分治 Divide and Conquer

Lecture & Modified by Colten
Credit by baluteshih, yp155136, TreapKing, Gino

分治的本質

- 相信大家看完影片後，對分治稍微有點感覺了。
  - 感覺好像把問題切一切，然後合併一下，就可以解決問題了。

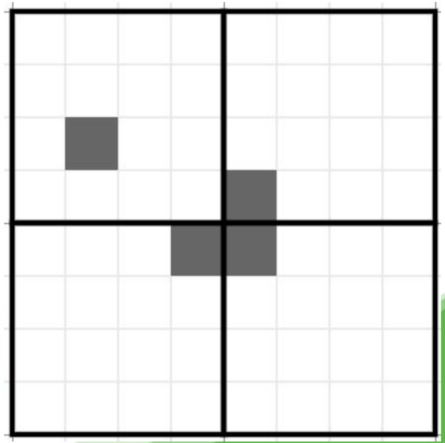
- 但你真的知道分治的原理嗎？
  - 為什麼可以這樣切
  - 為什麼這樣分治之後，複雜度會很神奇地降低（例：合併排序）

分治的本質

- 分治，分而治之。
- 分治不是一個特定的演算法，而是設計演算法的一種方法。
- 分治法的框架具有以下三部分：
  - 分（Divide）：大問題切割成多個小問題
  - 邊界條件：問題已經分割、簡化到可以直接算答案
  - 治（Conquer）：合併小問題的答案，得到大問題的解
- 怎麼知道什麼時候該使用分治呢？

適合分治的問題 - 1

- 有些問題具有某種特別的性質，並且在切割問題後，子問題也都具有該性質。
- 這類問題天生就適合用分治解決。
  - L形方塊問題具有的性質：「棋盤恰好缺一個格子」。
  - 從中間切成四等分，在中間放上一塊 L 形方塊。
  - 則四個規模為 $N/2$ 的子問題
  - 也具有「恰好缺一個格子」這個性質。



適合分治的問題 - 2

- 有些問題則反過來，先從規模較小的問題出發，利用小問題建構出更大規模問題的解。
- 這種方法一般稱為「倍增法」。
  - 等差數列的問題（課前影片）
  - 上週的手寫作業

適合分治的問題 - 3

- 有些問題則是可以利用分治來加速。
  - 解決問題可能有很多要知道的「資訊」。
  - 一個一個檢查那些「資訊」太耗時。
  - 利用分治，有可能會冒出一些很有用的性質，
  - 幫助我們預先知道一些資訊，
  - 這樣便可減少檢查資訊的時間，從而得到更快速的解法。
- 例子：合併排序、快速排序

排序再談

- 排序問題：將一個長度 N 的序列由小排到大。

排序再談

- 排序問題：將一個長度 N 的序列由小排到大。
- 一個序列具有 $C(N, 2)$ 個相異數對 (a_i, a_j)。

排序再談

- **排序問題：將一個長度 N 的序列由小排到大。**
- **一個序列具有 $C(N, 2)$ 個相異數對 (a_i, a_j)。**
- 如果想排好序列，你必須知道所有 (a_i, a_j) 之間的大小關係。
  - 也就是知道 $a_i > a_j$、$a_i = a_j$ 還是 $a_i < a_j$

排序再談

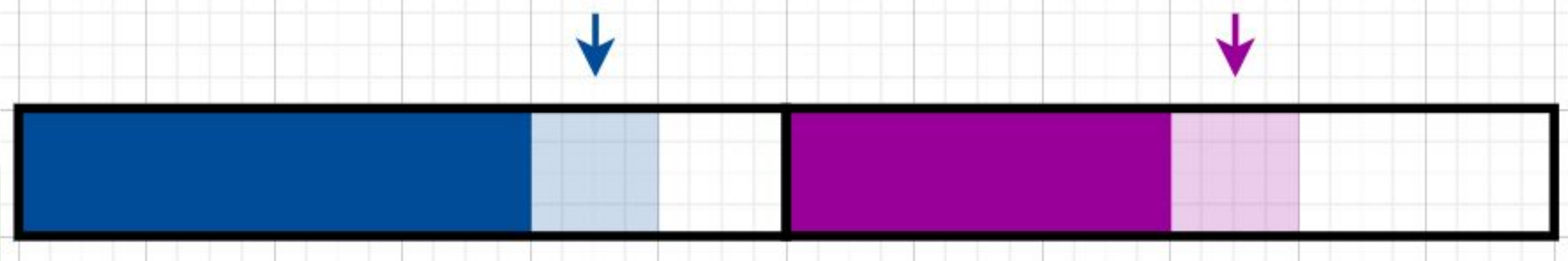
- **排序問題：將一個長度 N 的序列由小排到大。**
- **一個序列具有 $C(N, 2)$ 個相異數對 (a_i, a_j)。**
- 如果想排好序列，你必須知道所有 (a_i, a_j) 之間的大小關係。
  - 也就是知道 $a_i > a_j$、$a_i = a_j$ 還是 $a_i < a_j$
- 泡沫排序、插入排序、選擇排序複雜度是 $O(N^2)$
  - 這些演算法都需要一個一個慢慢檢查所有數對

排序再談

- **排序問題：將一個長度 N 的序列由小排到大。**
- **一個序列具有 $C(N,2)$** 個相異數對 (a_i, a_j)。
- 如果想排好序列，你必須知道所有 (a_i, a_j) 之間的大小關係。
  - 也就是知道 $a_i > a_j$、$a_i = a_j$ 還是 $a_i < a_j$
- 泡沫排序、插入排序、選擇排序複雜度是 $O(N^2)$
  - 這些演算法都需要一個一個慢慢檢查所有數對
- 那合併排序到底快在哪裡？

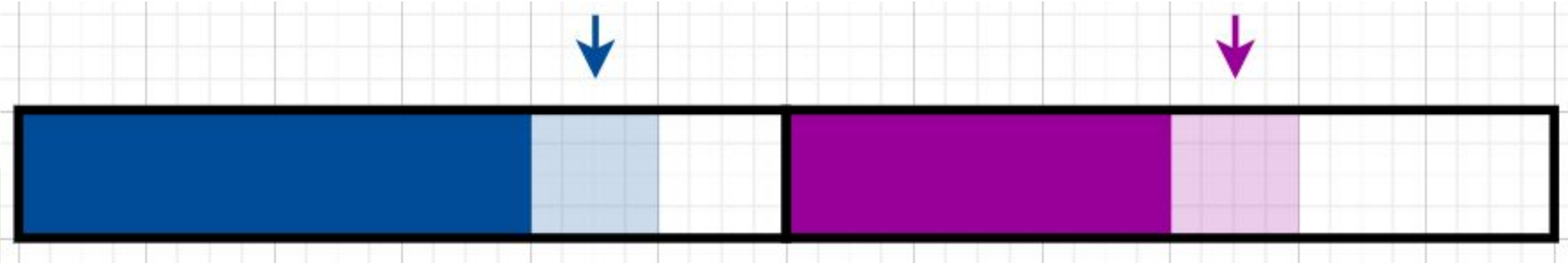
排序再談

- 觀察一下合併排序中，合併兩半邊的過程：



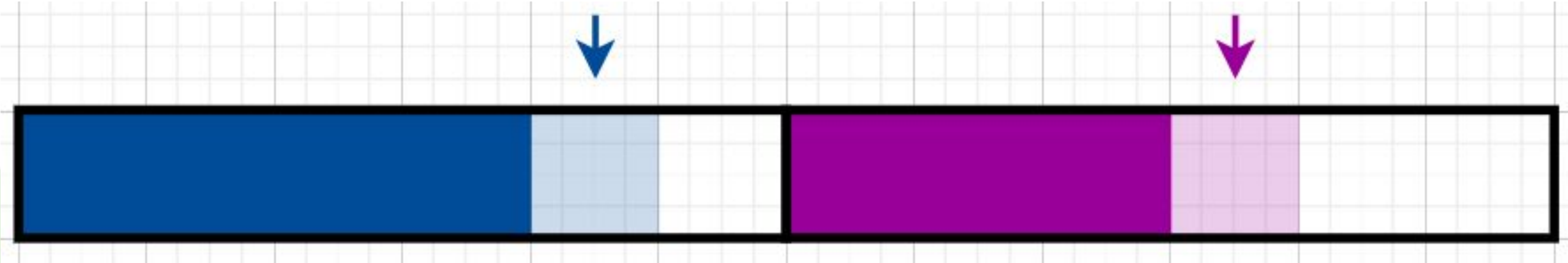
排序再談

- 觀察一下合併排序中，合併兩半邊的過程：
- 深色：已經排序好的部分／半透明：正要比大小的數字



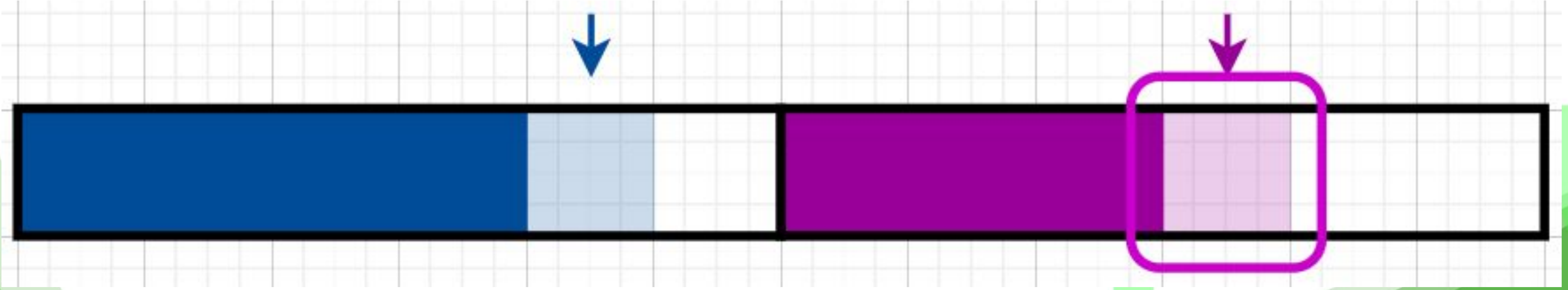
排序再談

- 觀察一下合併排序中，合併兩半邊的過程：
- 深色：已經排序好的部分／半透明：正要比大小的數字
- 比較兩邊箭頭的大小，把比較小的拿掉，放到新陣列
- 並把箭頭往後移動



