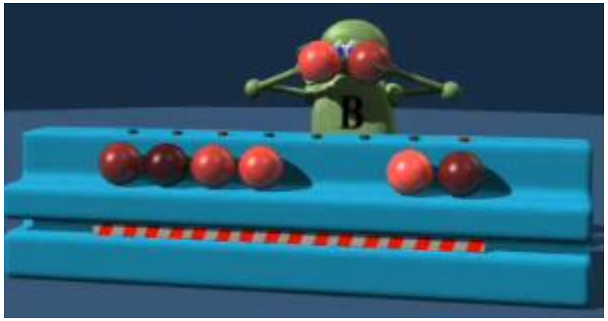
線性時間的sorting

Sorting in Linear Time
Chapter 8

Mei-Chen Yeh

3/21(二)

How fast can we sort?



- Comparison sorts
  - Bubble sort, selection sort, insertion sort, merge sort, quick sort 這些演算法的worst-case最好可以提升到 O(nlogn)
  - The **best** *worst-case* running time: $O(n\log n)$

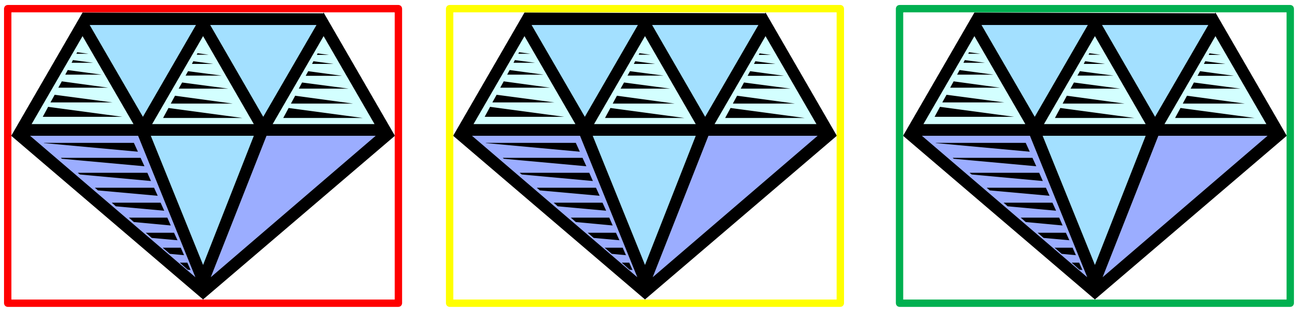
Q: Is $O(n\log n)$ the best we can do?

A ***decision tree*** can help us answer this question!

決策樹

Sorting 3 diamonds

把3個鑽石由輕排到重，只有天秤可以用，沒有磅秤

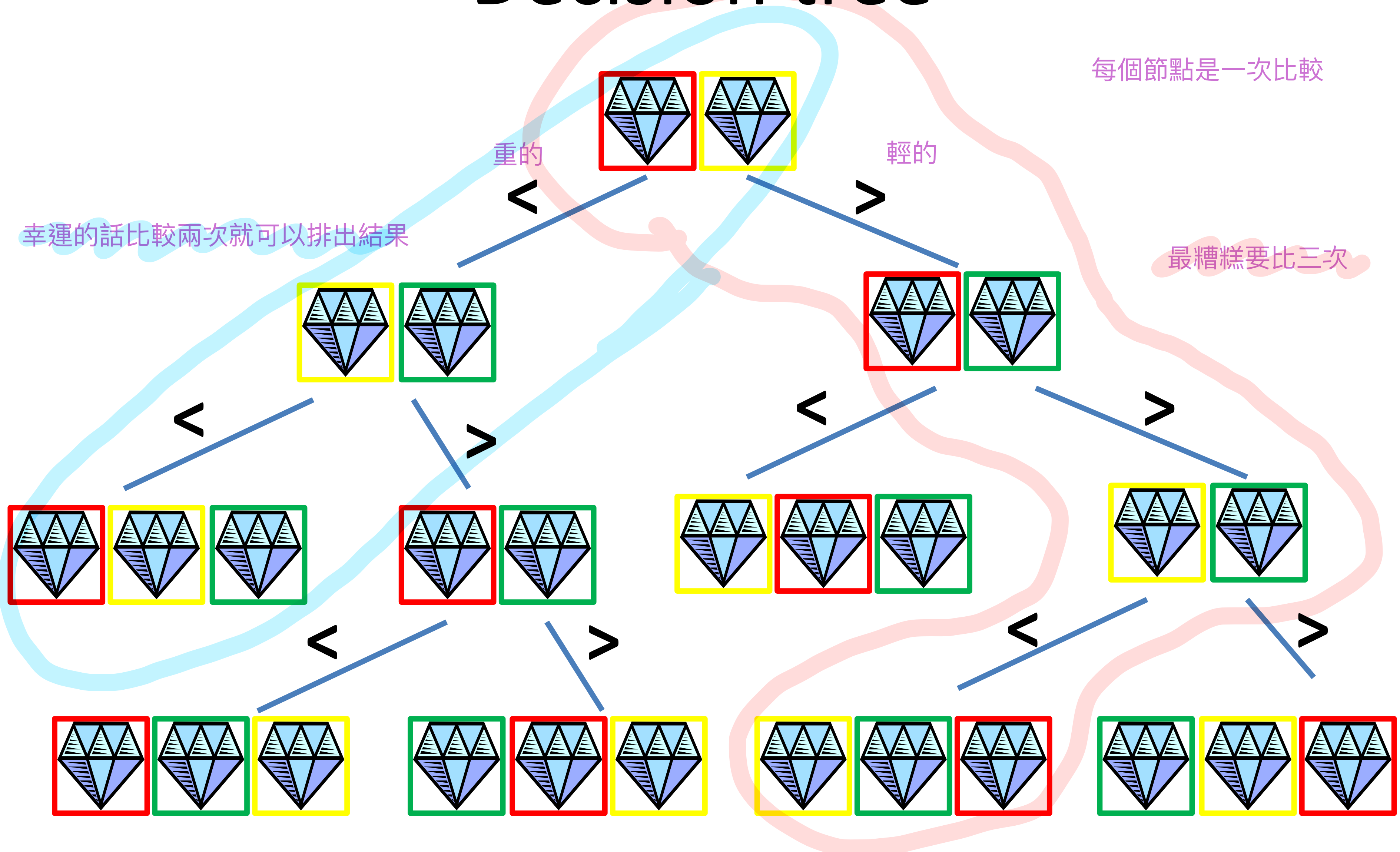


Minimal number of comparisons?

Maximal number of comparisons?

要排列3個鑽石，總共會有3!=6種排列結果

Decision tree



Decision tree model

- Each leaf contains a permutation. 每個樹葉都是一種排序結果
- *n* elements => *n*! permutations n個鑽石要排，共會有n!種結果
- # {comparisons} of the algorithm = the length of the path taken
- Worst-case running time = height of tree

Lower bound for comparison sorts

如果發生兩兩比較就一定會是O(nlogn)

Theorem

Any comparison sort algorithm requires comparisons in $\Omega(n\log n)$ in the worst case.

