



分治 Divide and Conquer

Lecture & modified by baluteshih

Credit by yp155136

Credit by TreapKing

Sprout



Q & A

- 影片有什麼問題嗎？

Sprout



何時使用分治

- 經驗！
 - 刷題！

Sprout



何時使用分治

- 經驗！
 - 刷題！
- 面對一個問題時，怎麼枚舉都找不到優化的切入點
- 突發奇想從中間切一半，就可能找到很棒的 conquer 性質
- 當這個性質足夠讓我們在良好的複雜度解決 conquer 時，就可能可以在犧牲頂多一兩個 $\log$ 的情況下完成整個問題
- 「在 conquer 時能夠省去維護儲存所有資訊的力氣，只需要專心處理跨分隔線的資訊」，就是分治的精髓



分治小技巧

- 養成良好習慣
 - 左閉右閉(我個人常用的方式)
 - 左閉右開
 -(?)
- divide 完之後遞迴下去, 直接假設遞迴後得到的結果是理想的, 這樣 conquer 的時候會比較好思考
- 當 n 很小的時候, 可以暴力做, 可能會減少遞迴的 cost
 - EX: merge sort 到 n 很小時, 就隨使用一個 $O(n^2)$ 的 sort
- 多寫題目(?)

Sprout



分析遞迴的工具

- Credit to Algorithm Design and Analysis, 2019 Fall

Sprout



Master Theorem

- 主定理
- 很好用的工具！

Sprout



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

Sprout



Master Theorem

- 白話一點 QQ

Sprout



Master Theorem

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

- 白話一點 QQ
- 不嚴謹的說, 就是比較 $f(n)$ 跟 $O(n^{\log_b a})$ 的關係
- 如果一樣的話, 就加個 $\log$, 否則就是比較大的那個
- 以下會分三種情況說明, 順便會舉點例子
 - 小於
 - 等於
 - 大於

Sprout



Master Theorem

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

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

所以 $T(n)$ 是 $O(n^2)$

- 可以試著把分治的過程畫出來！

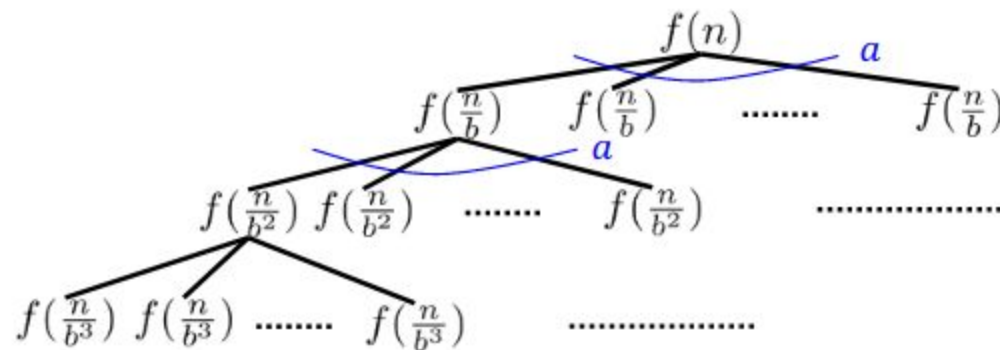
Sprout



Master Theorem

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

$$T(n) = 9T\left(\frac{n}{3}\right) + n, T(1) = 1$$



$$f(n) = n$$

$$af\left(\frac{n}{b}\right) = \frac{9}{3}n$$

$$a^2 f\left(\frac{n}{b^2}\right) = \left(\frac{9}{3}\right)^2 n$$

$$a^3 f\left(\frac{n}{b^3}\right) = \left(\frac{9}{3}\right)^3 n$$

⋮

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

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



Master Theorem

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

- 比較 $f(n)$ 跟 $O(n^{\log_b a})$ 的關係

- 如果 $f(n) = O(n^{\log_b a})$, 那麼

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

- 例如, $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

- $T(n) = O(n \log n)$, 其實就是 Merge Sort 的遞迴式

- 就是直接在後面多補一個 $\log$

Sprout



Master Theorem

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

- 比較 $f(n)$ 跟 $O(n^{\log_b a})$ 的關係
- 如果 $f(n) = O(n^{\log_b a})$ (**多項式量級相同時**), 那麼