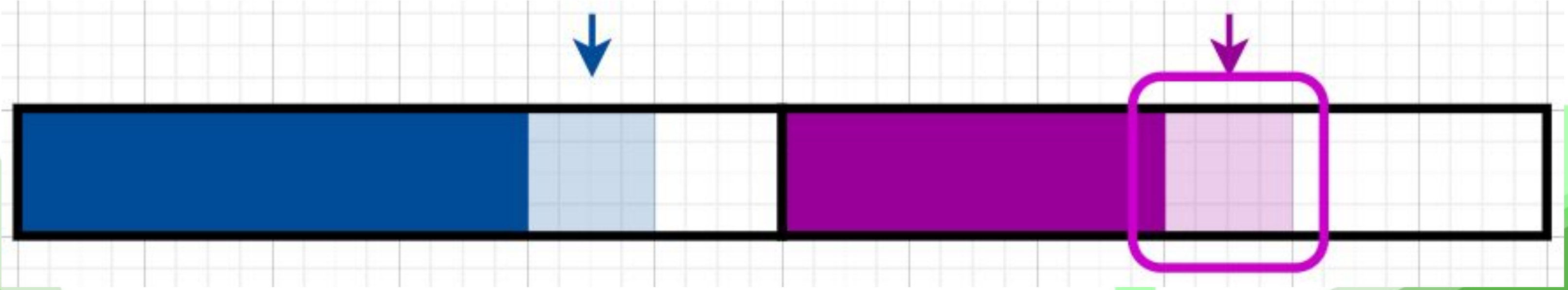
排序再談

- 假設右邊的比較小，那便可以直接把它拿下來，放進新陣列。



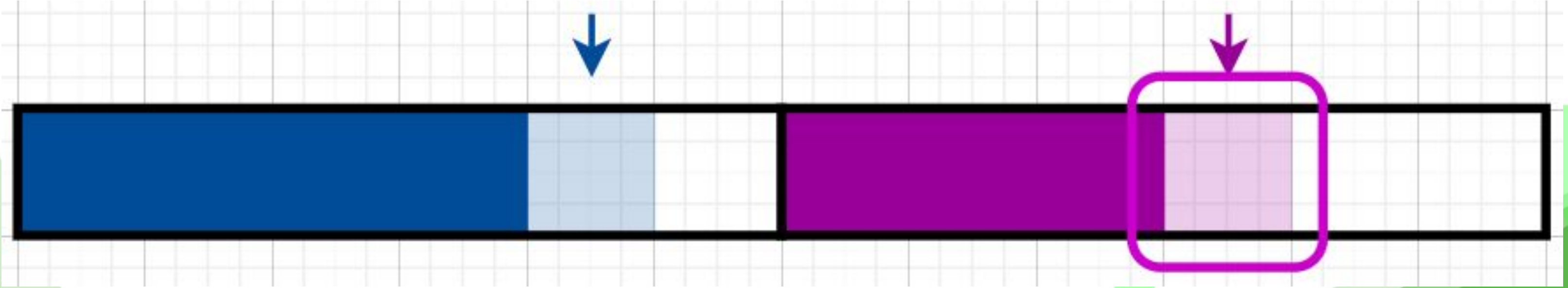
排序再談

- 假設右邊的比較小，那便可以直接把它拿下來，放進新陣列。
- 為什麼可以保證它比「先前已經放好的數字」都還要大？



排序再談

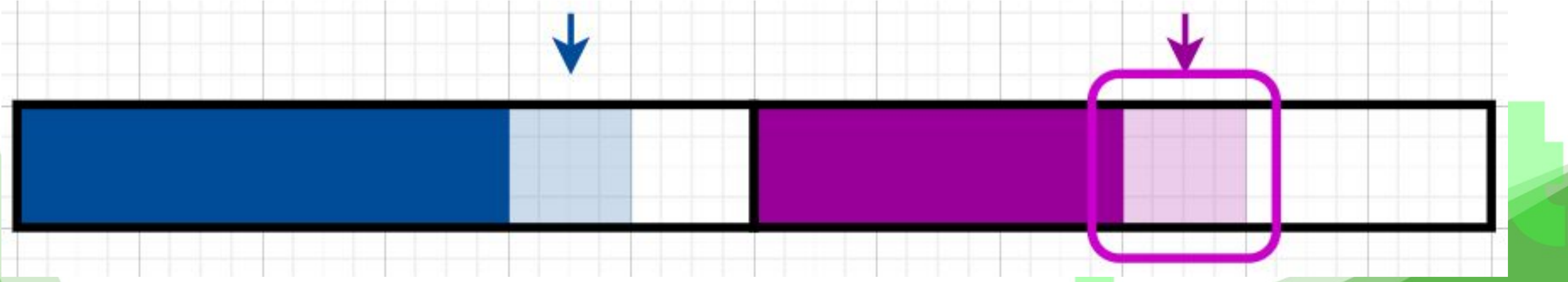
- 為什麼可以保證它比「先前已經放好的數字」都還要大？
- 對於**右半邊深粉色**的部分：
  - 我們已經遞迴將左右兩半邊都排序好。
  - 因此**放進去的這個數字**自然比**深粉色**的部分都還要大。



排序再談

- 為什麼可以保證它比「先前已經放好的數字」都還要大？
- 對於**左半邊深藍色**的部分：
  - 如果**深藍色**存在一個數字 k，比我**要放的數字**還大。
  - 那 k > **要放的數字** > **深粉色**，也就代表我先前的操作有誤，

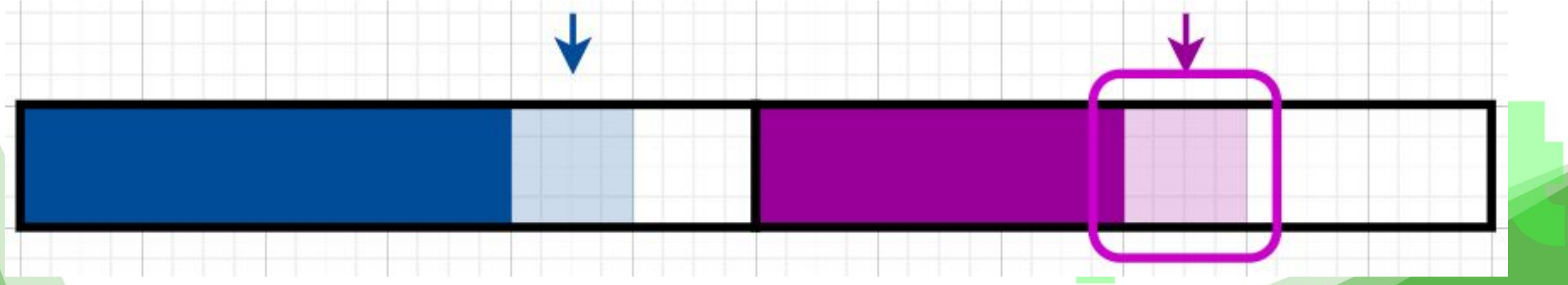
    不小心先放了比較大的數字。



排序再談

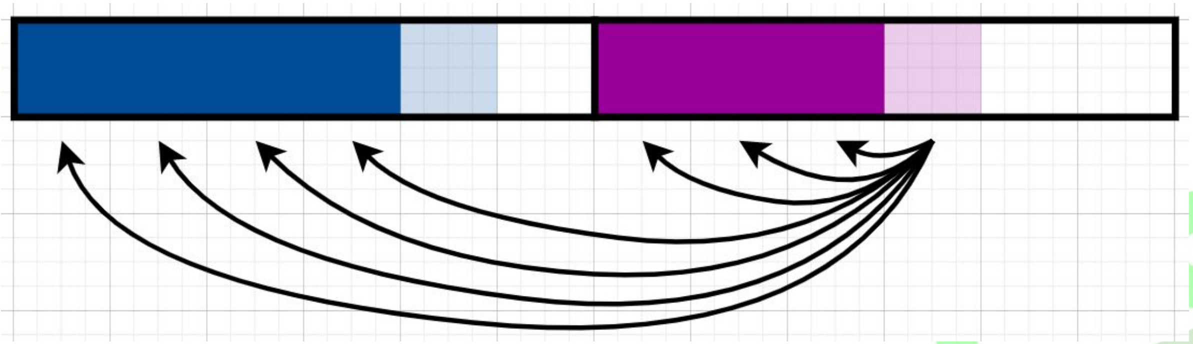
- 為什麼可以保證它比「先前已經放好的數字」都還要大？
- 對於**左半邊深藍色**的部分：
  - 如果**深藍色**存在一個數字 k，比我**要放的數字**還大。
  - 那 $k >$ **要放的數字** $>$ **深粉色**，也就代表我先前的操作有誤，

