8.Sorting in linear time

線性時間排序法

中國文化大學
資訊工程學系
副教授　張耀鴻

112學年度第2學期

How fast can we sort? 排序可以多快?

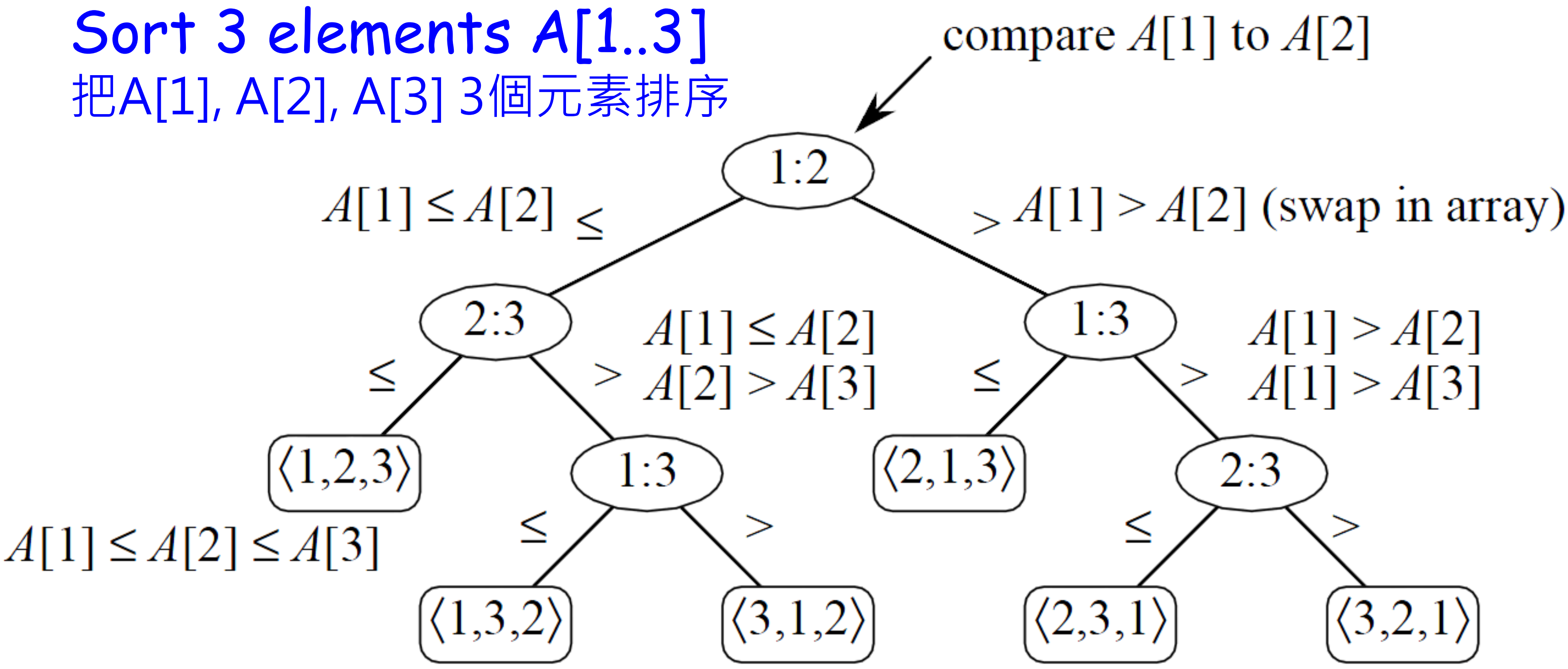
- All the sorting algorithms we have seen so far are ***comparison sorts***: only use comparisons to determine the relative order of elements.
  之前所介紹的排序法皆建立在「比大小」的基礎上
- *E.g., insertion sort, merge sort, quicksort,* heapsort.
- The best worst-case running time that we've seen for comparison sorting is $O(n \lg n)$.
  比大小方式排序理論上w.c.最快能在$O(n \lg n)$ 時間內完成
- ***Is $O(n \lg n)$ the best we can do?***
  問題是還有沒有更快的方法?
- ***Decision trees*** can help us answer this question.
  以下用決策樹來說明

8.1 Lower bound for sorting 排序之下界

The decision tree model 決策樹模型

Sort 3 elements A[1..3]

把A[1], A[2], A[3] 3個元素排序

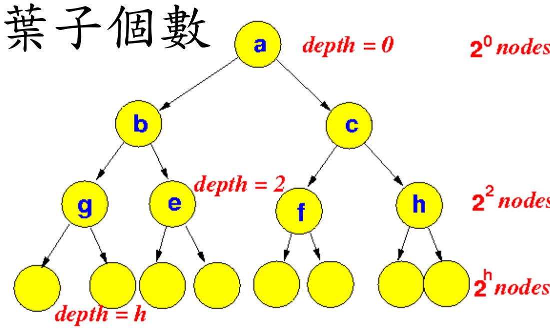


3個數比大小有3!=6種可能結果

決策樹最多有幾個葉子

Upper bound for # of leaves

- **lemma:** 高度為 h 的二元樹最多有 2^h 個葉子
  Any binary tree of height h has $\leq 2^h$ leaves
- i.e.,
  - ✓ ℓ = # of leaves 葉子個數
  - ✓ h = height 樹高
  - ✓ ➔ $\ell \leq 2^h$



- Proof: By induction on h. 以歸納法證明 $\ell \leq 2^h$
  以偏概全法
  Basis: h = 0: 樹有一節點, i.e., $2^h = 1$.
  Inductive Step: 假設高度小於 h-1 成立.
  #leaves for height $h = 2 \cdot 2^{h-1} = 2^h$

先入為主法
演一隻豬腳、跪那條培根
演譯（台語）歸納

最壞情況之下界

A lower bound for the worst case

> **Theorem 8.1.** Any comparison sort algorithm requires $\Omega(n \lg n)$ comparisons in the worst case.
>
> 以比大小方式排序理論上至少需要 $\Omega(n \lg n)$ 次比較運算

Proof:

1. The tree must contain $\geq n!$ *leaves, since* there are $n!$ *possible permutations.* N 個元素依大小順序排列有$n!$種可能. 因此決策樹至少有 $n!$ 個 leaves. (決策樹模型)
2. A height-h binary tree has $\leq 2^h$ *leaves.* 高度為h的二元樹最多可有2^h 個葉子. (完全二元樹特性)
3. Thus, $n! \leq 2^h$. 由1, 2 可得知: $n! \leq 2^h$

$$\begin{aligned} \therefore\ h &\geq \lg(n!) \\ &\geq \lg((n/e)^n) \\ &= n \lg n - n \lg e \\ &= \Omega(n \lg n) . \quad \square \end{aligned}$$

By Stirling's approximation or Equation (3.19)

比較排序法之下界

lower bound for comparison sorting

- **Corollary 8.2.** **Heapsort** and **MergeSort** are *asymptotically optimal* comparison sorting algorithms.