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

- 例如, $T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$
- $T(n) = O(n \log^2 n)$
- 其實在這個 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

Sprout



Master Theorem

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

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

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

- 例如, $T(n) = T\left(\frac{n}{2}\right) + O(n^2)$
- $T(n) = O(n^2)$
- 這個 case 其實有個額外的條件, 但是不常發生(?)

Sprout



分治經典題

- 接下來來講講實際上會遇到的題目

Sprout



最大連續和

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

Sprout



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

Sprout



最大連續和

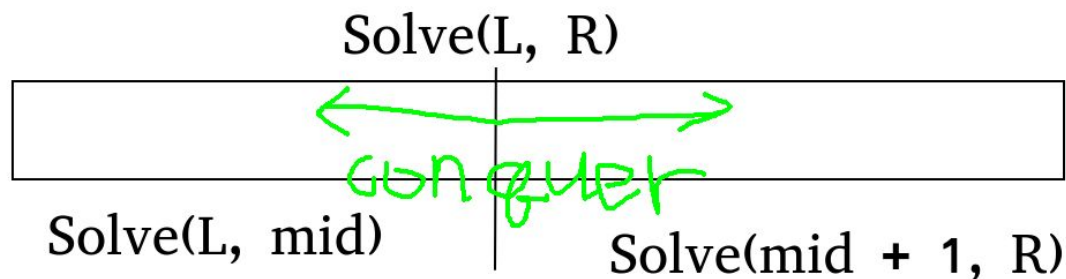
- 答案有 $O(N^2)$ 種
- 我們有沒有辦法好好的 divide 成一半, 然後想盡辦法 conquer 起來呢?