Proof

有L個樹葉

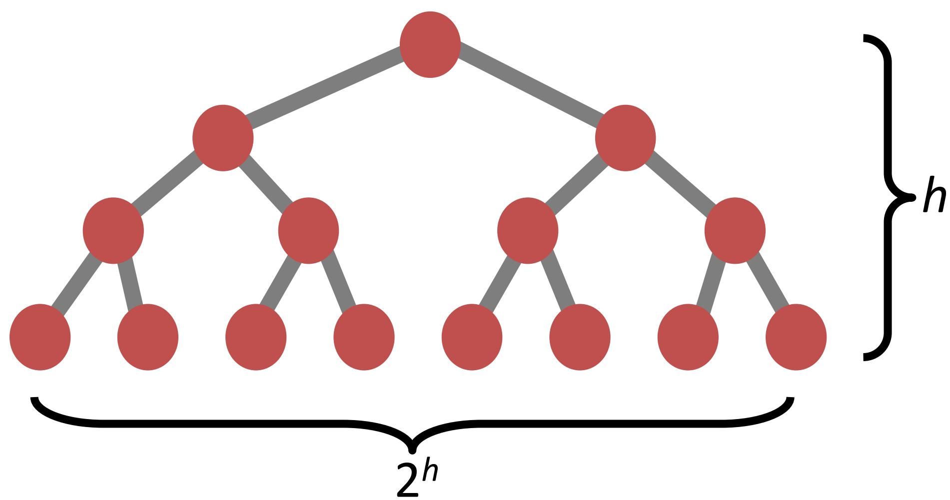
Consider a decision tree of height h with L leaves.

$n! \le L \le 2^h$

$h \ge \log(n!)$

$= \Omega(n\log n)$ *Why?*

h一定比OMEGA(nlogn)大



$\log(n!) = \Omega(n\log n)$

log(n!)可以寫成(nlogn)起跳的一個式子

$\Omega(n\log n)$

1. Informally,

$\log(n!) > \log((n/2)^{n/2}) = (n/2)\log(n/2) \rightarrow \Theta(n\log n)$

$n! = n(n\text{-}1)(n\text{-}2)\ldots\ldots1$

n/2

2.718

2. Stirling's Approximation: $n! > (n/e)^n$

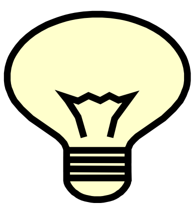
$\log(n!)$
$> \log((n/e)^n)$
$= n(\log n - \log e)$
$= \Theta(n\log n)$

Sorting in linear time

No comparisons between elements!

- Counting sort
- Radix sort

Counting sort：不兩個兩個比了，直接用數的



[Example] Sort by age (young → old):
Count the number of people who is younger than me!

Counting sort: the simplest setting

- Assumptions on the input
  - Distinct elements
  - Positive integers
  - Range is known (1..*k*)

Algorithm Counting_Sort

1. Allocate $C[1..k]$ $O(1)$
2. **for** $i = 1$ **to** k k代表數值的上界(要排序的數字最大的那個) $O(k)$
3. $C[i] = 0$
4. **for** $i = 1$ **to** n $O(n)$
5. $C[A[i]] = C[A[i]] + 1$
6. **for** $i = k$ **downto** 1 $O(k)$
7. **if** $C[i] > 0$ **print** i

$\max(O(n),O(k))$

	1	2	3
A	4	1	2

還沒排序的ArrayA

	1	2	3	4
C	1	1	0	1

>> 4 2 1

① 需要排列的數字有4、1、2
② 給他一個新的array來存（array的index就代表要排列的數字，比方說index4的位置，因為要排的數字有4，所以就在index4的位置存1)
③ 最後把有存1這個數字的index印出來，就會呈現正確的排序結果。

General setting

- Assumptions on the input
  - ~~Distinct elements~~
  - Positive integers
  - ~~Range is known (1..k)~~

0. Let k be the maximum of A[1]..A[n] $O(n)$

1. Allocate $C[1..k]$
2. **for** $i = 1$ **to** k
3. $\qquad C[i] = 0$
4. **for** $i = 1$ **to** n
5. $\qquad C[A[i]] = C[A[i]] + 1$
6. **for** $i = k$ **downto** 1 $O(kn)$

   $\qquad$ **for** $j = 1$ to $C[i]$ ☹

   $\qquad\qquad$ **print** i

最慘的情況有可能O(kn)

Alternate algorithm for counting sort

counting sort的意思就是，數一下比自己小的有幾個，自己要排在他們後面。

```
for i = 1 to k
    C[i] = 0
for i = 1 to n          數數值為i的，在k裡面有幾個
    C[A[i]] = C[A[i]] + 1
for i = 2 to k          累加：把比i小的都加起來
    C[i] = C[i] + C[i-1]
for i = n downto 1      i等於5到1
    B[C[A[i]]] = A[i]
    C[A[i]] = C[A[i]] - 1
```

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

C':	1	1	3	5

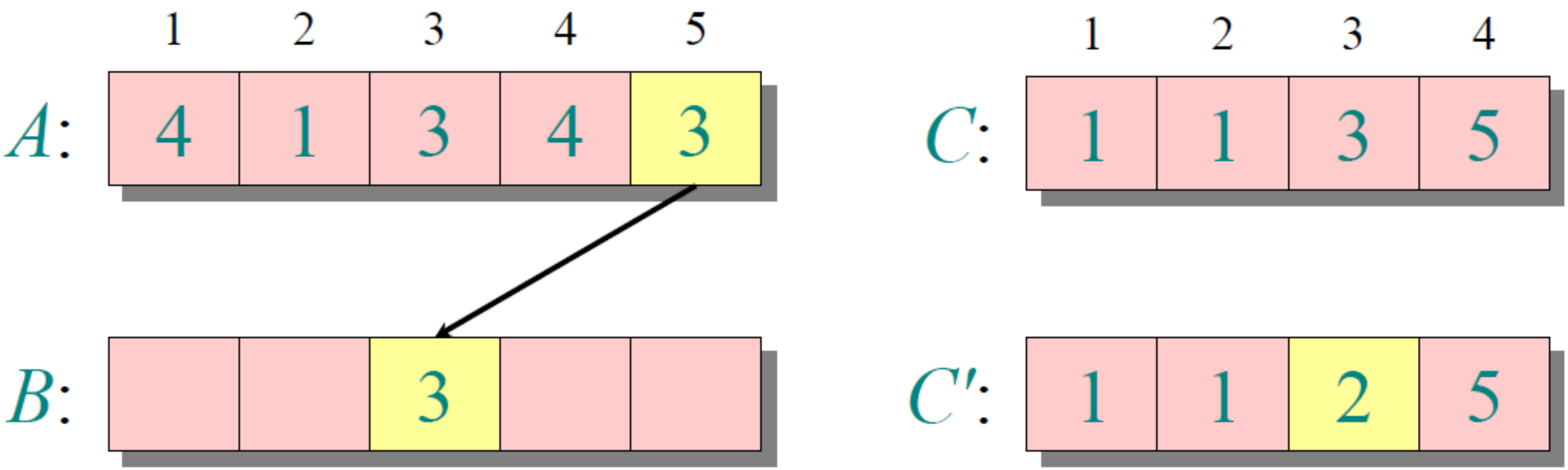
小於等於3的3個

小於等於4的5個

為什麼counting sort可以在線性時間做完
去掉了兩個兩個拿來比的程序，
他是多開一個array，還計算每個數出現了幾次
再用C’來做累加，例如：小於等於1的幾個，小於等於2的幾個