    不小心先放了比較大的數字。
  - 矛盾，因此**深藍色**的部分一定比我**要放的數字**還小。



排序再談

- 利用分治，便可以製造**單調性**（左右兩半邊都由小到大遞增）。
- 利用單調性，我們可以不需要檢查大部分的數對，
- 就可以知道該怎麼排好數字。
- 如下圖，黑色箭頭是我們「不需要檢查的數對」的其中一部份。
- 我們成功地加速了排序的過程，這便是分治的強大之處。



重點整理

- 分治的使用時機:
- **1. 利用分治建構答案:**
  - 問題本身帶有一些特殊性質，使得切割子問題後，會發現子問題也都具有該性質（例:L 形方塊)。
  - 或是可以從規模較小的問題拼湊出原問題的解（例:倍增)。
- **2. 利用分治加速演算法:**
  - 要求出一個問題的解可能要知道很多資訊。
  - 利用分治，可以製造一些好用的性質（例:單調性)，從而減少大量不必要枚舉的資訊。

符號定義 - $T(n)$

- 分治演算法的時間複雜度習慣以 $T(n)$ 表示。
  - n 表示輸入量大小。
- 既然是分治，那就表示 $T(n)$ 是一個遞迴函數。
- 以 Merge Sort 為例：
  - 每次會將問題切割成 2 個規模為 $n/2$ 的子問題。
  - 每次都要花費 $O(n)$ 的時間合併。
  - 故 $T(n) = 2T(n/2) + O(n)$
- 以快速冪為例：
  - $T(n) = T(n/2) + O(1)$

(Credit to NTU Algorithm Design and Analysis, 2019 Fall)

分治複雜度分析

- 有時候設計好了一個分治演算法，但我們不確定這個算法夠不夠快，想知道具體複雜度是多少。
- 而分治通常又沒辦法很直覺地看出複雜度是多少。
  - 你能一眼看出 $T(n) = T(n/5) + T(7n/10) + O(n)$ 的複雜度嗎？

(Credit to NTU Algorithm Design and Analysis, 2019 Fall)

分治複雜度分析

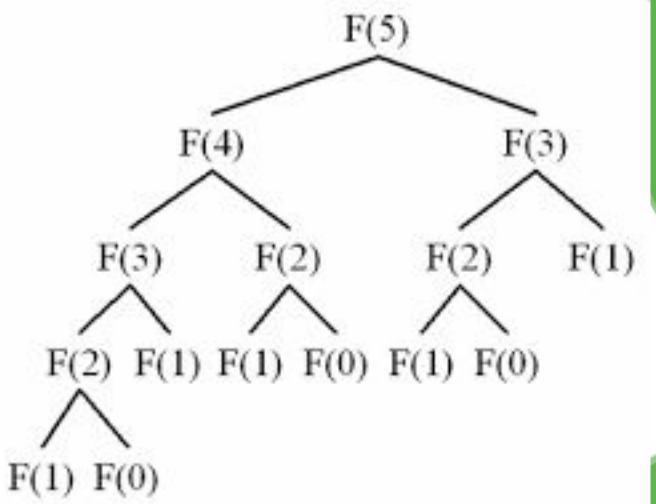
- 有時候設計好了一個分治演算法，但我們不確定這個算法夠不夠快，想知道具體複雜度是多少。
- 而分治通常又沒辦法很直覺地看出複雜度是多少。
  - 你能一眼看出 $T(n) = T(n/5) + T(7n/10) + O(n)$ 的複雜度嗎？
- 這時就必須仰賴一些工具。
- 以下將介紹一些分析分治／遞迴複雜度常用的工具：
  - Recursion-Tree Method (遞迴樹法)
  - Substitution Method (取代法)
  - Master Method (主定理)

(Credit to NTU Algorithm Design and Analysis, 2019 Fall)

Recursion-Tree Method

- 把遞迴過程畫成一棵樹狀圖，稱為**遞迴樹**。
- 接著把每層的時間複雜度加總即可。

▼（費式數列遞迴過程）

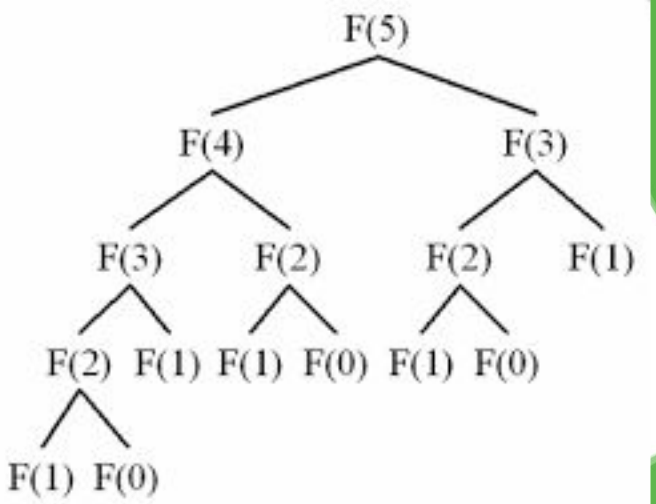


reference:http://kkc47.blogspot.com/2014/08/learning-dynamic-programming-day-1.html

Recursion-Tree Method

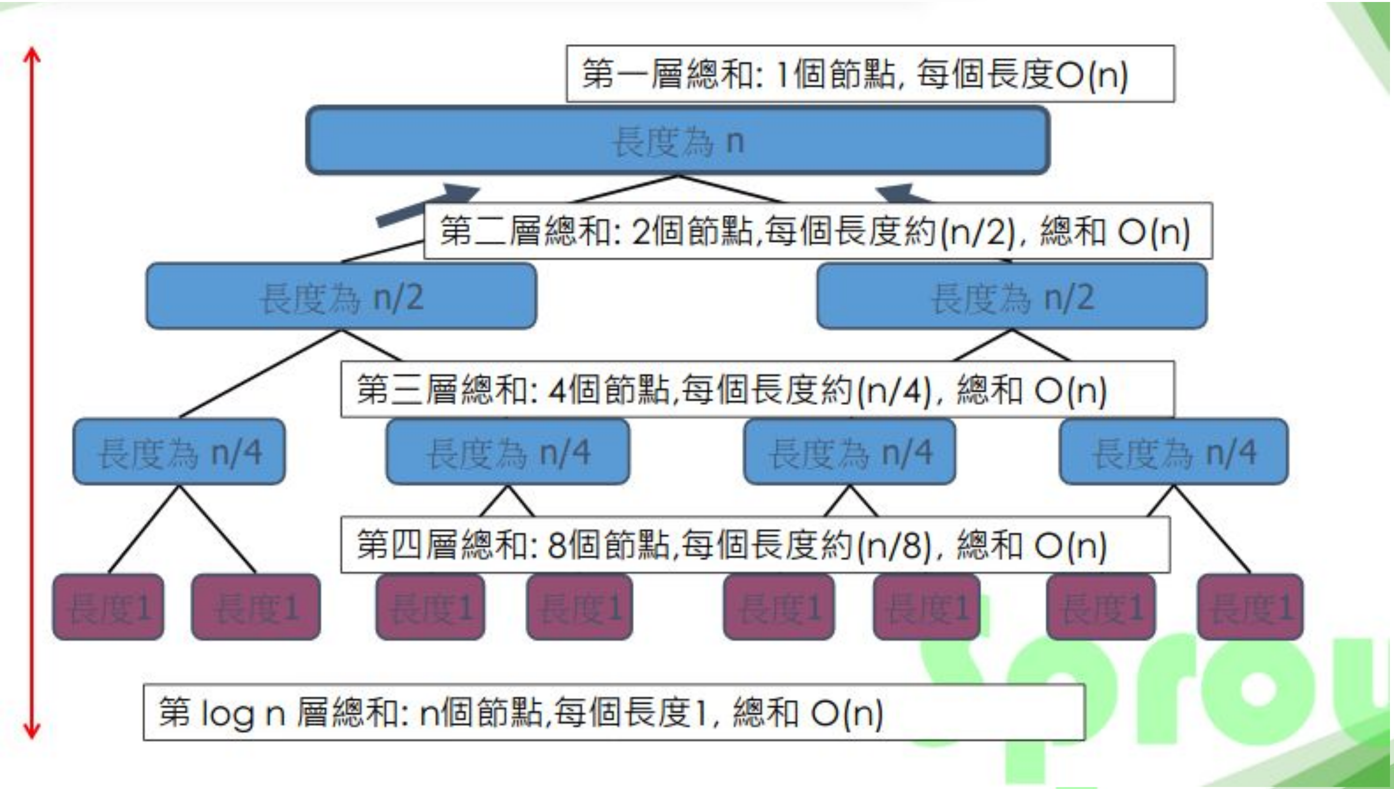
- 把遞迴過程畫成一棵樹狀圖，稱為**遞迴樹**。
- 接著把每層的時間複雜度加總即可。
- 遞迴樹法非常直觀，但並不嚴謹。
- 他只能幫你「猜」複雜度。
- 要嚴謹證明的話可以用等等會介紹的 substitution method。