Sprout



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



- $\text{Solve}(L, R)$ 會回傳 $a_L, \dots, a_R$ 的最佳解

Sprout



最大連續和 --- conquer

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

Sprout



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);  
}
```

prout



最大連續和

- 複雜度分析

Sprout



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, $T(n) = 2T(n / 2) + O(n)$

Sprout



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 $T(n) = O(n \log n)$

Sprout



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?

Sprout



多項式乘法

- 給你兩個 n 次多項式, 請你求出兩個多項式的乘積。

- Ex: $f(x) = 2x^2 + 3x - 1, g(x) = x + 4$

$$f(x) \times g(x) = 2x^3 + 11x^2 + 11x - 4$$

- 練習題: <https://tioj.ck.tp.edu.tw/problems/1064>
 - 拿部份分數就好了(?)
 - 把 x 當成 10 這樣

Sprout



多項式乘法

- 給你兩個 n 次多項式, 請你求出兩個多項式的乘積。
- Ex: $f(x) = 2x^2 + 3x - 1, g(x) = x + 4$
 $f(x) \times g(x) = 2x^3 + 11x^2 + 11x - 4$
- 來想想看要怎麼做吧

Sprout



多項式乘法

- Naïve 作法
- 用個雙重迴圈跑一跑乘一乘加一加
- 時間複雜度是 $O(n^2)$

Sprout



多項式乘法

- Naïve 作法
- 用個雙重迴圈跑一跑乘一乘加一加
- 時間複雜度是 $O(n^2)$
- 好慢(?)
- 有沒有更快的方式？

Sprout



多項式乘法

- Naïve 作法
- 用個雙重迴圈跑一跑乘一乘加一加
- 時間複雜度是 $O(n^2)$
- 據說在古早的時代，從來沒有人質疑過乘法可以做得更快
- 可能會有人想到 FFT(Fast Fourier Transform, 快速傅立葉轉換)，但我現在沒有要講這個
 - 有興趣的同學可以查查相關資料呦

Sprout



多項式乘法

- 把多項式表達為

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n = \sum_{i=0}^n a_ix^i$$

- 同理： $g(x) = \sum_{i=0}^n b_ix^i$

- 順便不失一般性的假設 $n+1$ 是 2 的次方 (如果不是的話，就加入一些係數為 0 的高次項)