Results are stored in *B*[] 把處理好的結果搬到Barray，最後輸出Barray，就會是排序好的結果

$i = n$ (5)



for $i = n$ **downto** 1

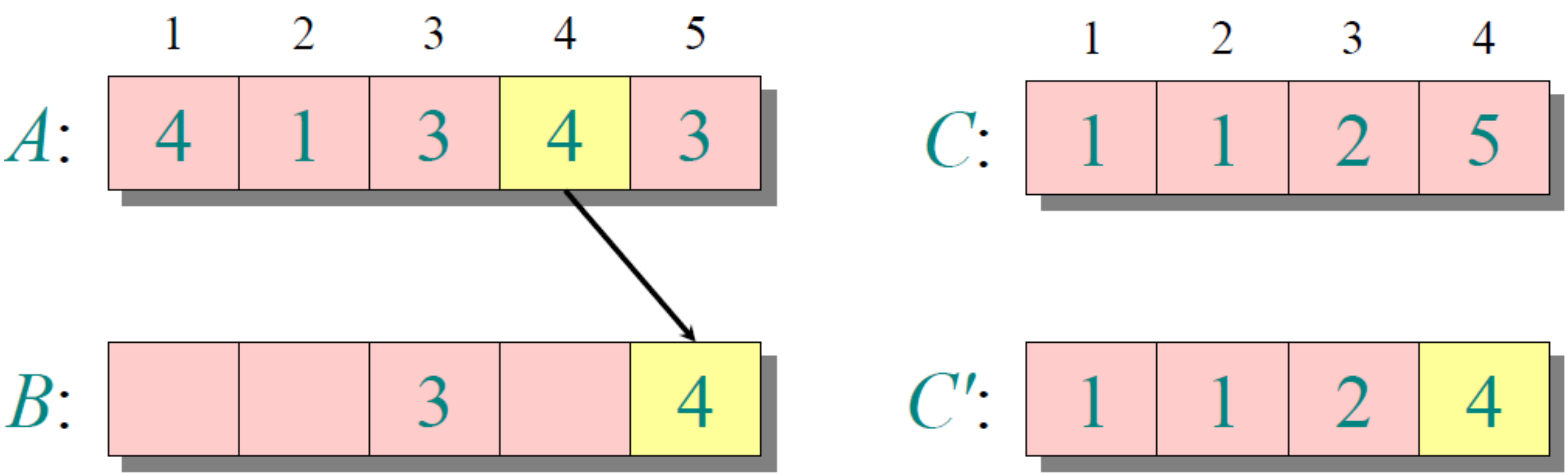
  B[C[A[i]]] = A[i]

  C[A[i]] = C[A[i]] - 1

前頁程式碼第7,8,9行

$i = n$-1 (4)

做完之後會把Aarray變成Barray，Barray是排序好的



for i = n **downto** 1

B[C[A[i]]] = A[$\overset{4}{i}$]　　C[4]是5

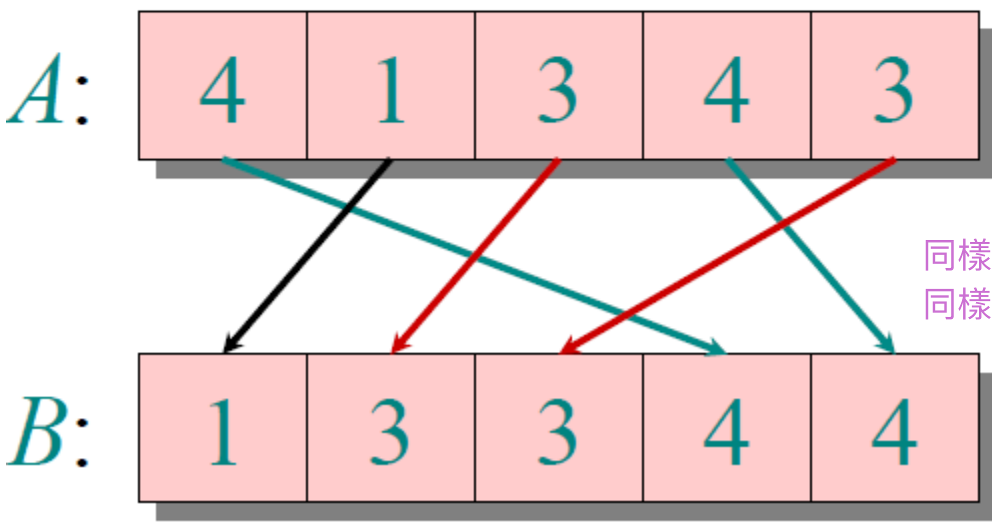
C[A[i]] = C[A[i]] - 1

… take-home practice

Stable sorting

兩個一樣的數字，在原本的array中排左邊的，在新的array還是會排在左邊

- Preserves the input order among equal elements



同樣數字出現的順序會保持一致，
同樣顏色的箭頭不會出現交叉

Q: *Counting sort is a stable sort?* O **Yes!**

Merge sort? O *Quick sort?* X

3/21 (五) end

作業手寫題要複習
期中考手寫，不上機
清明連假會放考古題到noodle

Sorting in linear time

No comparisons between elements!

- Counting sort
- Radix sort

3/23 (四) start

Radix sort

- Idea: digit-by-digit sort
- Most significant digit first or less significant digit first?