▼(費式數列遞迴過程)



reference:http://kkc47.blogspot.com/2014/08/learning-dynamic-programming-day-1.html

用遞迴樹分析 Merge Sort



Recursion-Tree Method

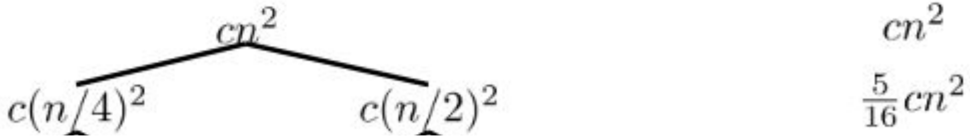
- 來看個例子
- 例：

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + cn^2$$

- 試試看把遞迴樹畫出來

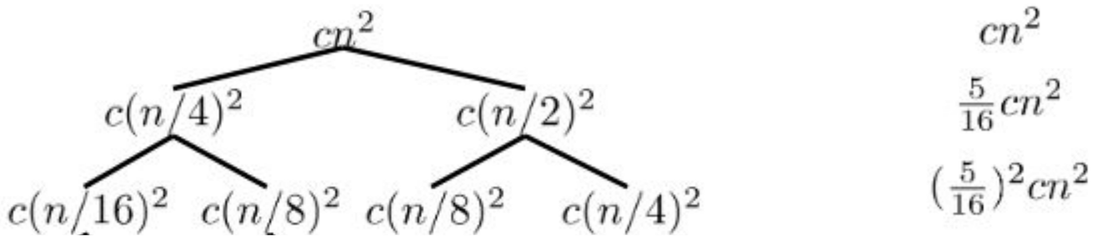
Recursion-Tree Method

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



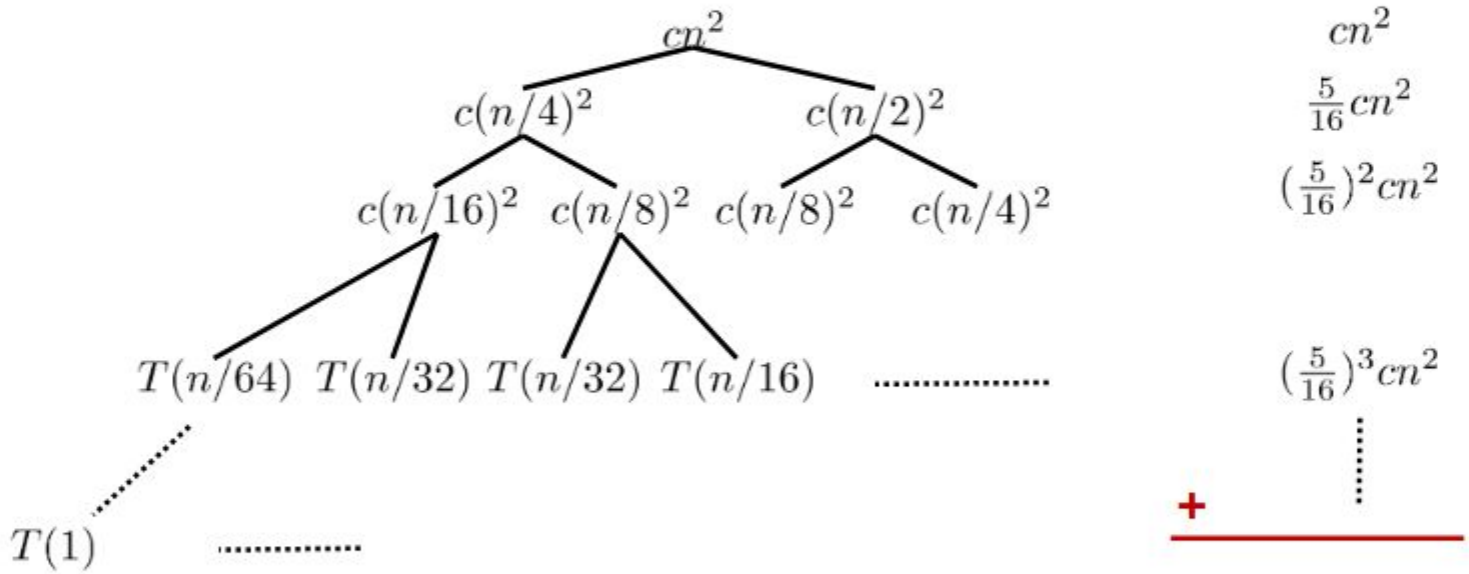
Recursion-Tree Method

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



Recursion-Tree Method

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



Recursion-Tree Method

- 例：$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + cn^2$
- 可以得到

$$T(n) \leq (1 + \frac{5}{16} + (\frac{5}{16})^2 + \ldots)cn^2$$

Sprout

Recursion-Tree Method

- 例：$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + cn^2$
- 可以得到 $T(n) \leq (1 + \frac{5}{16} + (\frac{5}{16})^2 + \ldots)cn^2$
- 利用等比級數公式，可得

$$T(n) \leq \frac{1}{1-\frac{5}{16}}cn^2 = \frac{16}{11}cn^2$$

- 因此，$T(n)$ 是 $O(n^2)$
- 但這不足以證明 $T(n)$ 是 $O(n^2)$
- 真的要分析的話，可以利用接下來要介紹的「取代法」

Substitution Method 先備知識

- 還記得複雜度分析的手寫作業嗎？
- Big-O 定義:

$$f(n) \in O(g(n)) \Leftrightarrow \exists c, n_0$$
$$\text{such that } 0 \leq f(x) \leq cg(n)\ \forall n \geq n_0$$

- 存在常數 c, n_0, 使得當 n 超過臨界點 n_0 時, $f(n)$ 都不會超過 $cg(n)$。
- Big-O 描述的是演算法複雜度的上界。
- 例:當 $f(n) = x^2 + 10x$, $g(n) = x^3$, 選擇 $c = 0.5, n_0 = 10$,
  便可證明 $f(n) \in O(g(n))$

Sprout

Substitution Method 先備知識

- **Big-O 描述的是演算法複雜度的上界。**
  - $f(n) \in O(g(n))$ 代表 $f(n)$ 的量級不會超過 $g(n)$。
- **Big-Ω 描述的是演算法複雜度的下界。**
  - $f(n) \in \Omega(g(n))$ 代表 $f(n)$ 的量級至少比 $g(n)$ 還大。
- **Big-Θ 描述的是演算法複雜度的量級（剛剛好）。**
  - $f(n) \in \Theta(g(n))$ 代表 $f(n)$ 的量級跟 $g(n)$ 相同。

Substitution Method

- 取代法
- 三大步驟:**Guess、Verify、Solve**
- **Guess**:
  - 猜 $T(n)$ 是屬於哪個複雜度。
  - 沒有一個標準的方法,只能靠經驗,或是利用遞迴樹先畫圖猜
- **Verify**:
  - 驗證猜想是否正確。
  - 幾乎都用 **數學歸納法** 驗證(取代法的名稱由來)。
- **Solve**:
  - 找到合適的常數 c, n_0、證明 big-O

Substitution Method

- 範例:

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

- 三大步驟:**Guess、Verify、Solve**

Substitution Method

- 範例：

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

- 先來做個簡單的化簡，把 big-O 給去掉

$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(\frac{n}{2}) + bn & \text{if } n \geq 2 \end{cases}$$

- 其中 $a, b > 0$

Sprout

Substitution Method

- 範例: $$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(\frac{n}{2}) + bn & \text{if } n \geq 2 \end{cases}$$
- 猜測: $$T(n) \leq b \times n \log n + an$$

Substitution Method

- 猜測 $T(n) \leq b \times n \log n + an$
- 使用數學歸納法
  - $n = 1$: trivial

Substitution Method

- 猜測 $T(n) \le b \times n \log n + an$
- 使用數學歸納法
  - $n = 1$: trivial
  - $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。

$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(\frac{n}{2}) + bn & \text{if } n \ge 2 \end{cases}$$

Substitution Method

- 猜測 $T(n) \le b \times n \log n + an$
- 使用數學歸納法
  - $n = 1$: trivial
  - $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。
  - 則: $T(n) = 2T(\frac{n}{2}) + bn$

$$\le 2(b \times \frac{n}{2} \log \frac{n}{2} + a\frac{n}{2}) + bn$$

歸納法假設

$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(\frac{n}{2}) + bn & \text{if } n \ge 2 \end{cases}$$