 - 為了方便之後的講解 :)

Sprout



多項式乘法

- 既然是在教分治，那就先把多項式切兩半看看

- 令 $A = a_0 + a_1x + \dots + a_{[n/2]}x^{[n/2]}$

$$B = a_{[n/2]+1} + a_{[n/2]+2}x + \dots + a_nx^{[n/2]}$$

- 同理 $C = b_0 + b_1x + \dots + b_{[n/2]}x^{[n/2]}$

$$D = b_{[n/2]+1} + b_{[n/2]+2}x + \dots + b_nx^{[n/2]}$$

Sprout



多項式乘法

- 既然是在教分治，那就先把多項式切兩半看看

- 令 $A = a_0 + a_1x + \dots + a_{[n/2]}x^{[n/2]}$

$$B = a_{[n/2]+1} + a_{[n/2]+2}x + \dots + a_n x^{[n/2]}$$

$$C = b_0 + b_1x + \dots + b_{[n/2]}x^{[n/2]}$$

$$D = b_{[n/2]+1} + b_{[n/2]+2}x + \dots + b_n x^{[n/2]}$$

- 所以, $f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$
 $= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$

Sprout



多項式乘法

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- A, B, C, D 都是 $(n/2)$ 次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！

Sprout



多項式乘法

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- A, B, C, D 都是 $(n/2)$ 次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！
- 成功切成子問題 => Divide & Conquer!
- 可是，這樣真的有比較快嗎？

Sprout



多項式乘法

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- A, B, C, D 都是 $(n/2)$ 次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！
- 成功切成子問題 => Divide & Conquer!
- 可是, 這樣真的有比較快嗎？
- 來分析一下複雜度, $T(n) = 4T(n / 2) + O(n)$
 - $O(n)$ 是加法的部份

Sprout



多項式乘法

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- A, B, C, D 都是 $(n/2)$ 次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！