要從個位數開始排（i=1開始）

329
457
657
839
436
720
355

Radix-Sort (A, d)

如果不是stable，只是單純地讓排好的東西不動，這樣高位數的時候沒有辦法整坨移動。

d是常數，在這個例子中d=3（三位數）

1. **for** i = 1 **to** d 要從個位數開始排（i=1開始），i代表位數
2. use a *stable sort* to sort A on digit i

這個排序一定要用stable sort來做（counting sort/merge sort）

$O(n)$

最後才是排百位數

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

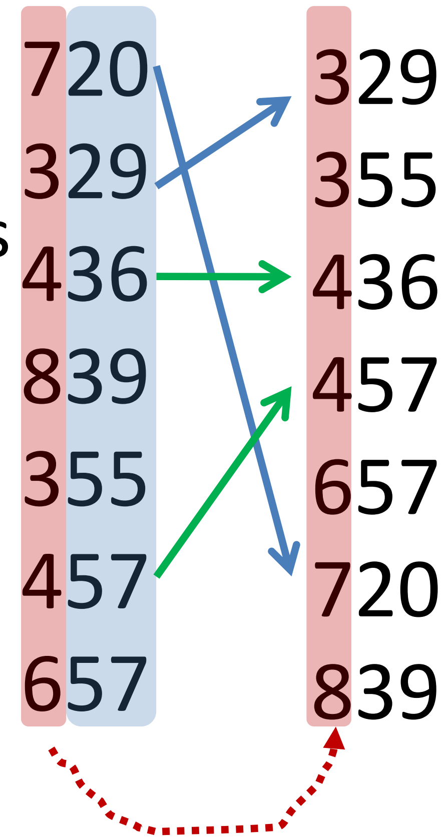
$O(d \cdot (n+k))$

根據個位數從小到大排，所以720上去

再來根據十位數排，720的2要在329的2上面

Correctness of Radix Sort

- Induction on digit position
  - Assume that the numbers are sorted by their low-order *k*-1 digits
  - Sort on digit *k*
    - Numbers that **equal** in digit *k*
    - Numbers that **differ** in digit *k*



為什麼從百位數開始排會錯，一定要從個位數開始排？

1. 有可能會遇到百位數的數字一樣
2. 有可能百位數排好了，結果到十位數結果被翻盤了

為什麼Radix Sort 一定要用stable sort排？

如果不stable的話，在低位數排好的東西，到高位數會亂掉。

The Selection Problem

Chapter 9

Problem statement

給我一個沒有sort的array，然後跟我講要找第 i 大的數字

- Input: A set A of n distinct numbers and an integer i, with $1 \le i \le n$.

輸入會給我一個未排序的array加上一個1~n的整數

- Output: The element x that is larger than exactly i-1 other elements of A.

輸出是從array抓一個數字出來，要剛好大於i–1個數字，也就是排序後第i個的數字

The i-th ***order statistic***

$i = 1 \rightarrow$ find the ***minimum***

$i = n \rightarrow$ find the ***maximum***

$i = n/2 \rightarrow$ find the ***median***

Example

- $i = 6$

index從1開始算

3 7 9 17 5 2 21 18 33 4

i=1 ↓ i=6 ↓

2, 3, 4, 5, 7, 9 ......

Naïve algorithm

什麼是Naive algorithm？

- Step 1: Sort A
- Step 2: Return $A[i]$

$O(n\log n)$ quick sort的時間複雜度

counting sort的時間複雜度

雖然counting sort比較有效率，
但是有些情況不能用counting sort

$O(n)$

The number of comparison to solve the selection problem is between **1.5*n*** and **5.43*n***.

Divide-and-Conquer Algorithm (1)

執行這個函式之後，會把元素跟pivot值比較後，拆成兩坨

q = *RANDOM-PARTITON*(A, p, r)

pivot值是隨機抓的

A 3 7 9 17 5 2 21 18 33 4

如果 i 是第六大的就停，如果比六小就找左邊，大就找右邊

A 3 7 5 2 4 9 17 21 18 33

q = 6 (the index starts from 1) q是告訴我們pivot值落在這個array中的第幾個

The selection problem: select the i-th ***order statistics***

What if $i = q$, $i < q$, and $i > q$?

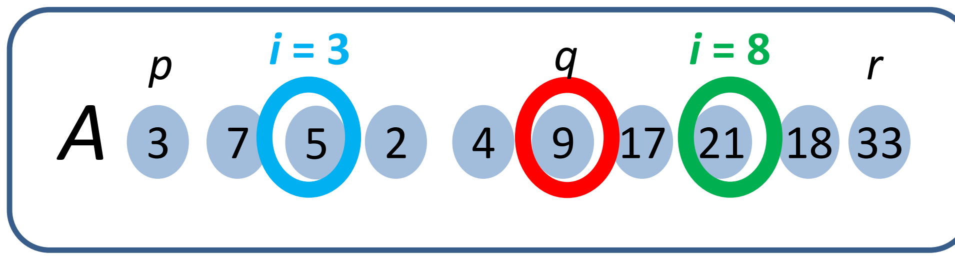
Divide-and-Conquer Algorithm (2)

會用到randomize–partition，所以叫這個名字

k把絕對位置換成相對位置。
假設今天只看一段數列，k可以幫助我們知道
在這一段pivot 值是在這一段的第幾個

```
RANDOMIZED-SELECT (A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION (A, p, r)
k = q – p + 1
if i == k
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT (A, p, q-1, i)
else return RANDOMIZED-SELECT (A, q+1, r, i-k)
```

假設pivot隨機選到9，他會給我一個結果
是以9為分界，拆成兩坨



如果i比k大，就知道
答案在右半邊，所以
從q+1開始

RANDOMIZED-SELECT (A, p, r, i)

- The worst-case running time: ?

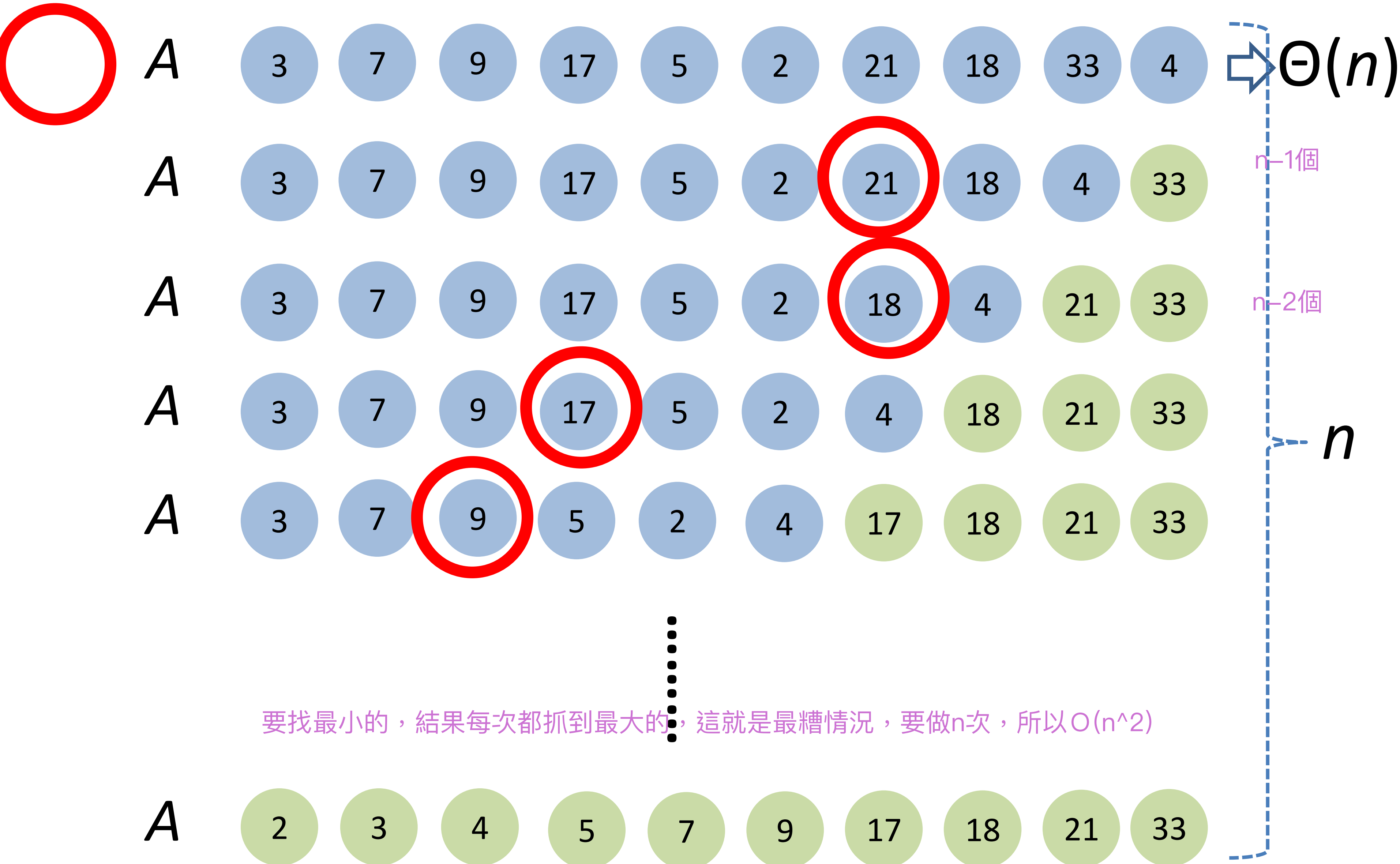
quick sort的worst case O(n^2)
因為無法掌控pivot值