Substitution Method

- 猜測 $T(n) \le b \times n \log n + an$
- 使用數學歸納法
  - $n = 1$: trivial
  - $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。
  - 則: $T(n) = 2T(\frac{n}{2}) + bn$

$$
\begin{aligned}
&\le 2(b \times \tfrac{n}{2}\log\tfrac{n}{2} + a\tfrac{n}{2}) + bn \\
&= bn\log n \underline{- bn\log 2} + an + \underline{bn} \\
&= bn\log n + an
\end{aligned}
$$

歸納法假設

對數律

相等可消掉 (注意 log 是以 2 為底)

Substitution Method

- 猜測 $T(n) \le b \times n \log n + an$
- 使用數學歸納法
  - $n = 1$: trivial
  - $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。
  - 則: $T(n) = 2T(\frac{n}{2}) + bn$

$$\le 2(b \times \frac{n}{2} \log \frac{n}{2} + a\frac{n}{2}) + bn$$

$$= bn \log n - \underline{bn \log 2} + an + \underline{bn}$$

$$= bn \log n + an$$

歸納法假設

對數律

相等可消掉 (注意 log 是以 2 為底)

- 不論 a, b 為何, $T(n)$ 皆滿足猜測。
- 因此由數學歸納法, 可以得到 $T(n) \in O(n \log n)$

Substitution Method

- 剛剛的例子似乎不需要特別找 c, n_0 就證明完成了。
- 其實只是運氣特別好(?)
- 讓我們來看更多例子吧！

Substitution Method

- 範例:
$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{if } n \geq 2 \end{cases}$$

- 試證明: $T(n) \in O(n^3)$

- Hint:

$$T(n) \in O(n^3) \Rightarrow \exists c, n_0, \text{s.t.} \forall n \geq n_0, T(n) \leq cn^3$$

Sprout

Substitution Method

- 使用數學歸納法證明 $T(n) \in O(n^3)$
- $n = 1$: trivial
- $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{if } n \geq 2 \end{cases}$$

Substitution Method

- 使用數學歸納法證明 $T(n) \in O(n^3)$
- $n = 1$: trivial
- $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。
- 則: $T(n) \le 4T(\frac{n}{2}) + bn$

$$\le 4(c(\frac{n}{2})^3) + bn = \frac{cn^3}{2} + bn$$

$$= cn^3 - (\frac{cn^3}{2} - bn)$$

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{if } n \ge 2 \end{cases}$$

Substitution Method

- 使用數學歸納法證明 $T(n) \in O(n^3)$
- $n = 1$: trivial
- $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。
- 則: $T(n) \le 4T(\frac{n}{2}) + bn$

$$\le 4(c(\frac{n}{2})^3) + bn = \frac{cn^3}{2} + bn$$

$$= cn^3 - (\frac{cn^3}{2} - bn)$$

$$\le cn^3$$

為了順利讓 $T(n) \le cn^3$, 就得讓 $cn^3/2 - bn \ge 0$

我們可以把 c 設定為 $2b$, 把 n_0 設定為 1

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{if } n \ge 2 \end{cases}$$

Substitution Method

- 使用數學歸納法證明 $T(n) \in O(n^3)$
- $n = 1$: trivial
- $n > 1$: 假設對所有 $k < n$, $T(k)$ 皆符合猜測。
- 則: $T(n) \leq 4T(\frac{n}{2}) + bn$

$$\leq 4(c(\frac{n}{2})^3) + bn = \frac{cn^3}{2} + bn$$

$$= cn^3 - (\frac{cn^3}{2} - bn)$$

$$\leq cn^3$$

為了順利讓 $T(n) \leq cn^3$, 就得讓 $cn^3/2 - bn \geq 0$

我們可以把 c 設定為 $2b$, 把 n_0 設定為 1

- 所以, 當 $c = 2b, n_0 = 1$ 時, $T(n) \leq cn^3$, 也就是 $T(n) \in O(n^3)$

Substitution Method

- 剛才介紹的 Substitution Method 如果搞不太懂為什麼要找常數、為什麼可以那樣設，沒關係。
  - 複雜度的定義與證明都是利用**極限**來分析。
  - 因為我們考慮的是 n 的規模趨近無限大的狀況。
  - 高三或大一的微積分課會學到更多有關**極限**的定義、性質。
  - 到時候再回來翻這些內容會變得非常好懂。
- 接下來要介紹一個更常用、更強大的工具 – 主定理。
- 主定理是前人歸納一些常見遞迴的複雜度，整理成一套好用的公式。

Master Theorem

- 假設 $T(n) = aT(\frac{n}{b}) + f(n)\ (a \geq 1, b > 1)$
- Case 1: $\exists \epsilon > 0$ s.t. $f(n) = O(n^{\log_b(a)-\epsilon})$

  $\Rightarrow T(n) = \Theta(n^{\log_b a})$
- Case 2: $\exists \epsilon \geq 0$ s.t. $f(n) = \Theta(n^{\log_b a} \log^{\epsilon} n)$

  $\Rightarrow T(n) = \Theta(n^{\log_b a} \log^{\epsilon+1} n)$
- Case 3: $\exists \epsilon > 0$ s.t. $f(n) = \Omega(n^{\log_b(a)+\epsilon})$

  且 $\exists 0 < c < 1$ s.t. $af(\frac{n}{b}) \leq cf(n)$ 在 n 足夠大時

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

Master Theorem

- 白話一點，就是比較 $f(n)$ 跟 $n^{\log_b a}$ 的關係。
- 如果一樣的話，就加個 log，否則就是取比較大的那個。
- 簡易證明可參考

  ：https://mycollegenotebook.medium.com/%E6%99%82%E9%96%93%E8%A4%87%E9%9B%9C%E5%BA%A6-%E9%81%9E%E8%BF%B4-%E4%B8%8B-master-th-307ad4608ab6
- 以下會分三種情況說明，順便會舉點例子
  - 小於
  - 等於
  - 大於

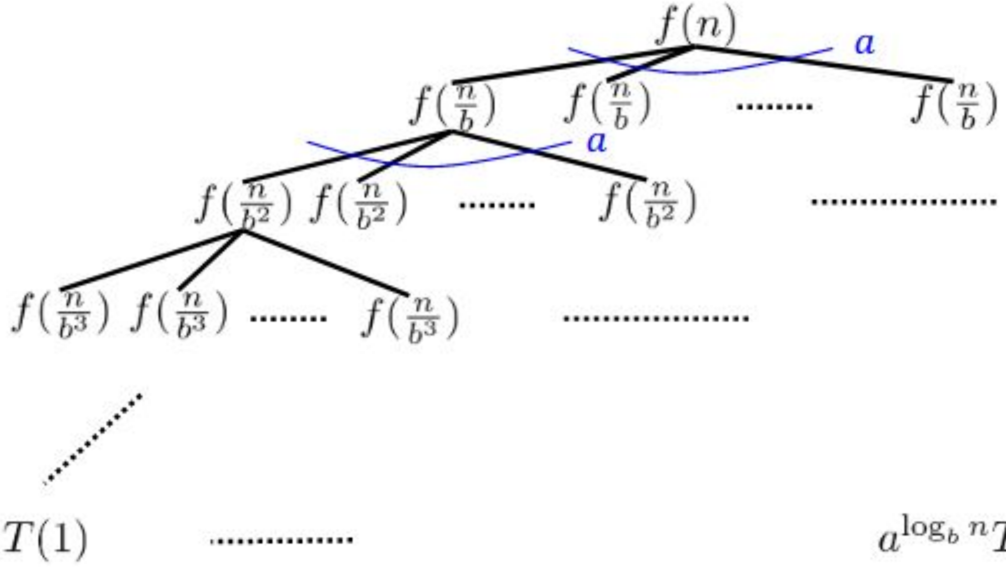
Master Theorem

- 比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果 $f(n) < n^{\log_b a}$，那麼 $T(n) = O(n^{\log_b a})$
- 例如，$T(n) = 9T(\frac{n}{3}) + n$
- 根據主定理，$n < n^{\log_3 9}$

  所以 $T(n) \in O(n^2)$
- 可以試著用 Substitution method 寫寫看
- 也可以試著畫畫看 Recursion tree

Master Theorem

$$T(n) = 9T(\tfrac{n}{3}) + n, T(1) = 1$$



$$f(n) = n$$

$$af(\tfrac{n}{b}) = \tfrac{9}{3}n$$

$$a^2 f(\tfrac{n}{b^2}) = (\tfrac{9}{3})^2 n$$

$$a^3 f(\tfrac{n}{b^3}) = (\tfrac{9}{3})^3 n$$

$$\vdots$$

$$a^{\log_b n} T(1) = 9^{\log_3 n} = (\tfrac{9}{3})^{\log_3 n} n$$

$f(n)$ grows polynomially slower than $n^{\log_b a}$

Master Theorem

- 比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果 $f(n) = n^{\log_b a}$（量級相同），那麼

$$T(n) = O(f(n)\log n) = O(n^{\log_b a}\log n)$$

- 例如， $T(n) = 2T(\frac{n}{2}) + O(n)$

- $T(n) = O(n \log n)$，前面也有用 substitution 證明過
- 可以想像成原本的複雜度多一個 log
- 在這個 case，$f(n)$ 如果多乘好幾個 log 也沒關係，依然可以直接多補一個上去