- 成功切成子問題 => Divide & Conquer!
- 可是, 這樣真的有比較快嗎？
- 來分析一下複雜度, $T(n) = 4T(n / 2) + O(n)$
- 用前面的教的 Master Theorem, 可以得到 $T(n) = O(n^2)$
- 沒有變快 QQQ

Sprout



Karatsuba

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- 我們要算 AC , $AD + BC$, BD

Sprout



Karatsuba

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- 我們要算 AC , $AD + BC$, BD
- 靈光一閃, 如果我們計算

$$E = (A + B) \times (C + D) = AC + AD + BC + BD$$

Sprout



Karatsuba

$$\begin{aligned} f(x) \times g(x) &= (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D) \\ &= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD \end{aligned}$$

- 我們要算 AC , $AD + BC$, BD
- 靈光一閃, 如果我們計算

$$E = (A + B) \times (C + D) = AC + AD + BC + BD$$

- 就可以把原本的式子化為

$$f(x) \times g(x) = AC + x^{\frac{n+1}{2}} (E - AC - BD) + x^{n+1} BD$$

- 乘法只剩下三次了！

Sprout



Karatsuba -- 複雜度

- 剛剛上面那個演算法, 被稱做 Karatsuba Algorithm
- 我們來分析一下時間複雜度吧

Sprout



Karatsuba -- 複雜度

- 剛剛上面那個演算法, 被稱做 Karatsuba Algorithm
- 我們來分析一下時間複雜度吧

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

- 根據前面教的 Master Theorem, 可以得到 $T(n)$ 就是

$$O(n^{\log_2 3}) \approx O(n^{1.58})$$

Sprout



Karatsuba -- 複雜度

- 剛剛上面那個演算法，被稱做 Karatsuba Algorithm
- 我們來分析一下時間複雜度吧

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

- 根據前面教的 Master Theorem, 可以得到 $T(n)$ 就是

$$O(n^{\log_2 3}) \approx O(n^{1.58})$$

- 類似的做法也有被應用在矩陣乘法上，有興趣的同學可以查查

Sprout



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。