A 3 7 9 17 5 2 21 18 33 4

i = 1 → find the minimum

What's the selection of pivots that leads to the worst case?

randomized–select的worst case的情況是：
每次pivot一抓都是抓到端點，然後端點沒辦法幫我分兩坨，
所以每次都得從n–1做

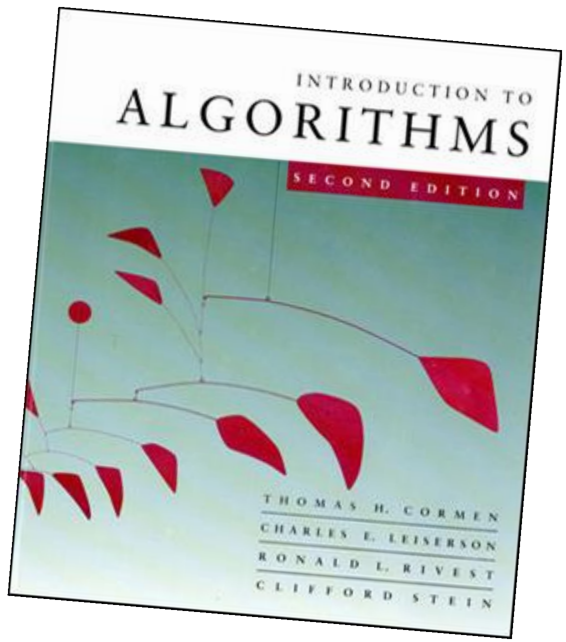


$$T(n) = T(n\text{-}1) + \Theta(n) = \Theta(n^2) \qquad \Theta(n^2)$$

RANDOMIZED-SELECT (A, p, r, i)

- The average-case running time: $\Theta(n)$

大家都會猜O(nlogn)，但其實average-case出乎意料地是O(n)



Pages 217-219

3/23(四) end

把randomized-selection的worst case變成線性時間

Selection in worst-case linear time

Guarantee a good split upon partitioning the array!

SELECT (A, p, r, i)

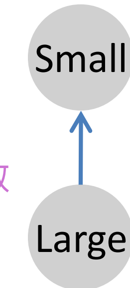
if $p == r$
  return $A[p]$
~~q = RANDOMIZED-~~
$k = q - p + 1$
if $i == k$
  return $A[q]$
elseif $i < k$
  return SELECT (A, p, q-1, i)
else return SELECT (A, q+1, r, i-k)

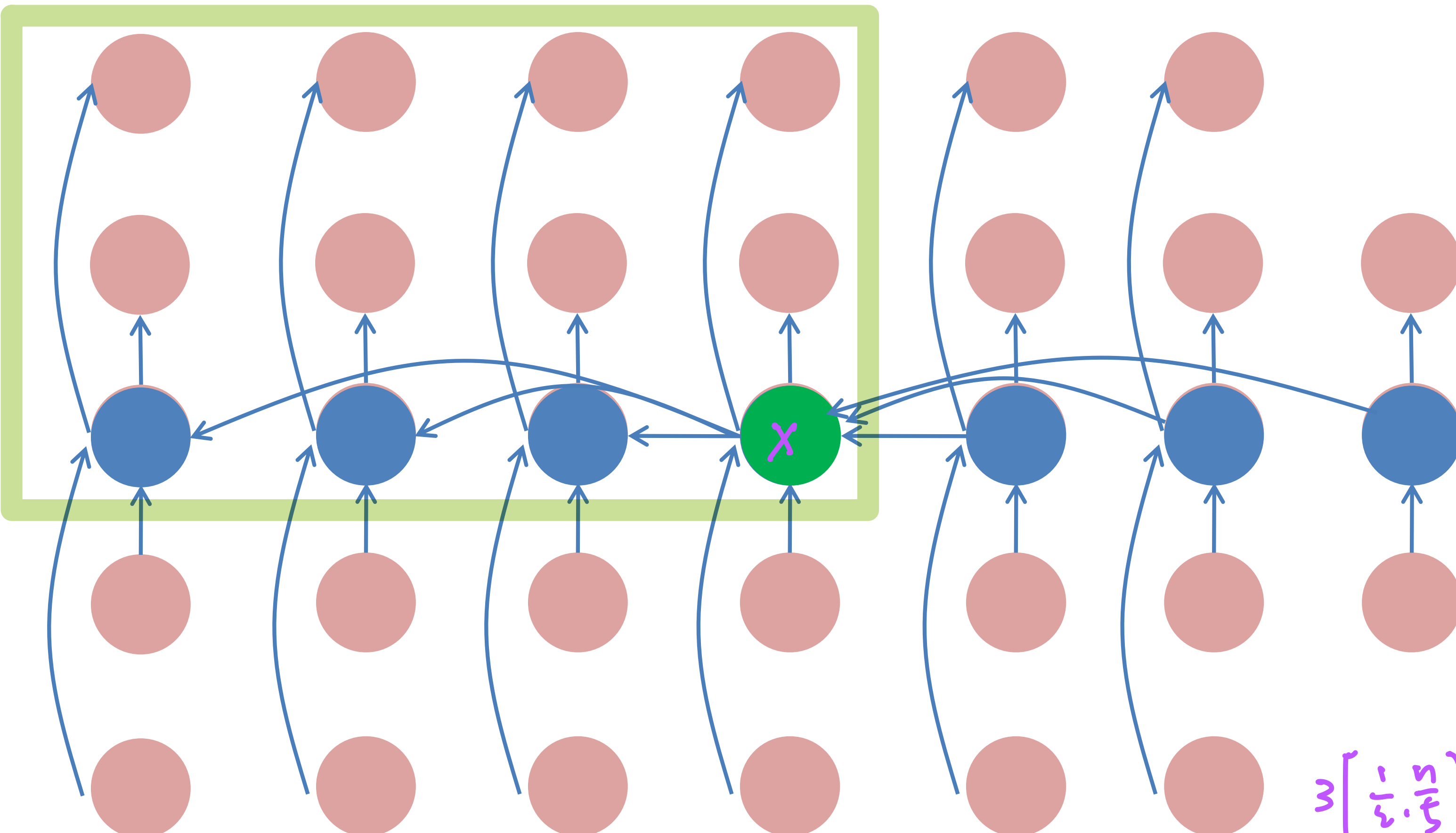
1. Divide the n elements into groups of **5**
2. Find the **median** of each 5-element group
3. Recursively select the **median** of those group medians as pivot
4. Partition the input using the pivot