Master Theorem

- 比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果 $f(n) > n^{\log_b a}$，那麼

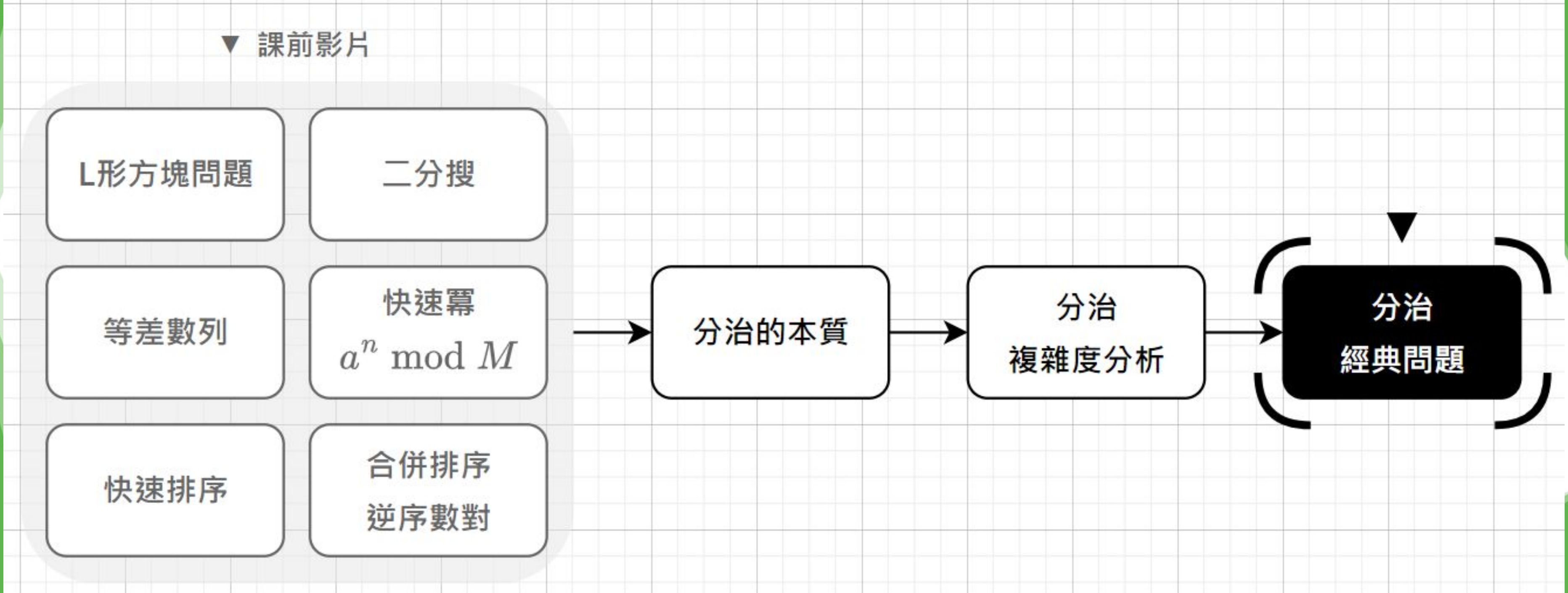
$$T(n) = O(f(n))$$

- 例如，$T(n) = T(\frac{n}{2}) + O(n^2)$
- $T(n) \in O(n^2)$，可以畫畫看 Recursion Tree 驗證
- 這個 case 其實有個額外的條件，但是不常發生(?)

分治複雜度分析

- 重點整理:
- **Recursion Tree:**
  - 直接把遞迴樹畫出來然後加總每個節點的時間
  - 直觀好用, 但不夠嚴謹, 需搭配 Substitution Method 證明
- **Substitution Method:**
  - 先猜測一個複雜度 (猜 $T(n) \leq$ 某個式子)
  - 利用數學歸納法證明
  - 盲點:就算複雜度是對的, 只要式子猜錯那就有可能證不出來
- **Master Theorem:**
  - 好用, 很香
  - 比較 $f(n)$ 和 $n^{\log_b a}$ 的大小關係
  - 碰到子問題規模不同時就沒辦法用 (例: $T(n) = T(n/5) + T(7n/10) + O(n)$)

學習地圖



▼ 課前影片
L形方塊問題
二分搜
等差數列
快速冪
$a^n \bmod M$
快速排序
合併排序
逆序數對
分治的本質
分治
複雜度分析
分治
經典問題

分治經典問題

- 接下來要介紹的一些經典問題。
- 這些問題都是利用分治來加速演算法、優化複雜度。

- 最大連續和
- 平面最近點對
- 序列第 k 大

最大連續和

- 給你一個長度為 N 的序列 a_1, a_2, ..., a_N，請你找到 (L, R)，滿足 a_L + ... + a_R 最大
- N <= 10^5
- 先來想想看要怎麼做
- 練習題：https://zerojudge.tw/ShowProblem?problemid=d784

最大連續和

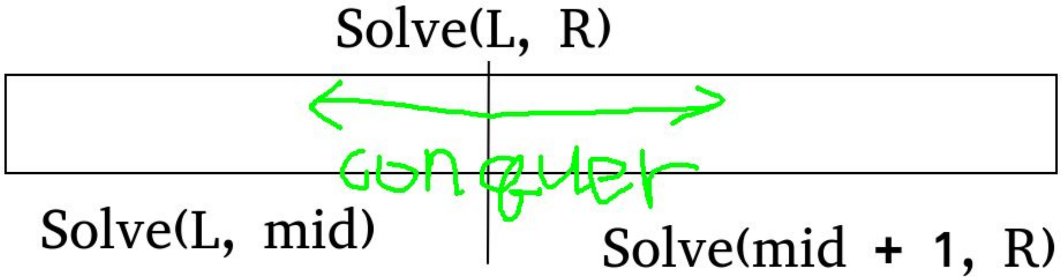
- O(N^3)
- O(N^2)
- O(N log N)
- O(N) (?)
- 那換一個限制，只能用「分治法」來做
  - 因為我們現在在教分治嘛

最大連續和

- 答案有 O(N^2) 種
- 我們有沒有辦法好好的 divide 成一半，然後想盡辦法 conquer 起來呢？

最大連續和

- 是可以的！
- 我們先把序列切成兩半，左右遞迴求出各自半部的最佳解



  - Solve(L, R) 會回傳 a_L, ..., a_R 的最佳解

最大連續和 --- conquer

- 跨過兩側，一定會拿 a[mid] 跟 a[mid + 1]
- 所以我們要求的東西 a[L] + ... + a[R]，可以換成找 (a[L] + ... + a[mid]) + (a[mid + 1] + ... + a[R])

最大連續和 --- conquer

- 跨過兩側，一定會拿 a[mid] 跟 a[mid + 1]
- 所以我們要求的東西 a[L] + ... + a[R]，可以換成找 (a[L] + ... + a[mid]) + (a[mid + 1] + ... + a[R])
- 有沒有發現，上面的式子就是從中間點開始，往左 & 往右的走一段路後，得到的總和
- 於是乎，取「從中間往左加的最大值」加上「從中間往右加的最大值」，最後加起來就完成 conquer 的部份了
- 簡單的前後綴最大值，O(n) 就做得到

最大連續和

```
int solve(int L, int R) {
    if (L == R) {
        return a[L];
    }
    int mid = (L + R) >> 1;
    int ans = max(solve(L, mid), solve(mid + 1, R));
    int lmax = a[mid], lpre = a[mid];
    for (int i = mid - 1; i >= L; --i) {
        lpre += a[i];
        lmax = max(lmax, lpre);
    }
    int rmax = a[mid + 1], rpre = a[mid + 1];
    for (int i = mid + 2; i <= R; ++i) {
        rpre += a[i];
        rmax = max(rmax, rpre);
    }
    return max(ans, lmax + rmax);
}
```

最大連續和

- 複雜度分析

最大連續和

- 複雜度分析
- 假設 T(n) 是 solve() 的長度為 n 的複雜度
- 那麼, $T(n) = 2T(n / 2) + O(n)$

最大連續和

- 複雜度分析
- 假設 T(n) 是 solve() 的長度為 n 的複雜度
- 那麼, $T(n) = 2T(n / 2) + O(n)$
- 所以, 最後的複雜度是 $T(n) = O(n \log n)$

最大連續和

- 複雜度分析
- 假設 T(n) 是 solve() 的長度為 n 的複雜度
- 那麼, T(n) = 2T(n / 2) + O(n)
- 所以, 最後的複雜度是 T(n) = O(n log n)
- Challenge:用分治法可以做到 O(n) 嗎?
  - 能不能做到 T(n) = 2T(n / 2) + O(1)