- $N \leq 100,000$

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

- 練習題: <https://tioj.ck.tp.edu.tw/problems/1500>
、 <https://codeforces.com/contest/429/problem/D>

Sprout



平面最近點對

- 開心 $C(N, 2) = O(N^2)$ 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?

Sprout



平面最近點對

- 我們可以先把所有輸入的點照 x 座標排序後，在中間畫一條分隔線。
- 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。

Sprout



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？

Sprout



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.....

Sprout



平面最近點對

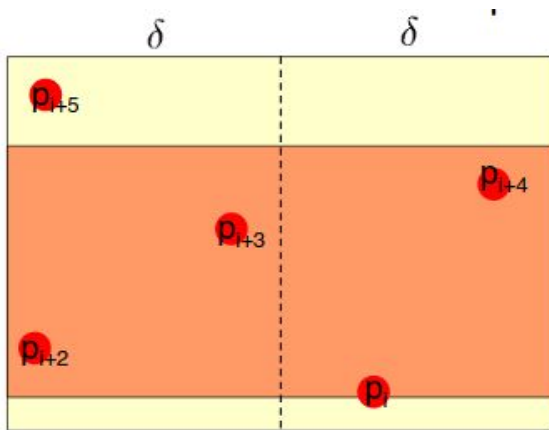
- 定神一想，會發現如果遞迴下去後找到的最近點對的距離是 d ，那麼我們根本就不需要考慮那些距離超過 d 的點對
- 所以離分隔線超過 d 的點都不需要去考慮。
- 對於每個點，也只有 y 座標差距不超過 d 的點可能可以讓你找到更近的點對
- 這樣子複雜度會是好的嗎？
- 聽起來只是個壓常數的剪枝，但在什麼情況下，這個做法一樣會退化成 $O(N^2)$ 呢？

Sprout



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！

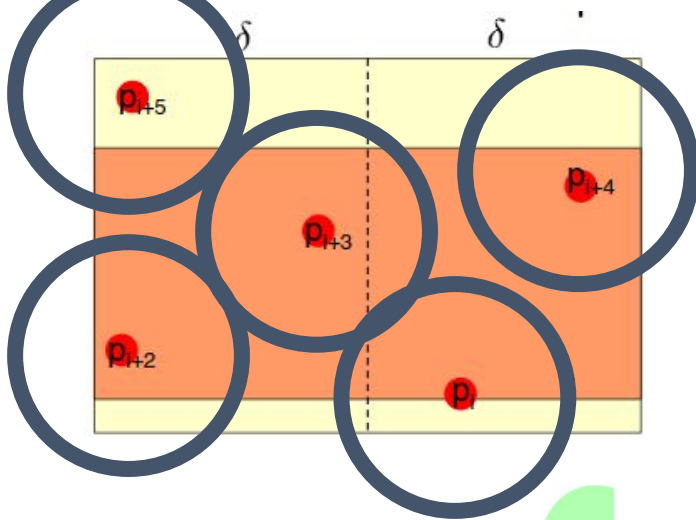


Sprout



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！

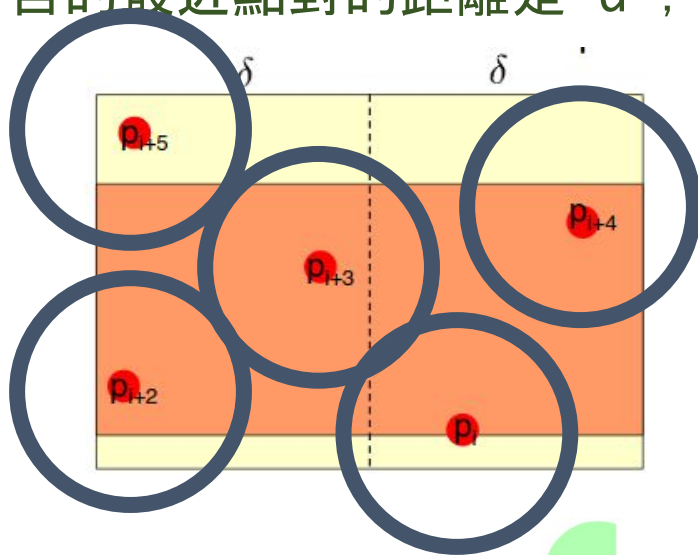


Sprout



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！



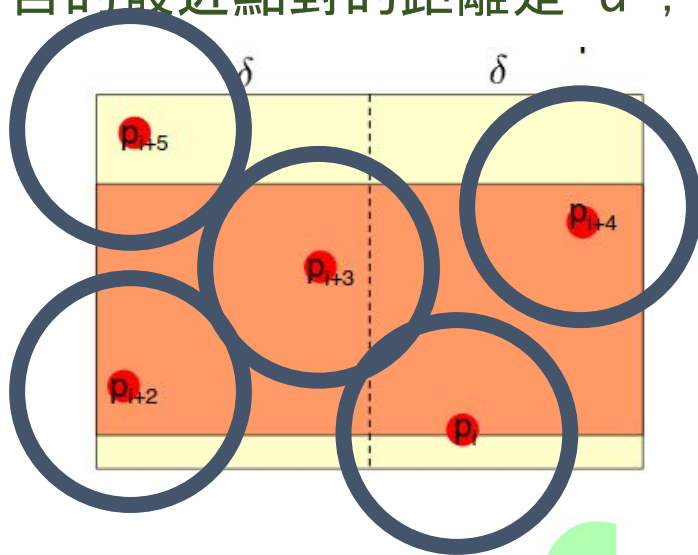
- 每個點都在各自的半邊張出半徑為 d 的圓

Sprout



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！



- 每個點都在各自的半邊張出半徑為 $d/2$ 的圓
- $\Rightarrow$ 每個點至少在自己半邊吃掉 $\pi(d/2)^2/4$ 的面積！



平面最近點對

- 每個點至少在自己半邊吃掉 $\pi(d/2)^2/4$ 的面積！

Sprout



平面最近點對

- 每個點至少在自己半邊吃掉 $\pi d^2/16$ 的面積！
- 一個半邊有 d^2 的面積
- $\Rightarrow$ 至多 $d^2 / (\pi d^2/16) = 16 / \pi$, 大約 5 個點！