3/28(二) start

Example: n = 33 ($A[0\text{-}32]$)

5個一組
選中位數
在中位數中選中位數





至少

$3\left\lceil \frac{1}{2}\cdot\frac{n}{5} \right\rceil = 3\left\lceil \frac{n}{10} \right\rceil$

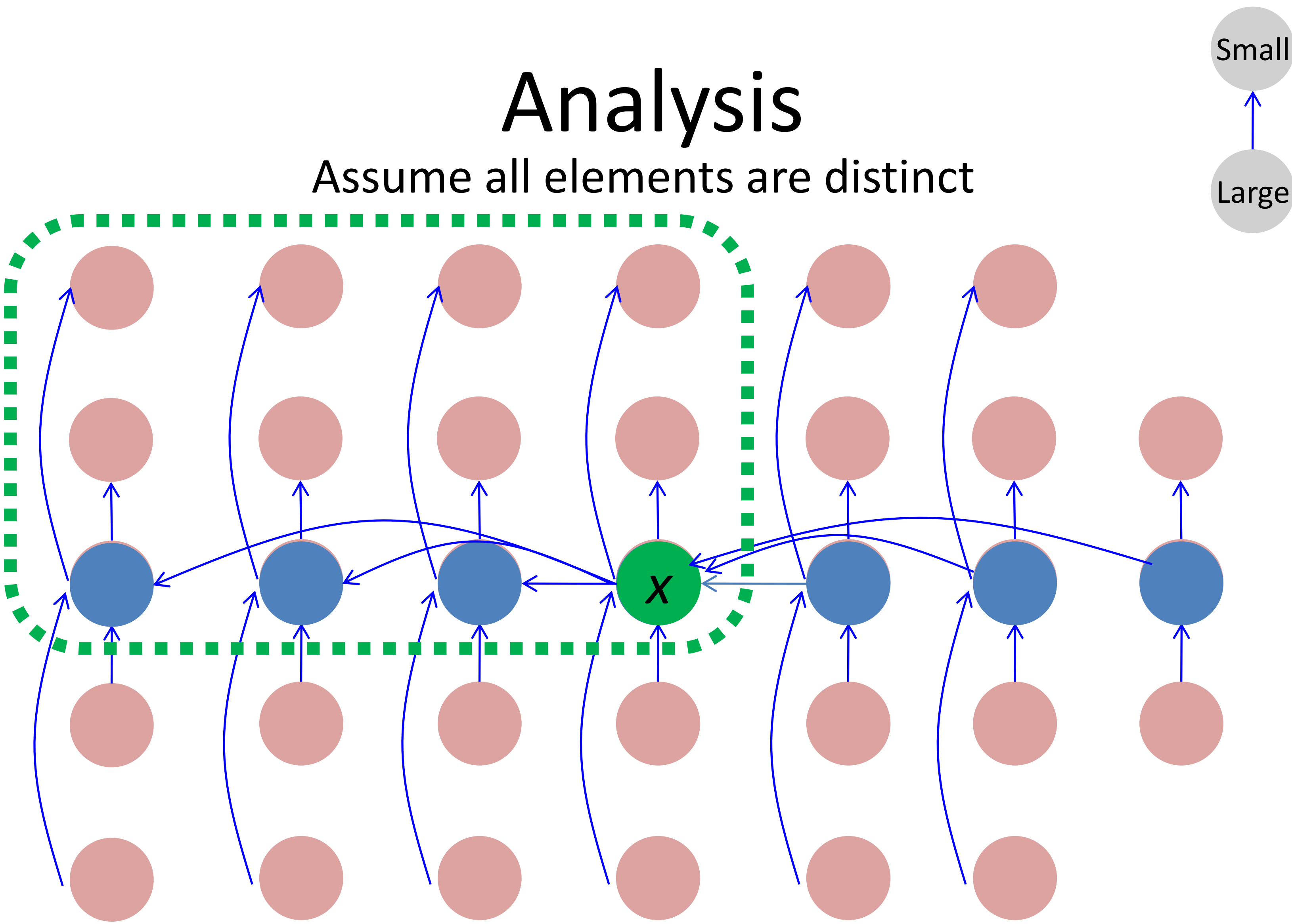
$\leq x$

Find medians in terms of ***element values***, not locations!

Linear time in the worst case!

