} + A_{11}), B_{22}) \\
M_6 &= Strassen(\frac{n}{2}, (A_{21} - A_{11}), (B_{11} + B_{12})) \\
M_7 &= Strassen(\frac{n}{2}, (A_{12} - A_{22}), (B_{21} + B_{22})) \\
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{21} &= M_1 - M_2 + M_3 + M_6
\end{aligned}\right\}$$

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$$

$$\mathbf{T(n) = O\left(n^{\log_2 7}\right) \sim O\left(n^{2.807}\right)}$$

矩陣相乘

- *Strassen* 演算法
  - 減少遞迴的呼叫次數
    - ✓ 8次→7次
  - 運算次數：
    1. 暴力解：$A \times n^3$
    2. *Strassen* 演算法：$B \times n^{log_2 7}$
  - 缺點
    1. 因 B > A，所以適用於矩陣較大的狀況
    2. 容易有溢位或浮點數運算誤差的問題
    3. 占用多餘空間

選擇問題

- 任意傳入一陣列
- 隨意指定常數 k，$1 \le k \le$ 陣列長度
- 回傳第 k 大的數字

- k=1, output = 74
- k=2, output = 68
- k=3, output = 52
- k=4, output = 52
- k=5, output = 52
- k=6, output = 47
- k=7, output = 36
- k=8, output = 35
- k=9, output = 27
- k=10, output = 12

35	52	68	12	47	52	36	52	74	27

選擇問題

- 想法1：先排序再取
  - $O(nlog_2n)$
- 選擇問題 ≤ 排序問題
- 有沒有更好的？

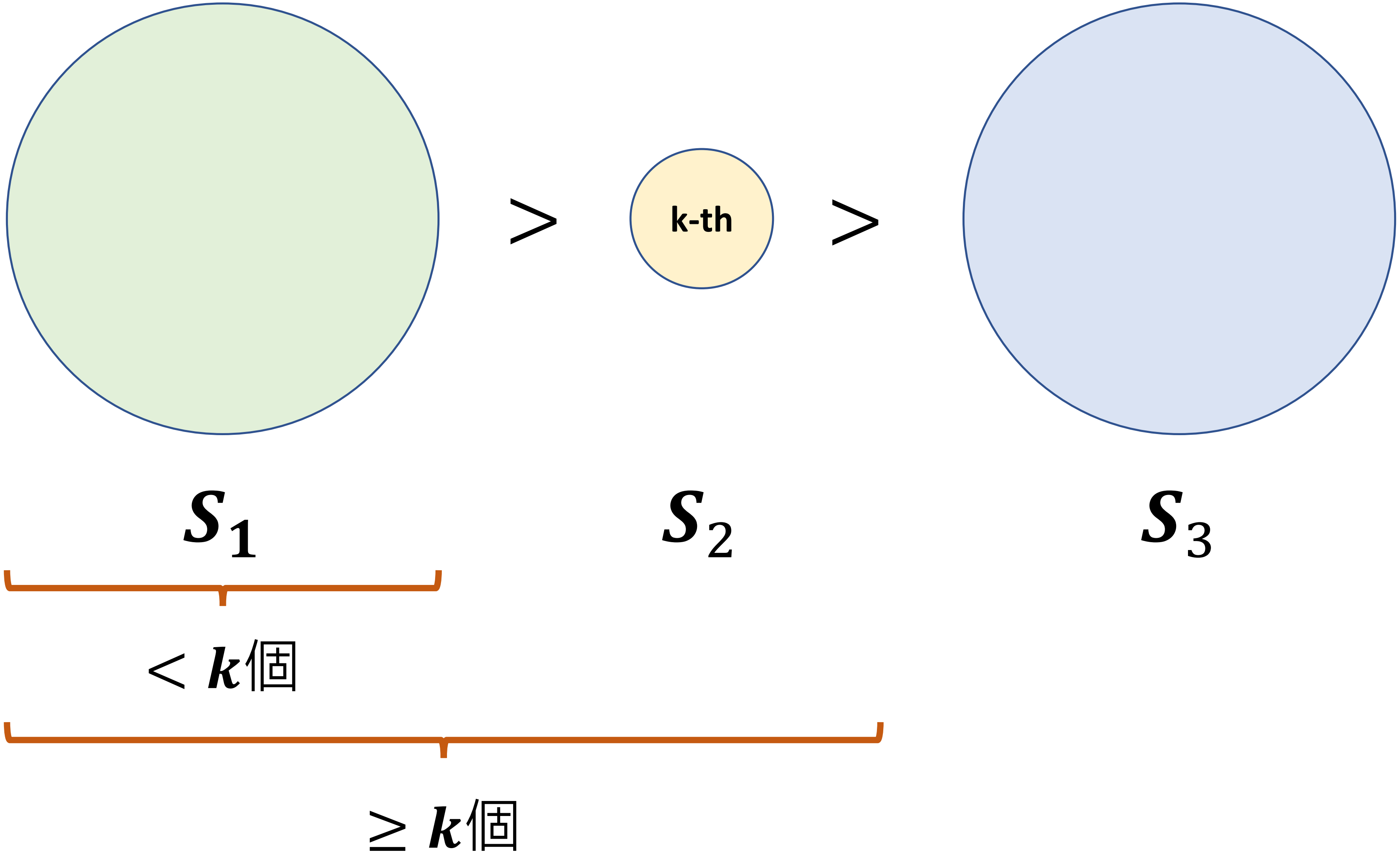
- k=1, output = 74
- k=2, output = 68
- k=3, output = 52
- k=4, output = 52
- k=5, output = 52
- k=6, output = 47
- k=7, output = 36
- k=8, output = 35
- k=9, output = 27
- k=10, output = 12