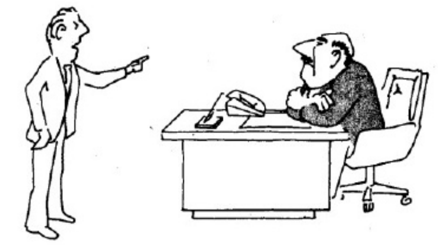
Heapsort 和 MergeSort理論上為比較排序問題中最佳化的演算法

Proof:

The upper bounds for HEAPSORT and MERGESORT is $O(n \lg n)$, which matches the $\Omega(n \lg n)$ worst-case lower bound from Theorem 8.1.

HeapSort和MergeSort的時間複雜度皆為$O(n \lg n)$,
而定理8.1說排序至少要做$\Omega(n \lg n)$ 次比較,
故得證.

最好的狀況當然是證明沒有好的方法存在。例如知名的 Sorting Problem 的 Lower Bound 是 O(n lg n).



我想不出好方法,
因為不可能有這種好方法！

命運必須靠自己創造

8.2 Counting sort 計數排序法 (ABC排序)

- **No comparisons** between elements. 無需比較
- Depends on a ***key assumption***: 基於鍵值假設(常數個)
  - ✓ numbers to be sorted are integers in {0, 1, 2, …, k}.
- • ***Input***: $A[1..n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
- • ***Output***: $B[1..n]$, **sorted**.
- • ***Auxiliary storage***: $C[0..k]$.

K為常數

假設 Array A 裡面的值
只有k個鍵值

Census (普查)

COUNTING_SORT(A,B,k)

let $c[0..k]$ be a new array

K有多大, C就有多大

for $i \leftarrow 0$ **to** k

  do $c[i] \leftarrow 0$

1.把 Array C 的內容初始化為 0

for $j \leftarrow 1$ **to** $A.length$

  do $c[A[j]] \leftarrow c[A[j]]+1$

2.c[i] 存放鍵值為 i 的有幾個

▶ $c[i]$ now contains the number of elements equal to i

for $i \leftarrow 1$ **to** k

  do $c[i] \leftarrow c[i]+c[i-1]$

3.再把 key ≤ i 的個數累加至 c[i]

▶ $c[i]$ now contains the number of elements less than or equal to i

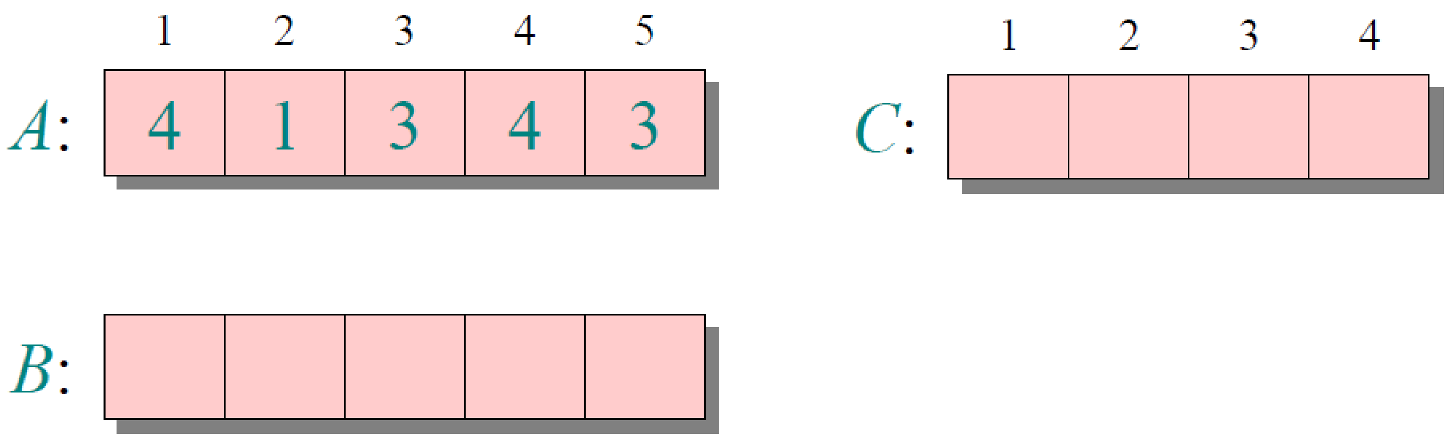
for $j \leftarrow A.length$ **downto** 1

  do $B[c[A[j]]] \leftarrow A[j]$

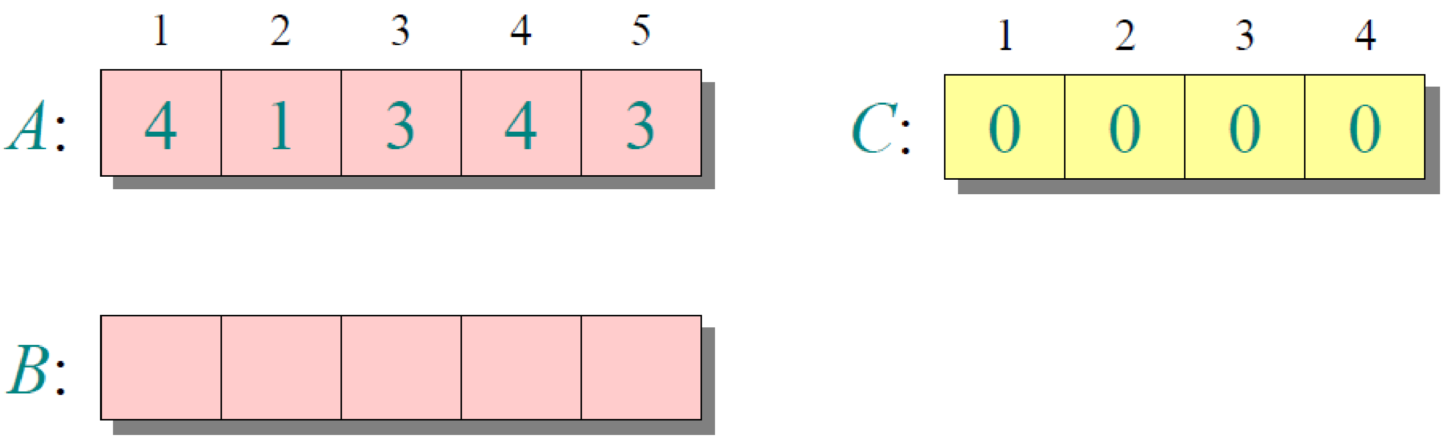
    $c[A[j]] \leftarrow c[A[j]]-1$

4.最後把 A[j] 內容
放到 B的第 c[A[j]] 個位置
(由後往前)