- 也就是說, 每個點往上看只要看對面 5 個點, 就可以把對面 y 座標大於自己不超過 d 的點看完了
- 實際上存在更緊的分析方式, 使得實作上可以
 - 把距離中線 $\leq d$ 的點都混在一起對 y 排序
 - 由 y 小掃到大, 每個點往前看三個點
 - 每看一次都跟最佳解取 $\min$, 就可以很神秘的找到最近點對了

Sprout



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

Sprout



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 master theorem , 都可以得到 $O(n \log^2 n)$
- 注意這裡的 master theorem 要套 Case 2

Sprout



平面最近點對

- 不能做到更好嗎？ $O(n \log^2 n)$ 感覺很慢

Sprout



平面最近點對

- 不能做到更好嗎？ $O(n \log^2 n)$ 感覺很慢
- 靈光一閃，想起 merge sort

Sprout



平面最近點對

- 不能做到更好嗎？ $O(n \log^2 n)$ 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 $O(n)$
- $T(n) = 2T(n / 2) + O(n)$
- $T(n) = O(n \log n)$!

Sprout



平面最近點對

- 不能做到更好嗎？ $O(n \log^2 n)$ 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 $O(n)$
- $T(n) = 2T(n / 2) + O(n)$
- $T(n) = O(n \log n)$!
- 這題也有非分治的作法，有興趣可以上網查查

Sprout



尋找第 k 大

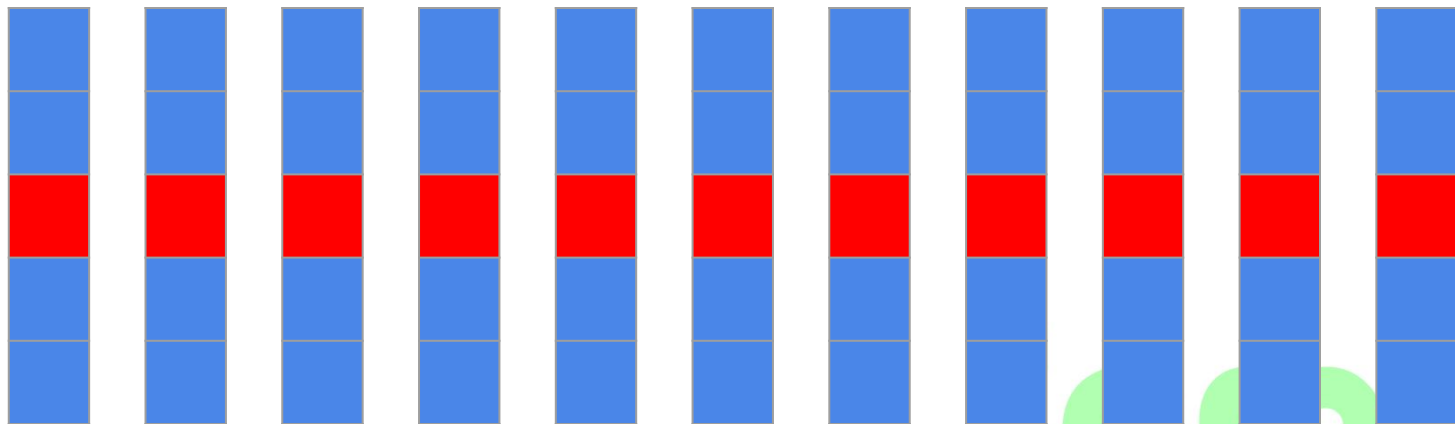
- 給你一個序列, 請你找到第 k 大
- 還不簡單? `sort` 就好啦!
- 但我希望可以做到 $O(n)$

Sprout



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**

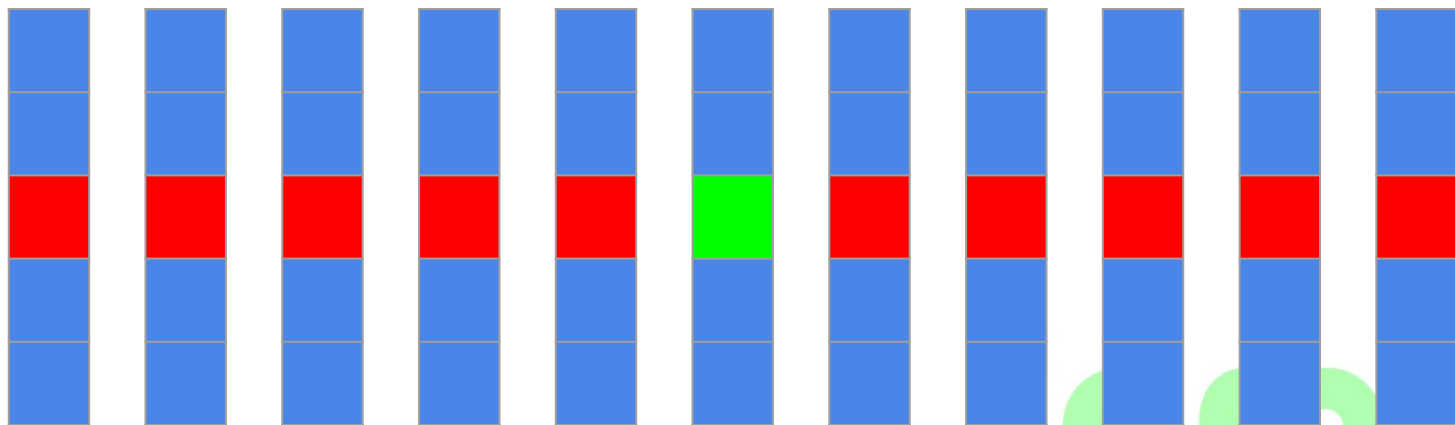


spout



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？

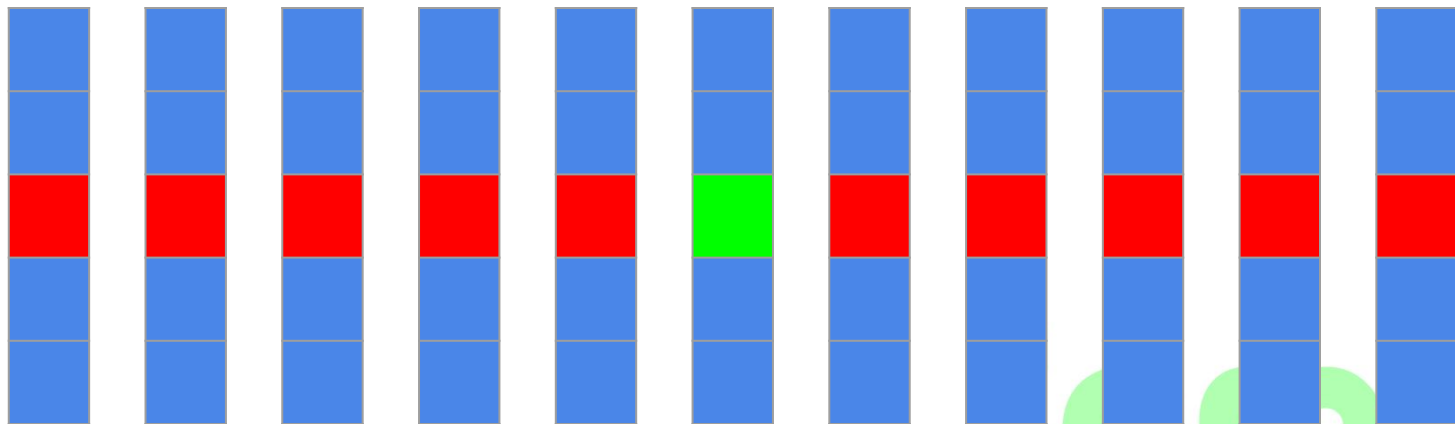


spout



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

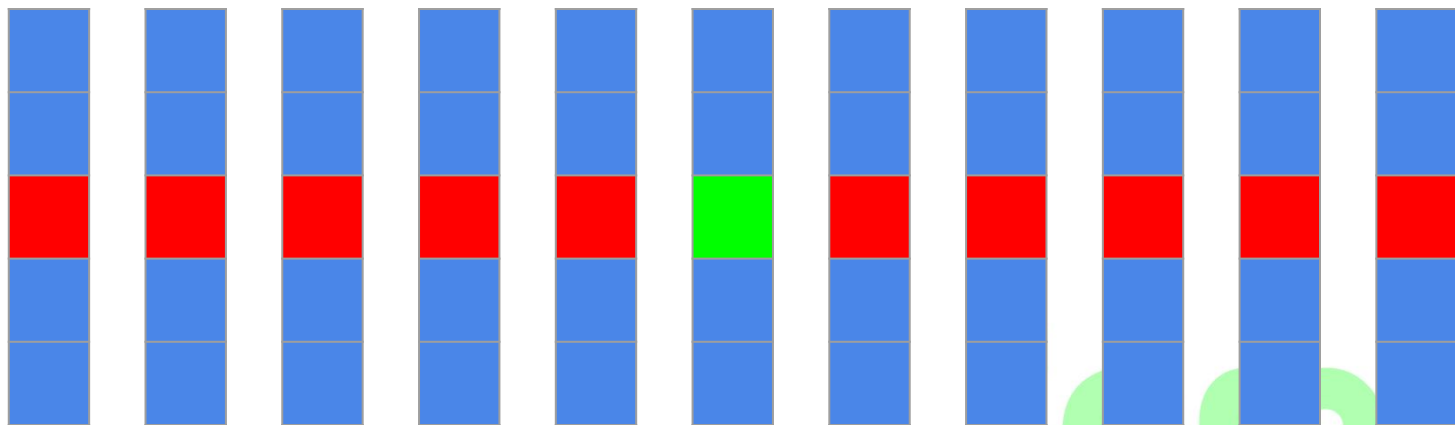


spout



尋找第 k 大

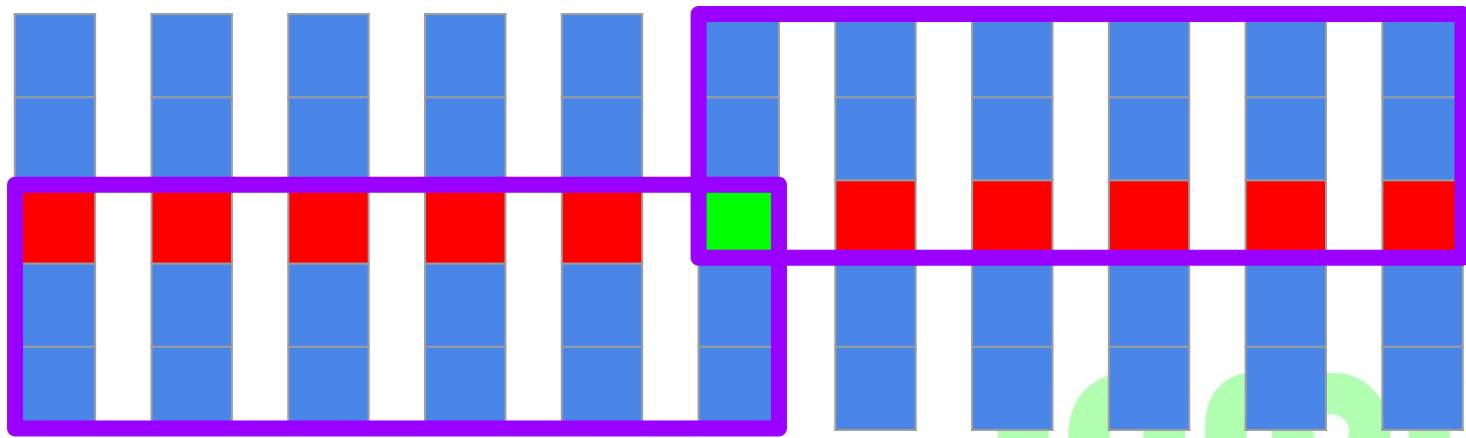
- 令剛剛找到的數字是 p , 把數字分成 $>p$ 跟 $<p$ 兩堆





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?

Sprout



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 $\rightarrow T(7n/10)$
- 分組、分兩堆等等： $O(n)$

Sprout



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 $\rightarrow T(7n/10)$
- 分組、分兩堆等等： $O(n)$

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

Sprout



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 $\rightarrow T(7n/10)$
- 分組、分兩堆等等： $O(n)$

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

- 不能主定理，那就來用數學歸納法！

Sprout



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 $\rightarrow T(7n/10)$
- 分組、分兩堆等等： $O(n)$

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

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

Sprout



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 $\rightarrow T(7n/10)$
- 分組、分兩堆等等： $O(n)$

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

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

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

Sprout



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 $\rightarrow T(7n/10)$
- 分組、分兩堆等等： $O(n)$

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

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

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

- 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

Sprout



總結分治

- 已經看過很多在序列、平面的題目了
- 但是其實分治還能在奇怪(?) 的地方分治
 - 例如樹、圖等等
- 往往分治還需要搭配很多噁心的資料結構、演算法
- 分治常常會在偏難的題目中走出一條通路
- 大家加油

Sprout