35	52	68	12	47	52	36	52	74	27

選擇問題

- Prune and Search
  - 給定一集合 $S = \{s_1, s_2, s_3, \ldots\ldots, s_n\}$，找出第 k 大的元素
  - 基本精神：
    1. 挑出一個元素 a (pivot)
    2. 利用 a 把集合 S 區分成
       a. 大於 a 的集合：$S_1 = \{s_i | s_i > a, 1 \le i \le n\}$
       b. 等於 a 的集合：$S_2 = \{s_i | s_i = a, 1 \le i \le n\}$
       c. 小於 a 的集合：$S_3 = \{s_i | s_i < a, 1 \le i \le n\}$
    3. 分成三種狀況：
       a. if：$|S_1| \ge k$，目標在 S_1
       b. $else\ if$：$|S_1| + |S_2| \ge k$，目標在 S_2，a 就是答案
       c. $else$：目標在 S_3

選擇問題

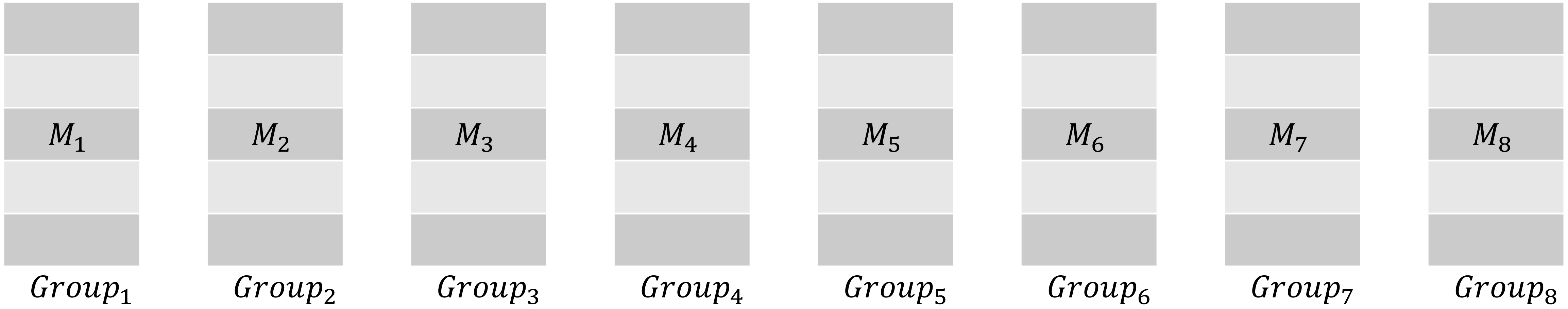


選擇問題

- 如何挑選 pivot？
  - ➢ 好的 pivot 的特色
    - ✓ 每次都能夠刪減固定比例的資料
  - ➢ 理想中的 pivot
    - ✓ 每次都能夠把資料平均分成三個子集合
    - ✓ 理想很豐滿，現實很骨感，How？