Counting sort example



Loop 1




```
for i ← 0 to k
    do c[i] ← 0
```

1.把 Array C 的內容全部設為 0

Loop 2

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
0	0	0	1

B:


```
for  j ← 1  to A.length
    do  c[A[j]] ← c[A[j]] + 1
```

2.先把 key = i 的個數放 c[i]

A:

1	2	3	4	5
4	1	3	4	3

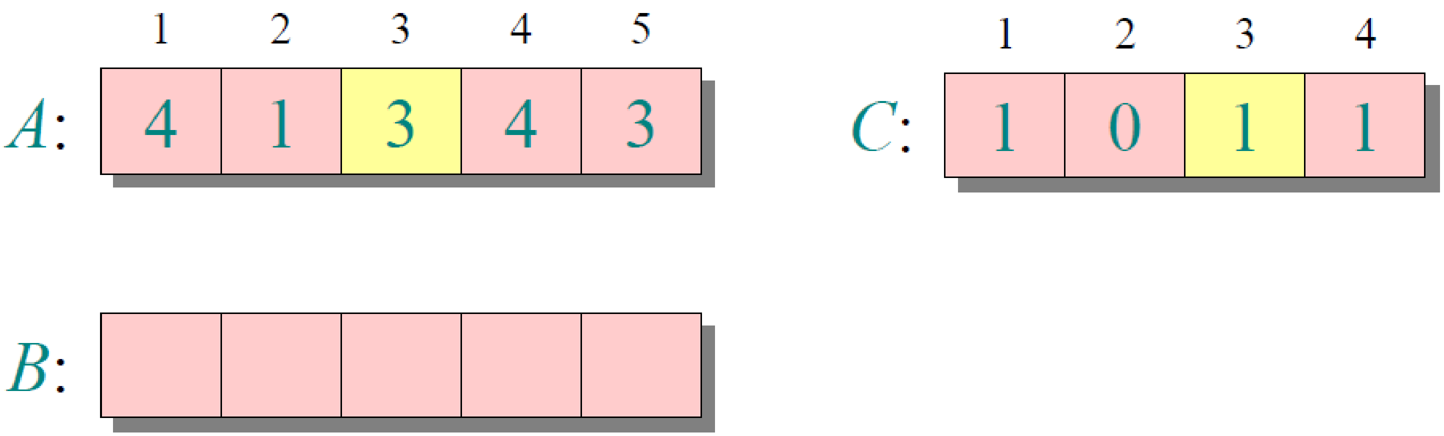
C:

1	2	3	4
1	0	0	1

B:

for $j \leftarrow 1$ **to** $A.length$

  do $c[A[j]] \leftarrow c[A[j]] + 1$




```
for  j ← 1  to A.length
     do  c[A[j]] ← c[A[j]] + 1
```

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	1	2

B:

for $j \leftarrow 1$ **to** $A.length$

  do $c[A[j]] \leftarrow c[A[j]] + 1$

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

	1	2	3	4
C:	1	0	2	2

B:					

```
for j ← 1 to A.length
    do c[A[j]] ← c[A[j]] + 1
```

Loop 3

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	2	2

B:

C':

1	1	2	2

for $i \leftarrow 1$ **to** k

  do $c[i] \leftarrow c[i] + c[i-1]$

3.再把 key ≤ i 的個數放 c[i]

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

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	2

for $i \leftarrow 1$ **to** k

  do $c[i] \leftarrow c[i] + c[i-1]$

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

	1	2	3	4
C:	1	0	2	2

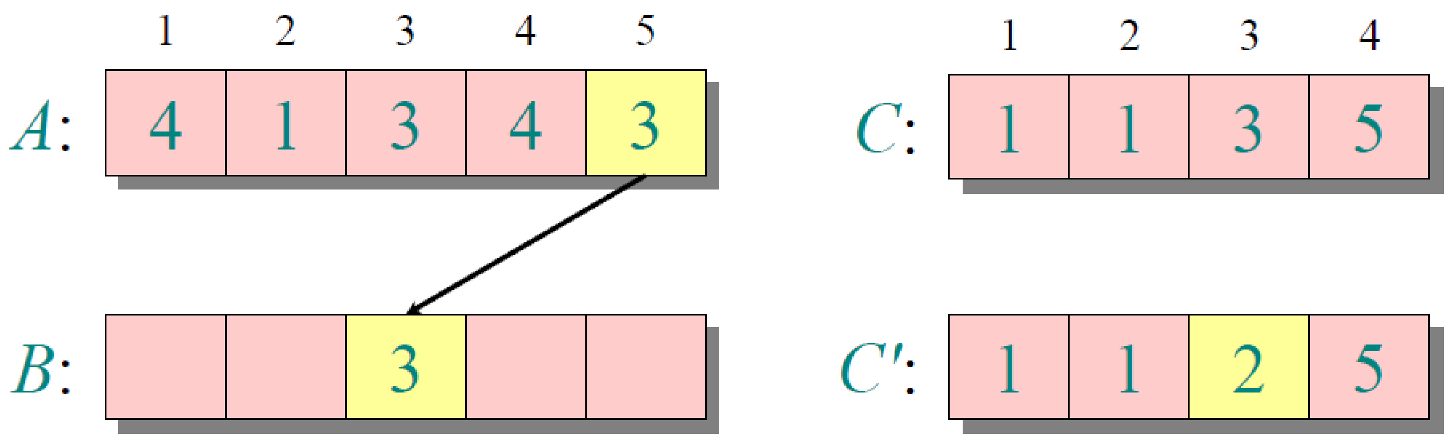
B:					