Analysis

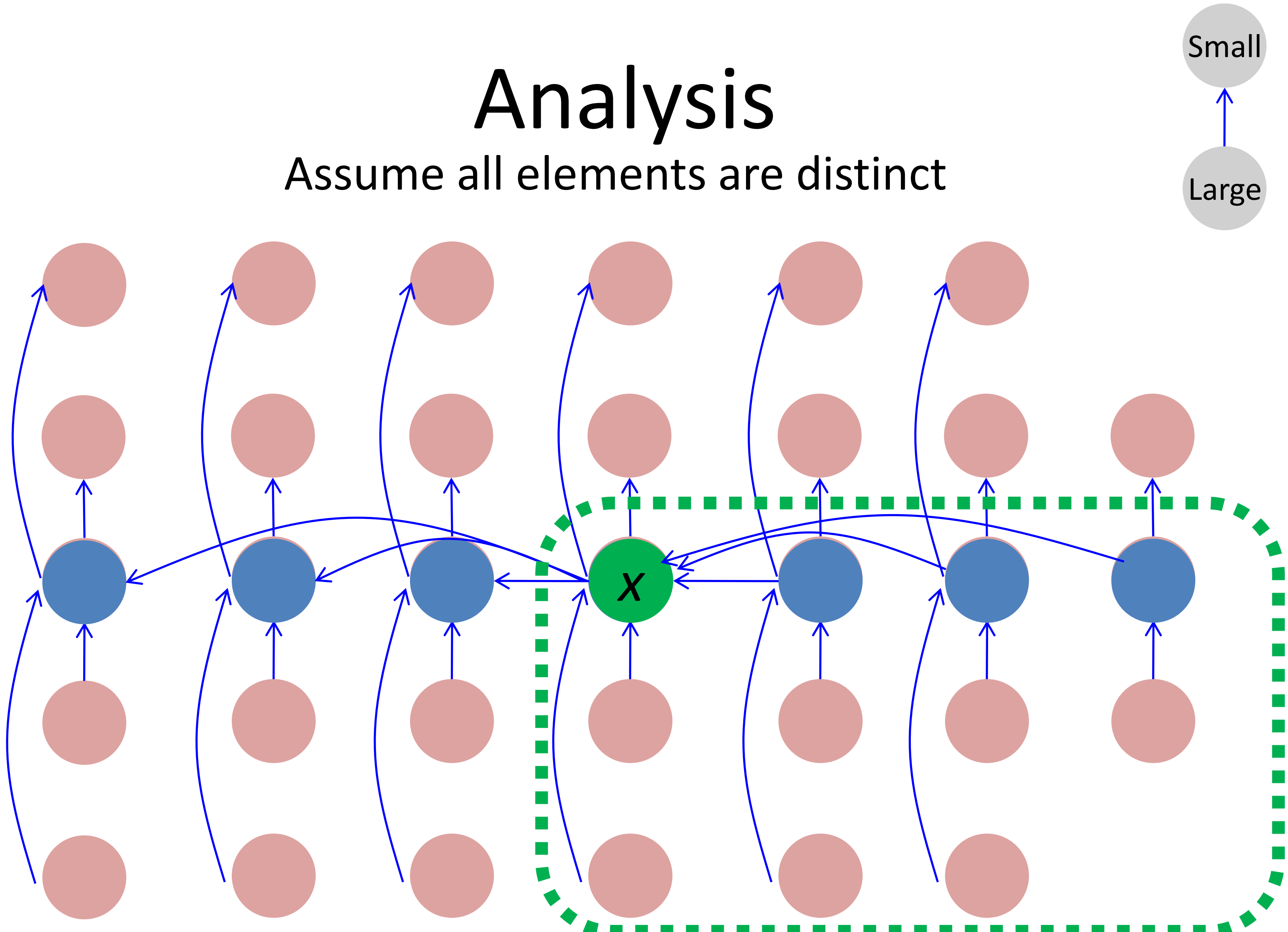
Assume all elements are distinct



How many elements are **≤ x**? *at least* $3\left\lfloor \frac{1}{2} \cdot \frac{n}{5} \right\rfloor = 3\left\lfloor \frac{n}{10} \right\rfloor$

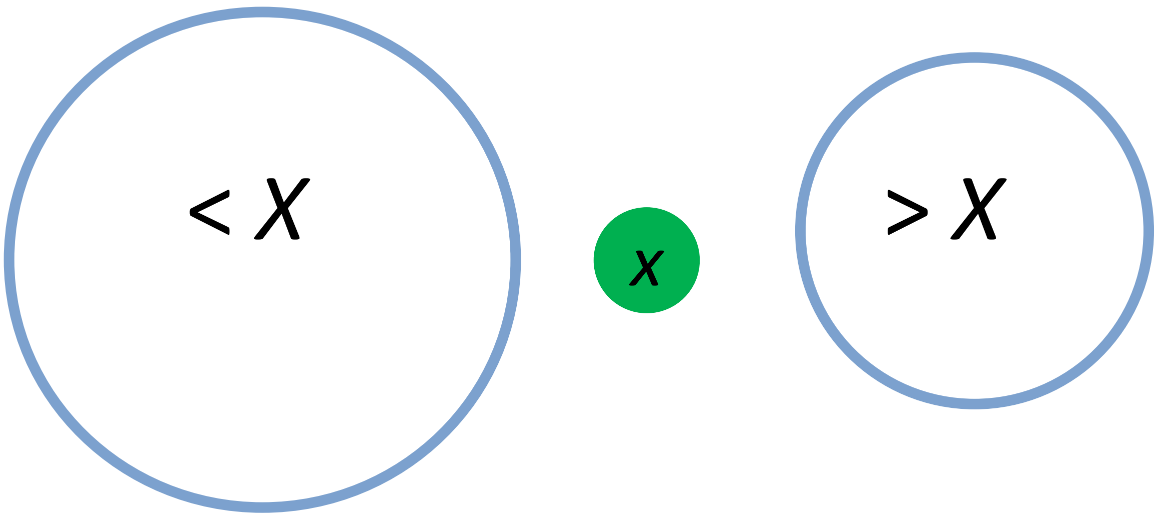
Analysis

Assume all elements are distinct



Similarly, at least $3\left\lfloor \frac{n}{10} \right\rfloor$ *elements* ≥ ***x***

- Worse case → the *i*-th statistic order always falls in the **larger** group for each partition



$|X_<|$ elements

$|X_>|$ elements

通通加起來就是T(n)

SELECT (A, p, r, i) $T(n)$

就算worst case發生，還是可以保證3n/10可以被丟棄

1. **if** $p == r$
2. **return** $A[p]$
3. *Choose the pivot*
4. $k = q - p + 1$
5. **if** $i == k$
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** SELECT $(A, p, q\text{-}1, i)$
9. **else return** SELECT $(A, q\text{+}1, r, i\text{-}k)$

$O(n)$
1. Divide the n elements into groups of **5**
2. Find the **median** of each 5-element group