選擇問題

- Prune and Search
  - ➢ 把資料分成五個五個一組
    - ✓ 不足的部分補上∞
  - ➢ 分別取出每一組五個元素中的中位數
  - ➢ 再取出所有組中的中位數的中位數
    - ✓ Median of Medain (MoM)



選擇問題

- Median of Medain (MoM) 是所有資料的中位數嗎？
  - ➢ 並不是！
  - ➢ 為何不直接取所有資料的中位數當 pivot？
    - ✓ 算不出來呀 QQ

$Group_1$	$Group_2$	$Group_3$	$Group_4$	$Group_5$	$Group_6$	$Group_7$
1	1	1	1	6	6	6
2	2	2	2	7	7	7
3	3	3	3	8	8	8
4	4	4	4	9	9	9
5	5	5	5	10	10	10

選擇問題

- Median of Medain (MoM) 是所有資料的中位數嗎？
  - ➢ MOM 至少會 ≥ 30% 的資料
  - ➢ MOM 至少會 ≤ 30% 的資料
  - ➢ 會以 30%、70%的方式切出

1	1	1	1	6	6	6
2	2	2	2	7	7	7
3	3	3	3	8	8	8
4	4	4	4	9	9	9
5	5	5	5	10	10	10
$Group_1$	$Group_2$	$Group_3$	$Group_4$	$Group_5$	$Group_6$	$Group_7$

選擇問題

- 如何挑選 pivot？

1. 將 S 分成 $\left\lceil\frac{n}{5}\right\rceil$ 組資料，每組有 5 筆資料：O(n)
   - ➢ 不足 5 個以 ∞ 補足。
2. 排序每一組組內的五筆資料：O(1)
3. 找出所有組別內的中位數：O(1)
4. 重複步驟1~3，取出這些組別中位數的中位數：T(n/5)
5. 把資料集合 S 分成 S_1、S_2、S_3：O(n)
6. 分成三種狀況加以判斷：T(7/10n)
   a. if：$|S_1| \geq k$，目標在 S_1
   b. $else\ if$：$|S_1| + |S_2| \geq k$，目標在 S_2，a 就是答案
   c. $else$：目標在 S_3

選擇問題

$$T(1) = \Theta(1)$$

$$T(n) \le T\left(\left\lfloor\frac{n}{5}\right\rfloor\right) + T\left(\left\lfloor\frac{7n}{10}\right\rfloor\right) + \Theta(n)$$

注意 n = 2~4，undefined？

$$T(n) = O(n)$$

Homework！

Master theorem

複雜度計算 (Review)

- **數學解法 (Mathematics-based Method)**
  - ➢ **直接以遞迴的觀念算出複雜度**

$$T(n) = \begin{cases} T(n-1)+3, if\ n>1 \\ 1, otherwise \end{cases}$$

$$\begin{aligned}
&T(n) \\
&= T(n-1)+3 \\
&= T(n-2)+3+3 \\
&= T(n-3)+3+3+3 \\
&= \ldots\ldots \\
&= T(1)+3(n-1) \\
&= 3n-2 \\
&\underline{T(n) \in O(n)}
\end{aligned}$$

複雜度計算 (Review)