C':	1	1	3	5

for $i \leftarrow 1$ **to** k

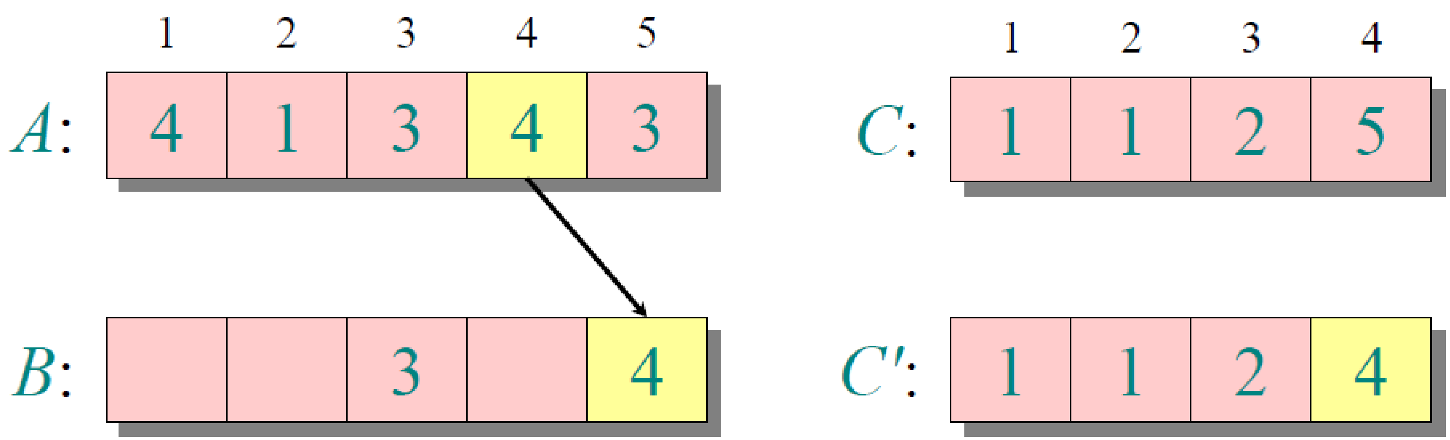
  do $c[i] \leftarrow c[i] + c[i-1]$

Loop 4



$$
\begin{aligned}
&\textbf{for } j \leftarrow A.length \ \textbf{ downto } 1\\
&\quad \textbf{do } B[c[A[j]]] \leftarrow A[j]\\
&\qquad c[A[j]] \leftarrow c[A[j]] - 1
\end{aligned}
$$

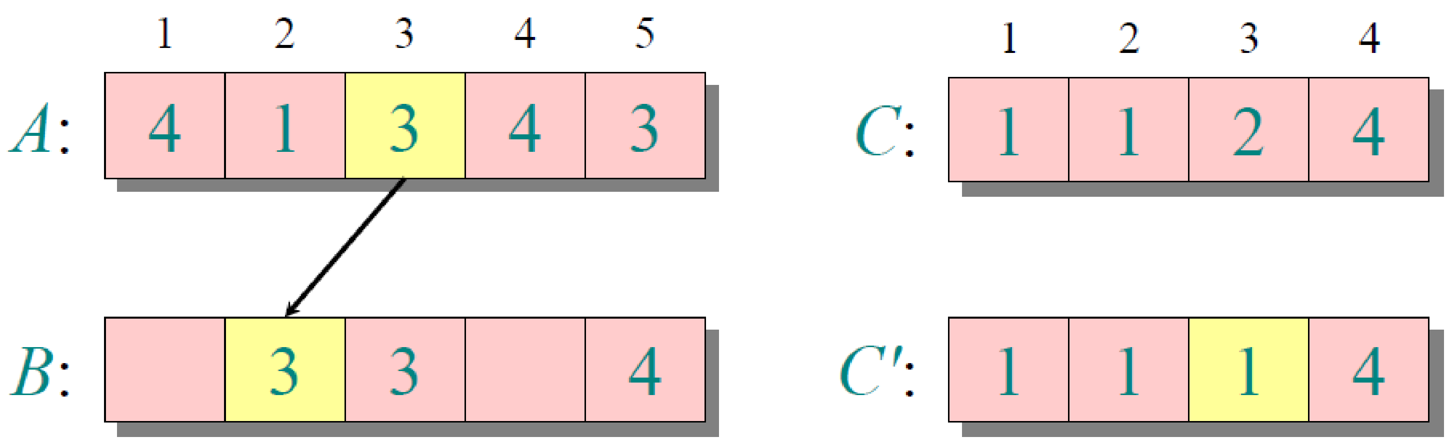
4.最後把 A[j] 放到 B 的第 c[A[j]] 個位置 (由後往前), i.e., 看A[j]的值是多少, 先到C陣列找到有幾個比目前的鍵值小的, 再複製到B陣列的那個位置



for $j \leftarrow A.length$ **downto** 1

  do $B[c[A[j]]] \leftarrow A[j]$

    $c[A[j]] \leftarrow c[A[j]] - 1$

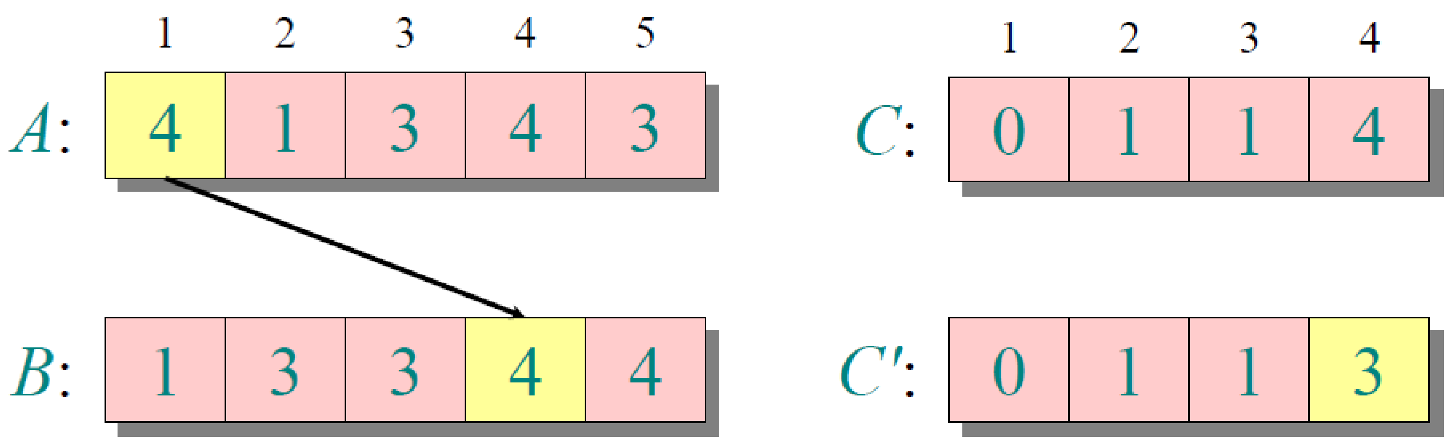



```
for j ← A.length downto 1
    do B[c[A[j]]] ← A[j]
       c[A[j]] ← c[A[j]] − 1
```