平面最近點對

- 在平面上給你 N 個點，要你找出歐式距離最短的兩個點。
- N <= 100,000

$$dis(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

平面最近點對

- O(N^2) 當然不是我們要的
- 如果在平面上想要做 divide and conquer，該怎麼分割問題？

平面最近點對

- 先把所有輸入的點照 x 座標排序後，在中間畫一條分隔線。
  - 分隔線切在 x 的中位數，這樣可以保證遞迴的時候點的數量都會少一半。
- 這樣可能的答案就分成(兩個點都在左邊)、(兩個點都在右邊)、(橫跨分隔線) 三種情況。
- 一些平面上的 D&C 題都會做類似的事。
- 顯然只有點對「橫跨分隔線」的情況需要討論，如果這個情況可以解決的話，其他兩個情況只要遞迴下去解就好了。

平面最近點對

- 如果只是要算分隔線兩邊的最近點對，有什麼好方法嗎？

平面最近點對

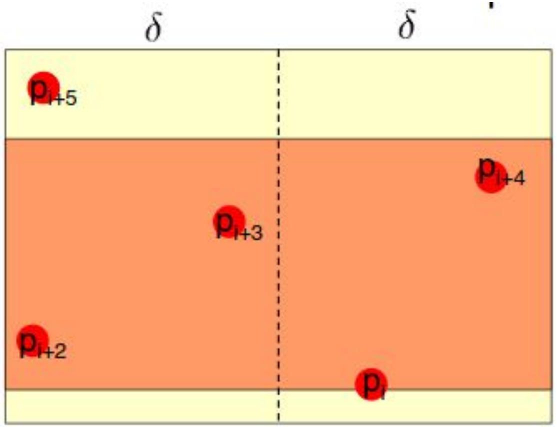
- 如果只是要算分隔線兩邊的最近點對，有什麼好方法嗎？
- 因為 x 座標的大小關係已經確立了，所以可以把兩邊直接照 y 座標排序
- 然後就發現還是好困難……

平面最近點對

- 定神一想，會發現如果遞迴下去後找到的最近點對的距離是 d ，那麼我們根本就不需要考慮那些距離超過 d 的點對
- 所以離分隔線超過 d 的點都不需要去考慮。
- 對於每個點，也只有 y 座標差距不超過 d 的點可能可以讓你找到更近的點對
- 這樣子複雜度會是好的嗎？
- 聽起來只是個壓常數的剪枝，但在什麼情況下，這個做法一樣會退化成 O(N^2) 呢？

平面最近點對

- 因為已經知道各自的最近點對的距離是 d ，所以每個點附近的點不會太多!



- 附近的點只會有常數個，所以直接跑下去複雜度就是好的！

平面最近點對

- 來分析一下複雜度

- $T(n) = 2T(n / 2) + O(n \log n)$
  - conquer 時要把點按照 y 座標排序

- 根據 recursion-tree 和 master theorem，都可以得到 $O(n (\log n)^2)$

- 注意這裡的 master theorem 要套 Case 2

平面最近點對

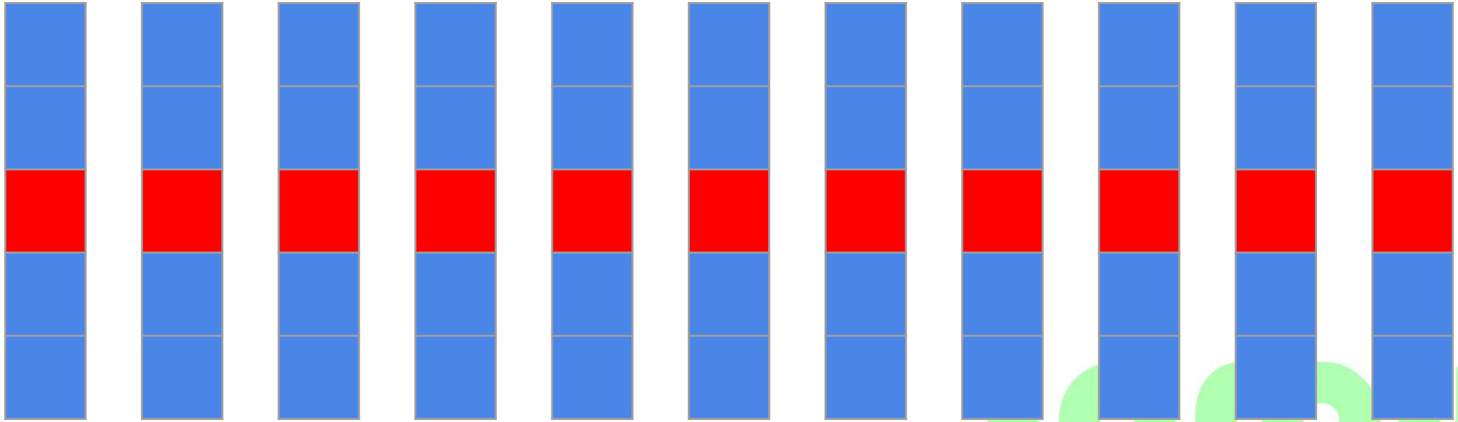
- 不能做到更好嗎？ $O(n\,(\log n)^2)$ 感覺很慢
- 靈光一閃，想起 merge sort
- 後面那個 $O(n\log n)$，可以用 merge sort 壓到 $O(n)$
- 類似逆序數對那樣
- $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) \in O(n\log n)$！
- 這題有非分治的作法，有興趣可以翻去年隨機演算法的投影片：https://www.csie.ntu.edu.tw/~sprout/algo2022/ppt_pdf/week11/random_inclass(hc).pdf

尋找第 k 大

- 給你一個序列，請你找到第 k 大
- 還不簡單？ sort 就好啦！
- 但我希望可以做到 O(n)