- 代換法 (Substitution Method)
  - 猜一個數字後帶入

$$T(n) = \begin{cases} T(n-1)+3, if\ n>1 \\ 1, otherwise \end{cases}$$

猜 $T(n) \in O(n)$

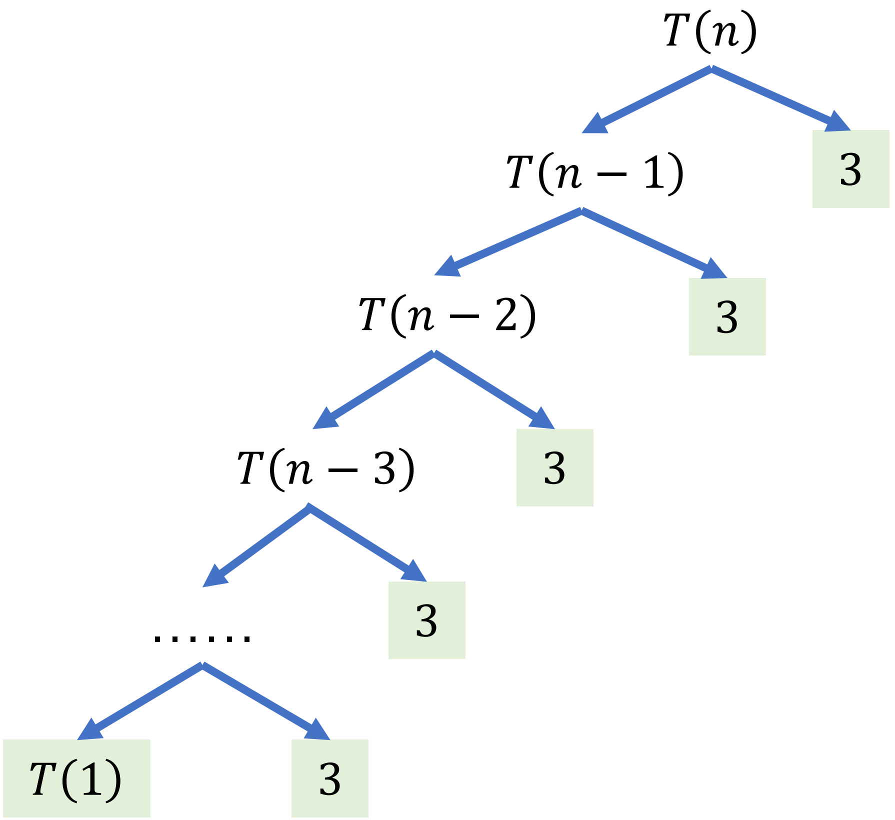
$T(n) \le c(n-1)+3$
$T(n) \le cn-c+3$
$\underline{T(n) \in O(n)}$

複雜度計算 (Review)

- 遞迴樹法 (Recurrence Tree Method)
  - ➢ 畫出遞迴樹後加總之

$$T(n) = \begin{cases} T(n-1)+3, if\ n>1 \\ 1, otherwise \end{cases}$$

$$T(n)$$
$$= T(1) + 3 \times (n-1)$$
$$= 3n - 2$$
$$\underline{T(n) \in O(n)}$$



支配理論

- **支配理論 (Master Method)**
  - ➢ **運用公式解**

$$
\begin{aligned}
&T(n) \\
&= aT\left(\frac{n}{b}\right) + f(n);\ a, b \geq 1 \\
&= a[aT\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)] + f(n) \\
&= a^2T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right)\ + f(n) \\
&= \cdots\ (let\ n = b^i) \\
&= a^iT\left(\frac{n}{b^i}\right) + a^{i-1}f\left(\frac{n}{b^{i-1}}\right) + a^{i-2}f\left(\frac{n}{b^{i-2}}\right) + \cdots + f(n)
\end{aligned}
$$

支配理論

- 支配理論方法 (Master Method)
  - 運用公式解

$$n = b^i$$
$$i = log_b\, n$$

$$T(n)$$
$$= a^i T\left(\frac{n}{b^i}\right) + a^{i-1} f\left(\frac{n}{b^{i-1}}\right) + a^{i-2} f\left(\frac{n}{b^{i-2}}\right) + \cdots + f(n)$$
$$= a^{\log_b n} T(1) + a^{i-1} f\left(\frac{n}{b^{i-1}}\right) + a^{i-2} f\left(\frac{n}{b^{i-2}}\right) + \cdots + f(n)$$
$$= n^{\log_b a} T(1) + a^{i-1} f\left(\frac{n}{b^{i-1}}\right) + a^{i-2} f\left(\frac{n}{b^{i-2}}\right) + \cdots + f(n)$$

$$a^{\log_b n} = n^{\log_b a}$$
$$\log_b\left(a^{\log_b n}\right) = \log_b\left(n^{\log_b a}\right)$$
$$\log_b n \log_b a = \log_b a \log_b n$$

支配理論

$$T(n) = n^{\log_b a} T(1) + a^{i-1} f\left(\frac{n}{b^{i-1}}\right) + a^{i-2} f\left(\frac{n}{b^{i-2}}\right) + \cdots + f(n)$$