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

	1	2	3	4
C:	1	1	1	4
C':	0	1	1	4

```
for  j ← A.length  downto 1
    do  B[c[A[j]]] ← A[j]
        c[A[j]] ← c[A[j]] − 1
```




```
for j ← A.length downto 1
    do B[c[A[j]]] ← A[j]
       c[A[j]] ← c[A[j]] − 1
```

The operation of Counting-sort on an input array A[1..8]

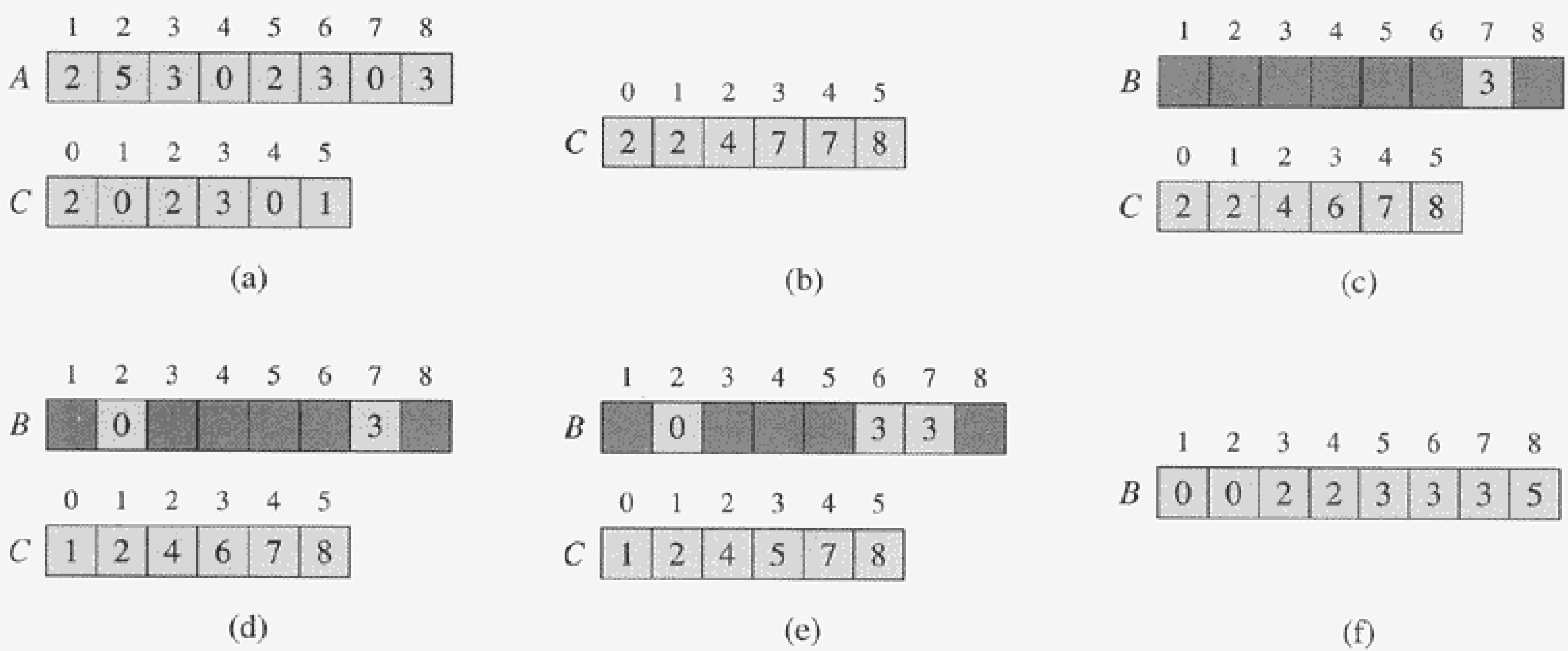


Figure 8.2 The operation of COUNTING-SORT on an input array $A[1\,..\,8]$, where each element of A is a nonnegative integer no larger than $k = 5$. (a) The array A and the auxiliary array C after line 4. (b) The array C after line 7. (c)–(e) The output array B and the auxiliary array C after one

時間複雜度分析:

COUNTING_SORT(A,B,k)

let $c[0..k]$ be a new array

for $i \leftarrow 0$ **to** k

  do $c[i] \leftarrow 0$ $\quad \Theta(k)$

for $j \leftarrow 1$ **to** $A.length$

  do $c[A[j]] \leftarrow c[A[j]] + 1$ $\quad \Theta(n)$

▶ $c[i]$ now contains the number of elements equal to i

for $i \leftarrow 1$ **to** k

  do $c[i] \leftarrow c[i] + c[i-1]$ $\quad \Theta(k)$

▶ $c[i]$ now contains the number of elements less than or equal to i

for $j \leftarrow A.length$ **downto** 1

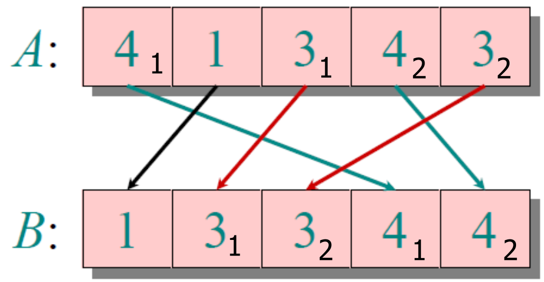
  do $B[c[A[j]]] \leftarrow A[j]$ $\quad \Theta(n)$

    $c[A[j]] \leftarrow c[A[j]] - 1$

Total: $\Theta(n+k)$

Discussion 穩定排序: 鍵值相同者, 排序前後出現順序一致

- Counting sort is a ***stable*** sort
  - ✓ Preserves input order among equal elements



key 一樣時 Sorting 前後出現的順序也會一樣.