$T(n/5)$
3. Recursively select the **median** of those group medians as pivot

$O(n)$
4. Partition the input using the pivot

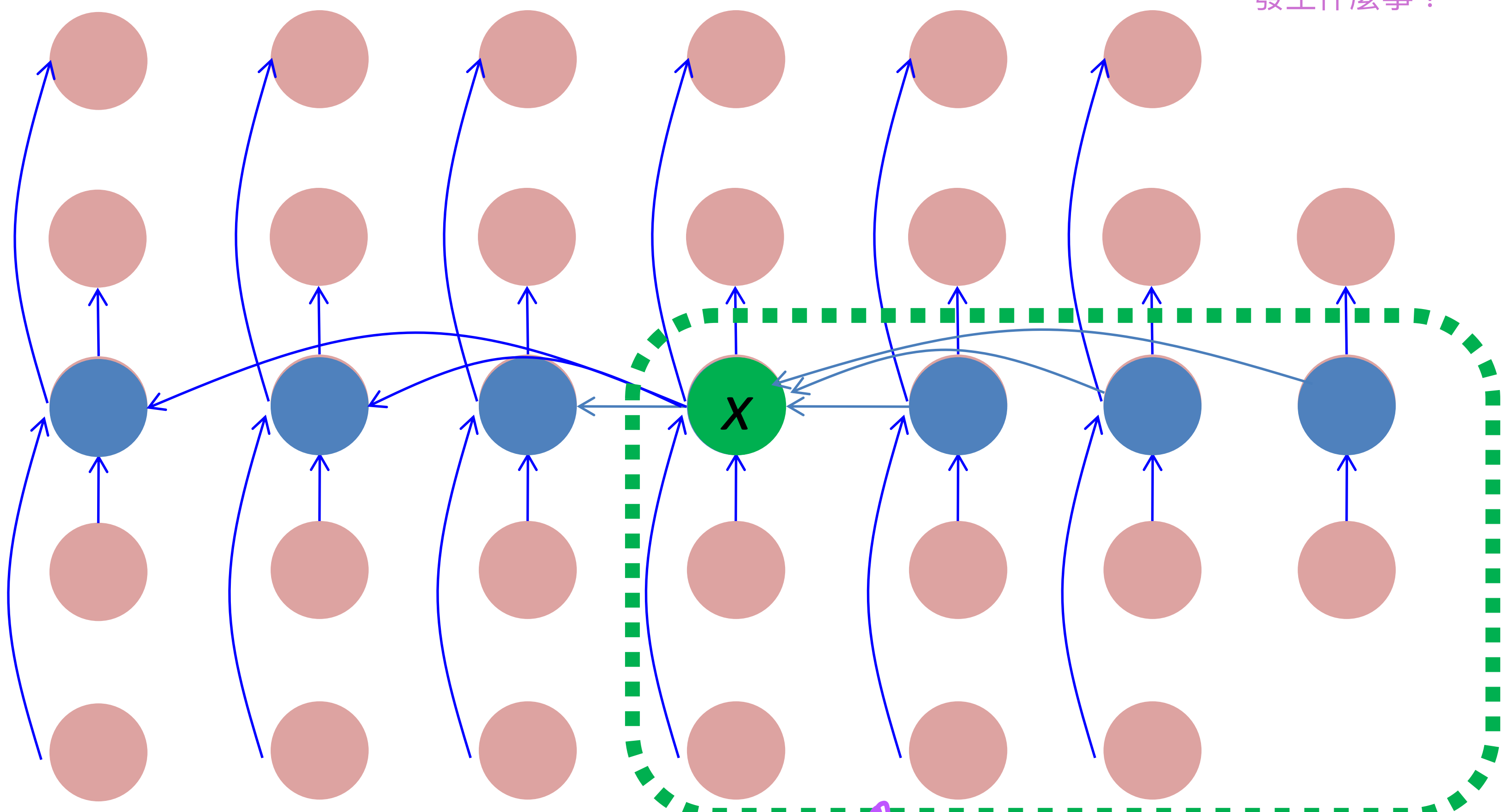
$\max(T(|X_<|), T(|X_>|))$
$= O(7n/10)$

Deleting at least 3n/10 elements!

Q.

為什麼一定要五個一組？

如果是三個、七個，會發生什麼事？

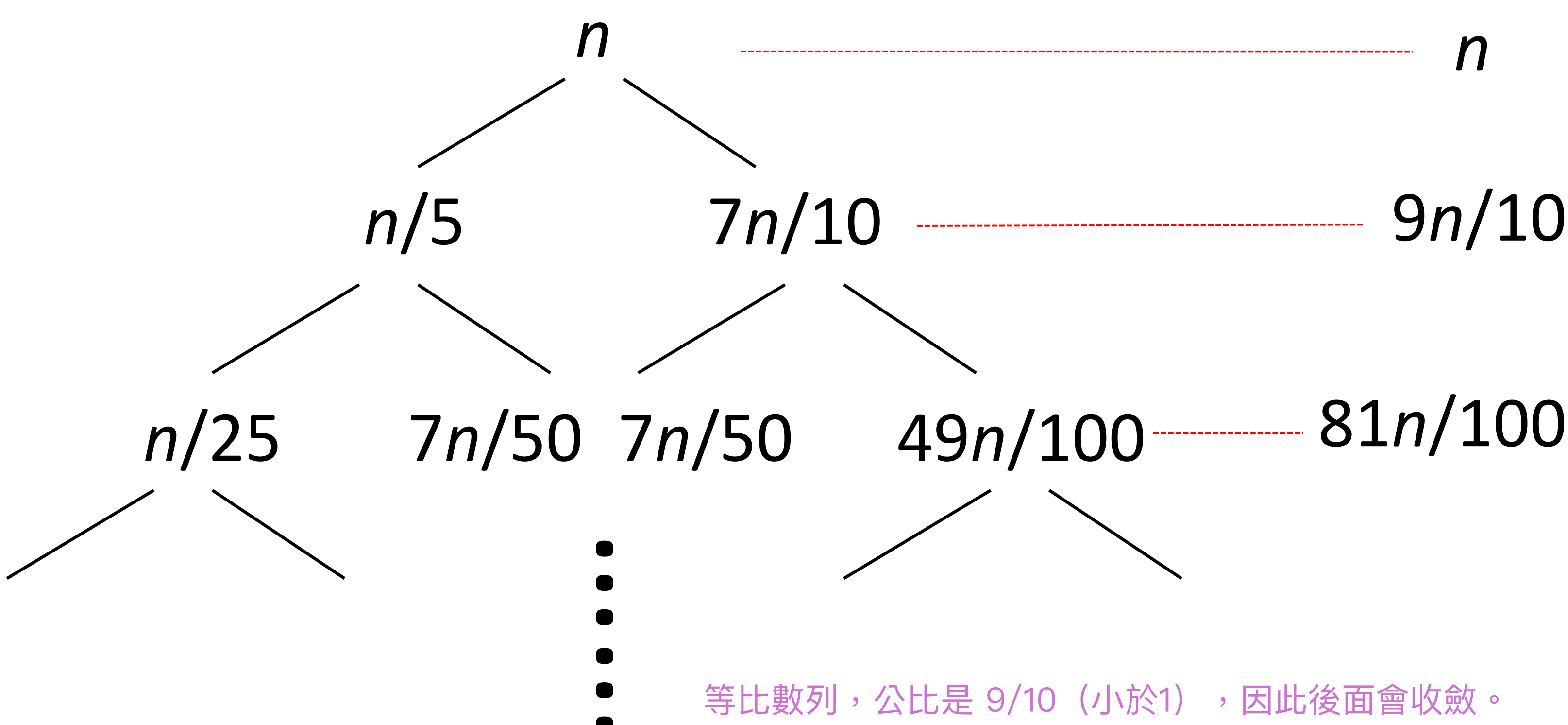


A.

如果三個一組：組數會太多，會跑出太多中位數，會花很多時間去sort。

如果是七個一組：每一組會太大，反而會花時間在組內排中位數。

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



等比數列，公比是 9/10（小於1），因此後面會收斂。
總和會是一個常數，故worst-case為 O(n)。

$$T(n) = n(1+9/10+81/100+\ldots) = O(n)$$