讓 $n^{\log_b a}$ 跟 $f(n)$ 比大小

1. $f(n) = O\left(n^{\log_b (a-\varepsilon)}\right), \varepsilon > 0 \rightarrow n^{\log_b a}$ 比 $f(n)$ 大

$$T(n) = \Theta(n^{\log_b a})$$

2. $f(n) = \Omega\left(n^{\log_b (a+\varepsilon)}\right), \varepsilon > 0 \rightarrow f(n)$ 比 $n^{\log_b a}$ 大

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

3. $f(n) = \Theta(n^{\log_b a}) \rightarrow n^{\log_b a}$ 跟 $f(n)$ 一樣大

$$T(n) = \Theta(n^{\log_b a} \times log_b f(n))$$

矩陣相乘

$$Strassen(n, A, B)$$

$if\,(n == 1)$

$\quad return\ AB;$

$$\left.\begin{aligned}
M_1 &= Strassen(\frac{n}{2}, (A_{11} + A_{22}), (B_{11} + B_{22})) \\
M_2 &= Strassen(\frac{n}{2}, (A_{21} + A_{22}), B_{11}) \\
M_3 &= Strassen(\frac{n}{2}, A_{11}, (B_{12} - B_{22})) \\
M_4 &= Strassen(\frac{n}{2}, A_{22}, (B_{21} - B_{11})) \\
M_5 &= Strassen(\frac{n}{2}, (A_{12} + A_{11}), B_{22}) \\
M_6 &= Strassen(\frac{n}{2}, (A_{21} - A_{11}), (B_{11} + B_{12})) \\
M_7 &= Strassen(\frac{n}{2}, (A_{12} - A_{22}), (B_{21} + B_{22})) \\
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{21} &= M_1 - M_2 + M_3 + M_6
\end{aligned}\right\}$$

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$$

讓 $n^{\log_2 7}$ 跟 n^2 比大小

$$n^{\log_2 7} > n^2$$

$$T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.807})$$

$$\boldsymbol{T(n) = O\left(n^{\log_2 7}\right) \sim O(n^{2.807})}$$

Practice 4

Mission

請利用支配理論計算合併排序的複雜度。

合併排序

```
void Merge_Sort(int data[],int start,int finish){
    if(finish>start){
        int middle = (finish+start)/ 2;
        Merge_Sort(data, start, middle);
        Merge_Sort(data, middle+1, finish);
        Merge(data,start,finish,middle);
    }
}
```

Divide&Conquer

Combine

$$T(n) = \begin{cases} O(1), if\ n = 1 \\ T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + O(n), if\ n \geq 2 \end{cases}$$

合併排序

$$T(n)=\begin{cases} O(1), if\ n=1 \\ 2T(\frac{n}{2})+O(n), if\ n\geq 2 \end{cases}$$

$$T(n)$$

$$\leq 2T\left(\frac{n}{2}\right)+cn$$

$$\leq 2\left[2T\left(\frac{n}{4}\right)+c\frac{n}{2}\right]+cn=4T\left(\frac{n}{4}\right)+2cn$$

$$\leq 4\left[2T\left(\frac{n}{8}\right)+c\frac{n}{4}\right]+2cn=8T\left(\frac{n}{8}\right)+3cn$$

…….

$$\leq 2^kT\left(\frac{n}{2^k}\right)+kcn, let\ k=log_2n$$

$$T(n)\leq nT(1)+cnlog_2n=O(n)+O(nlog_2n)$$

$$T(n)=O(nlog_2n)$$

合併排序

$$T(n) = \begin{cases} O(1), if\ n = 1 \\ 2T(\frac{n}{2}) + O(n), if\ n \geq 2 \end{cases}$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$a = 2, b = 2, f(n) = n$$

讓 $n^{\log_2 2}$ 跟 $f(n)$ 比大小

$$n^{\log_2 2} = n = f(n)$$

$$T(n) = \Theta(n^{\log_b a} \times \log_b f(n))$$

$$T(n) = \Theta(n^{\log_2 2} \times \log_2 n) = \Theta(n \times \log_2 n)$$

實戰練習

Practice

Mission

Try LeetCode #169. Majority Element

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

- Example 1:
  - Input: nums = [3,2,3]
  - Output: 3

Ref : https://leetcode.com/problems/majority-element/

Practice

Mission

Try LeetCode #240. Search a 2D Matrix II

Write an efficient algorithm that searches for a target value in an m x n integer matrix. The matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Ref：https://leetcode.com/problems/search-a-2d-matrix-ii/