- Running time: $\Theta(n)$ if $k = O(n)$
- Practical value of k ? K值多大時適用?
  - ✓ 32-bit ➔ 2^{32} = 4294927696 ➔ No way
  - ✓ 16-bit ➔ 2^{16} = 65536 ➔ Nah…
  - ✓ 8-bit ➔ 2^{8} = 256 ➔ May be (depending on n)
  - ✓ 4-bit ➔ 2^{4} = 16 ➔ Probably (unless n is really small)

基數排序(讀卡機排序)

8.3 Radix sort

利用 Counting-Sort 來排序

Used by the card-sorting machines you can now find only in computer museum.

Key idea:
從個位數開始用Stable-sort排序

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

RADIX_SORT(A,d)

for $i \leftarrow 1$ **to** d

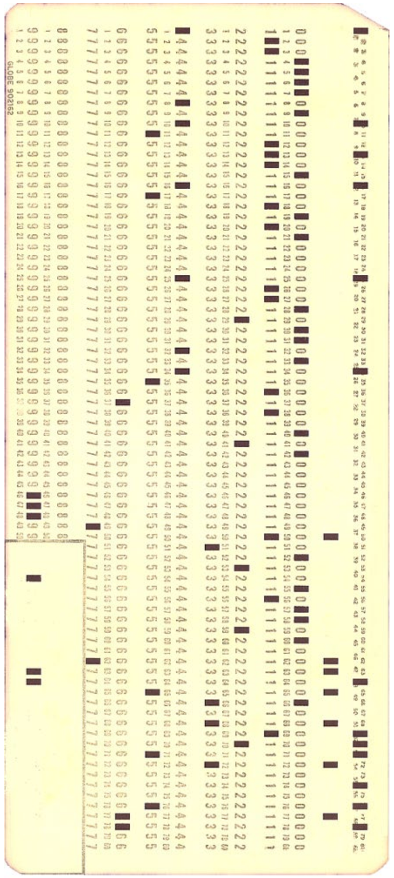
    do use a stable sort to sort array A on digit i

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

時間複雜度分析:

Analysis of Radix Sort:

1. Assume use counting sort as the auxiliary sort.
2. Θ(n+k) per pass
3. d passes
4. Θ(d(n+k)) total
5. If k = O(n), time = Θ(dn)
6. If d is constant, time = **Θ(n)**

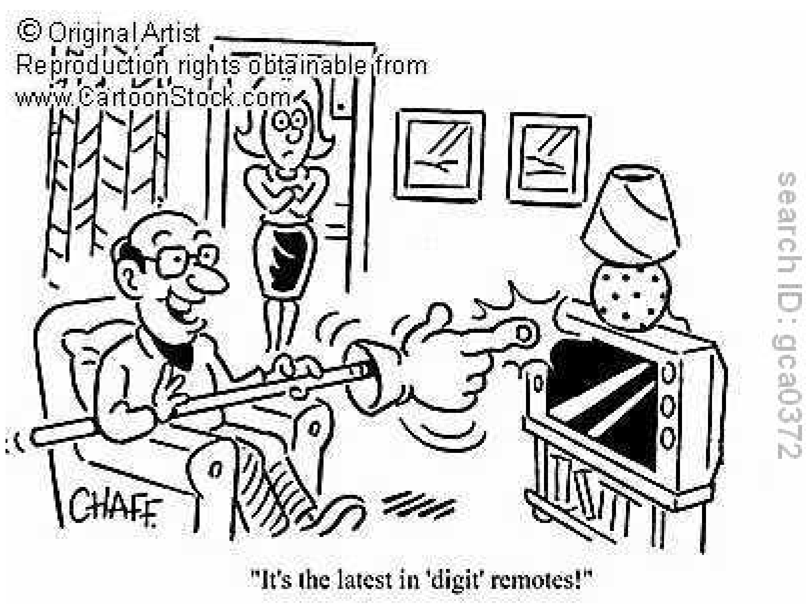


1. 假設以**CountingSort**為其附屬排序法
2. 每一位數要做 $\Theta(n+k)$ 次
3. d 位數需重覆以上步驟 d 次
4. 共需 $\Theta(d\,(n+k))$ 次
5. 若 k 值最多有 n 個, 則時間需 $\Theta(dn)$
6. 若 d 為常數, 則時間複雜度為 $\boldsymbol{\Theta(n)}$

Lemma 8.3

Given n d-digit numbers in which each digit can take on up to k possible values, **Radix-Sort** correctly sorts these number in $\Theta(d(n+k))$ time.

給定n個d位數的字碼, 其中每個字碼(位數)有k種可能值.
Radix-Sort 可在 $\Theta(d(n+k))$ 時間內將所有字碼排序.

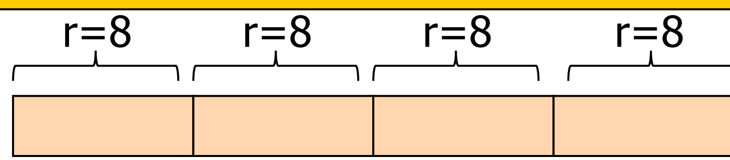


Lemma 8.4

Given n b-bit numbers and any positive integer $r \le b$, **Radix-Sort** correctly sorts these numbers in $\Theta((b/r)(n+2^r))$ time.

給定n個b-位元的數字, 其中每個數字切成 $d=b/r$ 段(每段r位元), **Radix-Sort** 可在 $\Theta((b/r)(n+2^r))$ 時間內將所有數字排序.

Example:



b個bits切成d段,
每一段有r個bits.

32-bit words, 8-bit digits. b=32, r=8,

$d=\lceil b/r\rceil=\lceil 32/8\rceil=4$, i.e., 做4 次 counting sort,

$k=2^r-1=2^8-1=255$. (意義: r個bit內的值最多可以有幾種變化)

Time = $\Theta((b/r)(n+2^r)) = \Theta(4(n+255)) = \Theta(n)$

Proof ：

1.Choose $d = \lceil b/r \rceil$ digits, r-bit each.

2.For each digit k, $0 \leqq k \leqq 2^r-1$

3.Use b/r passes of counting sort,

4.each pass takes $\Theta(n+k) = \Theta(n+2^r)$ time.