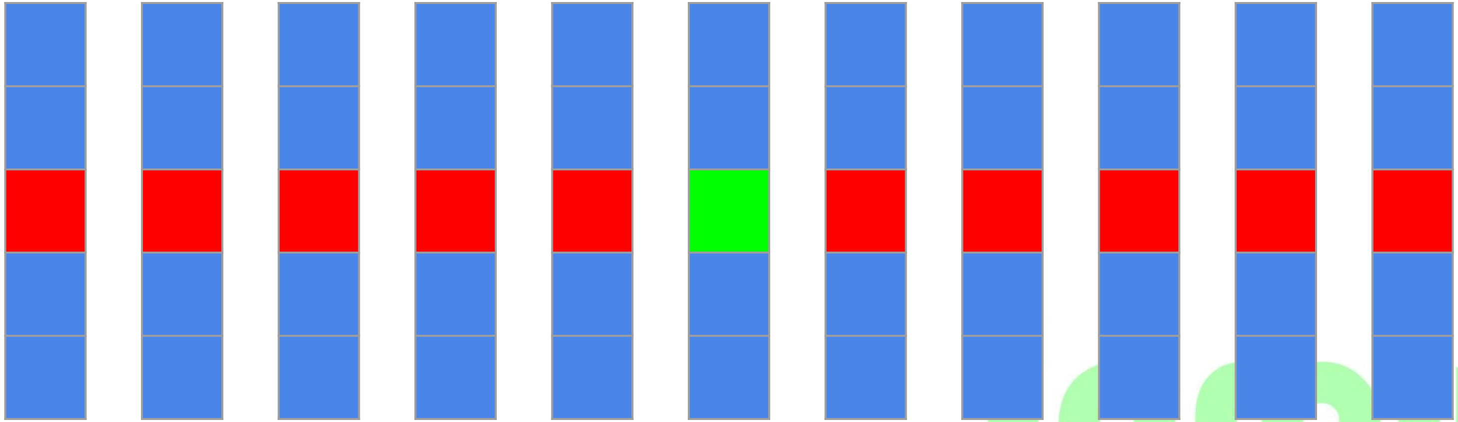
尋找第 k 大

- 神仙分治想法
- 我們首先先把序列五個五個分一組，並找到每組的**中位數**



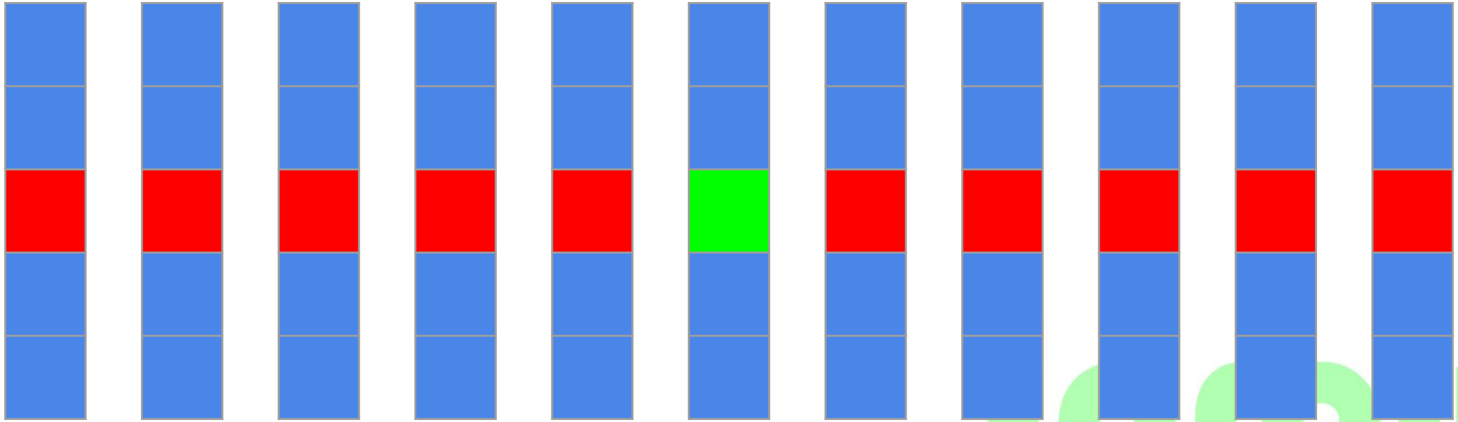
尋找第 k 大

- 把所有中位數蒐集起來，再找到「中位數的中位數」
  - 怎麼再找中位數？



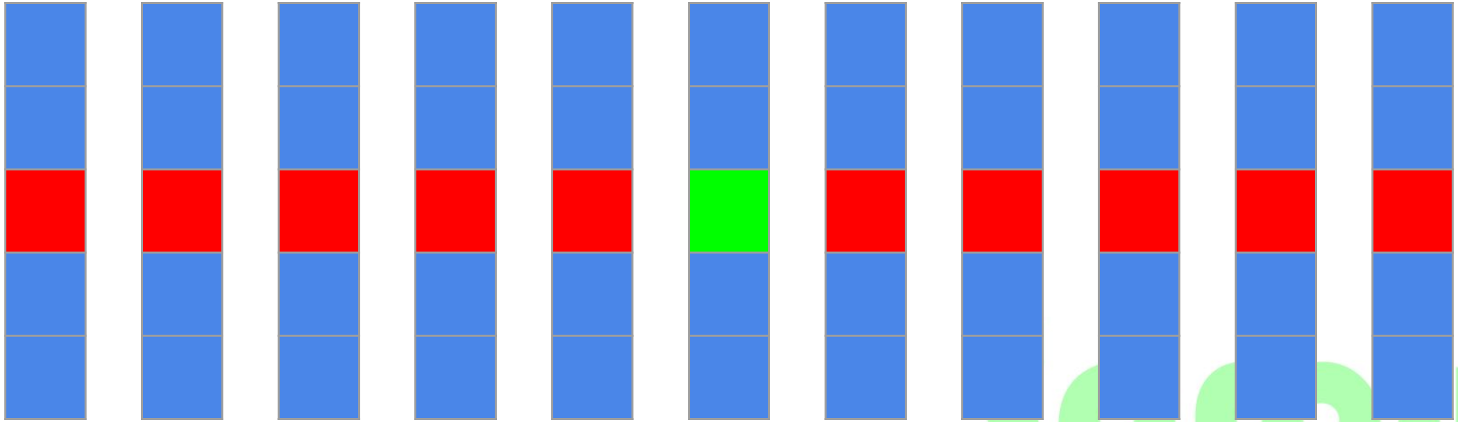
尋找第 k 大

- 把所有中位數蒐集起來，再找到「中位數的中位數」
  - 怎麼再找中位數？**對規模 n/5 的問題呼叫尋找第 n/10 大**



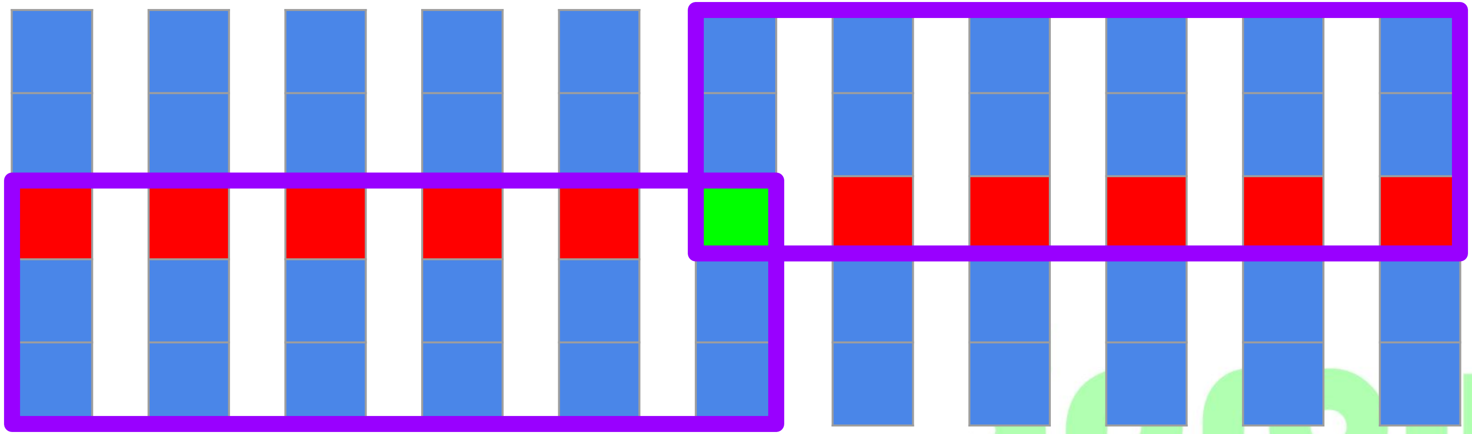
尋找第 k 大

- 令剛剛找到的數字是 p，把數字分成 >p 跟 <p 兩堆



尋找第 k 大

- 令剛剛找到的數字是 p，把數字分成 >p 跟 <p 兩堆
- 注意到至少有 3n/10 個數字比 p 小、至少有 3n/10 個數字比 p 大



尋找第 k 大

- 令剛剛找到的數字是 p，把數字分成 >p 跟 <p 兩堆
- 注意到至少有 3n/10 個數字比 p 小、至少有 3n/10 個數字比 p 大
- 看第 k 大在哪邊，往那邊遞迴就可以了！

- 這種神仙操作到底複雜度長怎樣呢？

尋找第 k 大

- 分析複雜度:
- 對規模 n/5 的問題呼叫尋找第 n/10 大: T(n/5)
- 遞迴找 k 大: 少掉至少 3n/10 個數字 -> T(7n/10)
- 分組、分兩堆等等: O(n)

尋找第 k 大

- 分析複雜度:
- 對規模 n/5 的問題呼叫尋找第 n/10 大: T(n/5)
- 遞迴找 k 大: 少掉至少 3n/10 個數字 -> T(7n/10)
- 分組、分兩堆等等: O(n)

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$$

尋找第 k 大

- 分析複雜度：
- 對規模 n/5 的問題呼叫尋找第 n/10 大：T(n/5)
- 遞迴找 k 大：少掉至少 3n/10 個數字 -> T(7n/10)
- 分組、分兩堆等等：O(n)

$$T(n) = T(\tfrac{n}{5}) + T(\tfrac{7n}{10}) + O(n)$$

$$\Rightarrow T(n) \leq T(\tfrac{n}{5}) + T(\tfrac{7n}{10}) + c \cdot n$$

尋找第 k 大

- 分析複雜度：
- 對規模 n/5 的問題呼叫尋找第 n/10 大：T(n/5)
- 遞迴找 k 大：少掉至少 3n/10 個數字 -> T(7n/10)
- 分組、分兩堆等等：O(n)

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$$

$$\Rightarrow T(n) \le T(\frac{n}{5}) + T(\frac{7n}{10}) + c \cdot n$$

$$\Rightarrow T(n) \le 10 \cdot c \cdot n \in O(n)$$ by 數學歸納法！

尋找第 k 大

- 分析複雜度：
- 對規模 n/5 的問題呼叫尋找第 n/10 大：T(n/5)
- 遞迴找 k 大：少掉至少 3n/10 個數字 -> T(7n/10)
- 分組、分兩堆等等：O(n)

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$$

$$\Rightarrow T(n) \leq T(\frac{n}{5}) + T(\frac{7n}{10}) + c \cdot n$$

$$\Rightarrow T(n) \leq 10 \cdot c \cdot n \in O(n)$$ by 數學歸納法！

- 這東西還蠻詭異的，但真的就是線性XD
- 主要是想讓你們感受一下分治法的強大

分治實作小技巧

- 切割的區間，依照個人習慣可以選擇開區間或閉區間。
  - 我自己是習慣左閉右閉。
- 思考分治題的時候，可以直接假設 Divide 下去左右兩邊遞迴都是好的。
- 這時候只要專心想怎麼 Conquer 就好，
  - 也就是怎麼處理跨越兩半的答案，不用管左右邊遞迴會發生什麼事。
- 遞迴常數頗大，當一個問題 Divide 到規模足夠小的時候，就可以改成使用暴力而不繼續遞迴下去。

More About 分治

- 今天我們提到了：
  - 分治到底是什麼／到底快在哪
  - 分治的複雜度怎麼分析
  - 一些分治的酷酷問題
- 實際上跟分治有關的演算法遠不止這些：
  - CDQ 分治／操作分治
  - FFT (前面教的多項式乘法其實可以用 FFT 做，複雜度更好)
  - 動態規劃的分治優化
  - 平行二分搜
  - 樹重心分治、重心剖分
  - ...
- 今天這堂課算是帶大家對分治有基本的認識
  - 掌握了這些知識，往後學更多分治演算法或是解分治題目會變得更容易

課程結束

大家辛苦了 ＞＜