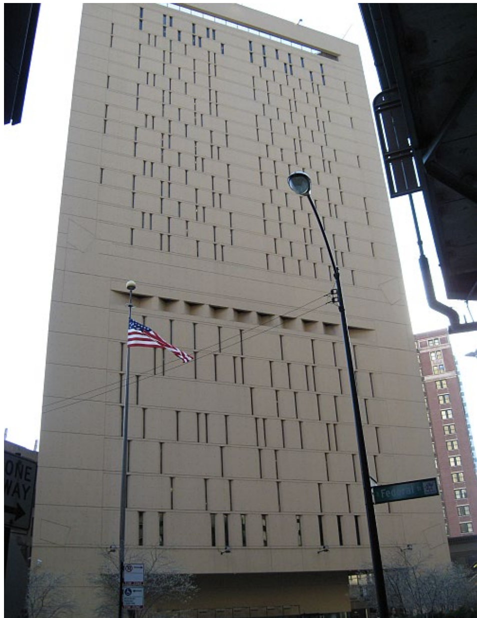
重點是每一段要選幾個bits?

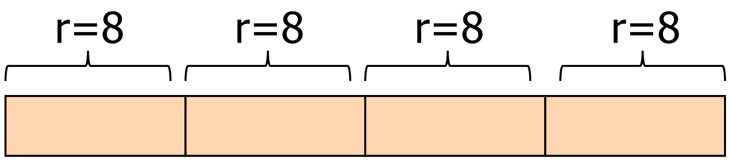
1.將原位元組切成 $d = \lceil b/r \rceil$ 段(位數)的字碼,

2.每一段(位數)有r-bit. 因此每一段(位數)
　的值k會落在0和2^r-1之間.

3.針對每一段執行 counting sort, 共 d=b/r 次.

4.每次需時 $\Theta(n+k) = \Theta(n+2^r)$ time.





Choosing r

Objective:

minimize $$T(n,b)=\Theta\left(\frac{b}{r}(n+2^r)\right)$$

例如：要將2^{16}個32-bit數字排序，
可選r = lg n = lg 2^{16} = 16 bits,
那麼d = $\lceil b/r \rceil$ = 2 passes

由觀察法，不希望 $2^r \gg n$

→ 令 max $r = \lg n$ 可代入上式可得

→ $T(n,b) = \Theta((b/\lg n)\ (n+n)) = \Theta(bn / \lg n)$

若b位數字介於 0 到 2^d-1 之間，其中 d 為常數，

則 $b = d \lg n$ ($\because d = \lceil b/r \rceil$ and $r = \lg n$)

→ radix sort runs in $\Theta(dn)$ time.

Radix-Sort vs. MergeSort and QuickSort

- 排序1 百萬(2^{20})個32-bit整數
- Radix-Sort: $d = \lceil b/r \rceil$
  $= \lceil 32/20 \rceil = 2$ passes

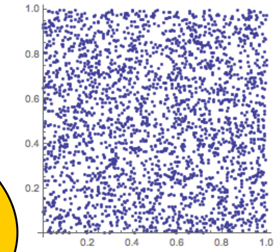
  每個 pass 又包括 census 和 copy data.
- MergeSort: lg n = 20 passes
- QuickSort: lg n = 20 passes (randomized)

- 結論:
  - ✓ **Radix-Sort** 假設排序鍵值(key)為d-digit數字(碼)
  - ✓ 利用 **Counting-Sort** 來取得 key 的資訊,
  - ✓ 所取得的 key 值用來當做 array indices,
  - ✓ 而不是直接拿2個 keys 來比較.

快樂水桶排序法

8.4 Bucket sort

假設輸入資料分佈為
random uniform distribution [0, 1).
([0,1)區間的隨機均勻分佈)



1. 把 [0,1)切成n個水桶.
2. 把n個輸入值分到這些水桶裡.
3. 對每個水桶做排序.
4. 把每個水桶內容依序印出.

BUCKET_SORT(A)

$n \leftarrow length[A]$

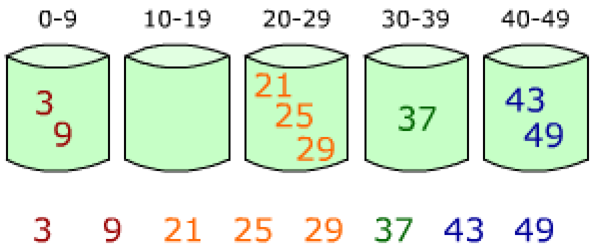
for $i \leftarrow 1$ **to** n

    do insert $A[i]$ into list $B\lfloor nA[i] \rfloor$

for $i \leftarrow 1$ **to** $n\text{-}1$

    do sort list $B[i]$ with insertion sort

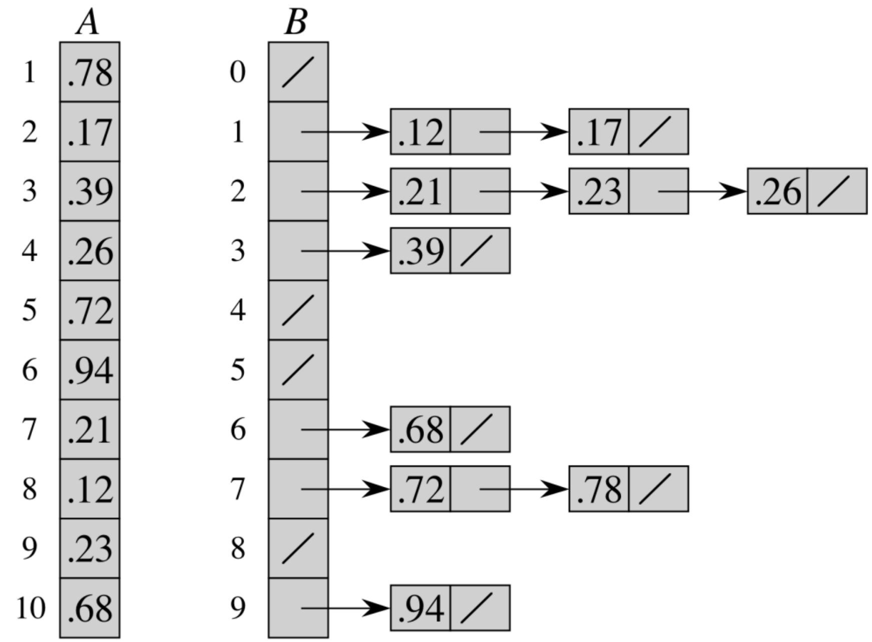
concatenate $B[0], B[1], \ldots, B[n-1]$ together in order



快樂水桶排序法

8.4 Bucket sort

假設輸入資料為
random uniform distribution [0, 1).
([0,1)區間的隨機均勻分佈)



(a)

BUCKET_SORT(A)

$n \leftarrow length[A]$

for $i \leftarrow 1$ **to** n

  do insert $A[i]$ into list $B\lfloor nA[i] \rfloor$ $\Theta(n)$

for $i \leftarrow 0$ **to** n-1

  do sort list $B[i]$ with insertion sort $\sum_{i=0}^{n-1} O(n_i^2)$

concatenate $B[0], B[1], \ldots, B[n-1]$ together in order $\Theta(n)$

(Hairy) Analysis

bucket sort 的執行時間:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2).$$

等號兩邊取期望值, 利用期望值的線性特性, 可得:

$$\begin{aligned} E[T(n)] &= E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \\ &= \Theta(n) + \sum_{i=0}^{n-1} O[E(n_i^2)] \end{aligned}$$

Claim 宣稱

$$E[n_i^2] = 2 - 1/n$$

Proof of claim

定義 indicator random variables

X_{ij} = I {*A*[*j*] falls into bucket *i*} 第 j 個元素掉到第 i 個水桶

for i = 0, 1, …, n-1 and j = 1, 2,…,n.

thus,

$$n_i = \sum_{j=1}^{n} X_{ij}.$$

第 i 個水桶內的元素個數

= (第 1 個元素掉到第 i 個水桶 or

第 2 個元素掉到第 i 個水桶 or

……

第 n 個元素掉到第 i 個水桶)

$$E[n_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$= E\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right]$$

$$= E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n\\ k\ne j}} X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n\\ k\ne j}} E[X_{ij}X_{ik}],$$

(第 ***i*** 個水桶內的元素個數)^2

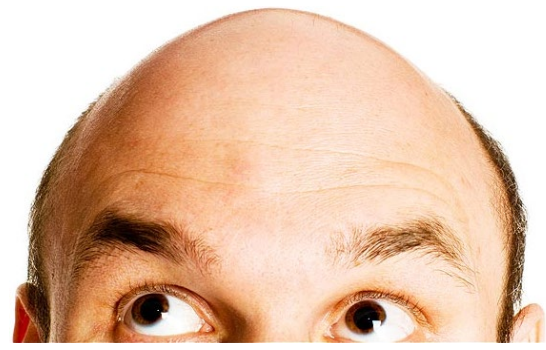
=(第 ***j*** 個元素掉到水桶 ***i*** and

(1st元素掉到水桶 ***i*** or

2nd元素掉到水桶 ***i*** or

……

kth 元素掉到水桶 ***i***))



第j個元素掉進第i個水桶 Xij=1

Indicator random variable X_{ij} = 1 的機率為 1/n，其他情況為 0，因此

$$E[X_{ij}^2] = 1^2 \cdot \frac{1}{n} + 0^2 \cdot \left(1 - \frac{1}{n}\right)$$

$$= \frac{1}{n}$$

第j個元素沒掉進第i個水桶 Xij=0

當 $k \neq j$, X_{ij} 與 X_{ik} 為獨立隨機變數, 因此(具有線性特性)

$$E[X_{ij}X_{ik}] = E[X_{ij}]E[X_{ik}]$$

$$= \frac{1}{n} \cdot \frac{1}{n}$$

$$= \frac{1}{n^2}.$$

$$E[n_i^2] = \sum_{j=1}^{n} \frac{1}{n} + \sum_{1 \le j \le n} \sum_{\substack{1 \le k \le n \\ k \ne j}} \frac{1}{n^2}$$

$$= n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2}$$

$$= 1 + \frac{n-1}{n}$$

$$= 2 - \frac{1}{n}$$

Therefore,……

$$
\begin{aligned}
E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \\
&= \Theta(n) + \sum_{i=0}^{n-1} O\left(2 - \frac{1}{n}\right) \\
&= \Theta(n) + n \cdot O\left(2 - \frac{1}{n}\right) \\
&= \Theta(n) + O(n) \\
&= \Theta(n)
\end{aligned}
$$

結論: the expected time for bucket sort is

$\Theta(n)+n\cdot O(2\text{-}1/n)= \Theta(n)$.

線性!

Discussion

- **Bucket-Sort** 為 non-comparison sort
- 以key值將array元素分發到水桶裡
- 本章以probabilistic analysis來分析演算法
  - ✓ 執行時間取決於 input distribution
  - ✓ **Bucket-Sort**分析之前提為假設輸入key值分佈為 uniform distribution [0,1)
- 不同於randomized algorithm (RA)之分析方式
  - ✓ **Randomized-Hire-Assistant** (Ch5)
  - ✓ **Randomized-QuickSort** (Ch7)
  - ✓ **RA之執行時間與 input distribution 無關**

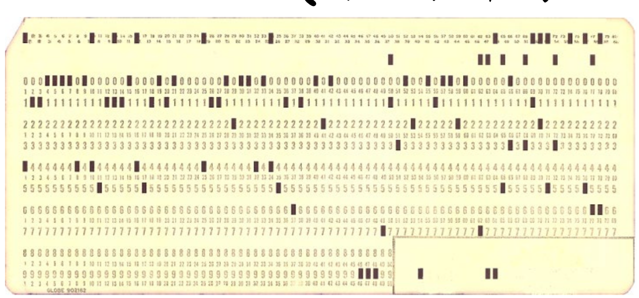
Summary

- Decision tree model
  - ✓ #leaves of binary tree = n! (n個數比大小)
- Comparison sort(需比較)
  - ✓ Theorem 8.1: 基於比較的排序法之下限 $\Theta(n \lg n)$
  - ✓ Theorem 8.2: Asymptotically optimal algorithms
- Non-Comparison sort(無比較)
  - ✓ **Counting-Sort**: (ABC 排序法) $\Theta(n+k)$
  - ✓ **Radix-Sort**: (digit排序法) $\Theta(\ d(n+k)\ ) = \Theta(\ (b/r)(n+2^r)\ )$
  - ✓ **Bucket-Sort**: (水桶排序法) $\Theta(n)$

鍵值個數

鍵值位數

Average case
(假設輸入為 U{0,1})




"THE FUTURE'S NOT SET.
THERE'S NO FATE
BUT WHAT WE MAKE FOR OURSELVES..."
- Sarah Connor
Virux711

"NO!

Try not!

DO or DO NOT,

There is no try."