SORTING IN LINEAR TIME

Michael Tsai

2017/05/09

Sorting Algorithm in Linear Time?

- Comparison-based (比較元素然後交換其順序) sorting的complexity lower bound為$\Omega(n \log n)$

- 如何突破這個障礙?
- 不要使用comparison-based的方法
- 因此可達到$O(n)$

- 需要利用額外的假設

- 通常是"以空間換取時間"

例子：電話排序問題

- 電話排序假設全部電話號碼有s個種可能 (所有排列組合)
- 我們有n組電話要排
- 則使用超大表格排序法需要O(s)的時間 (空間換取時間)

Input: {20000002, 89999999, 20000000, ...}

共s個

Index	Array
20000000	有
空	空
20000002	有
...	...
...	...
89999999	有

例子：電話排序問題

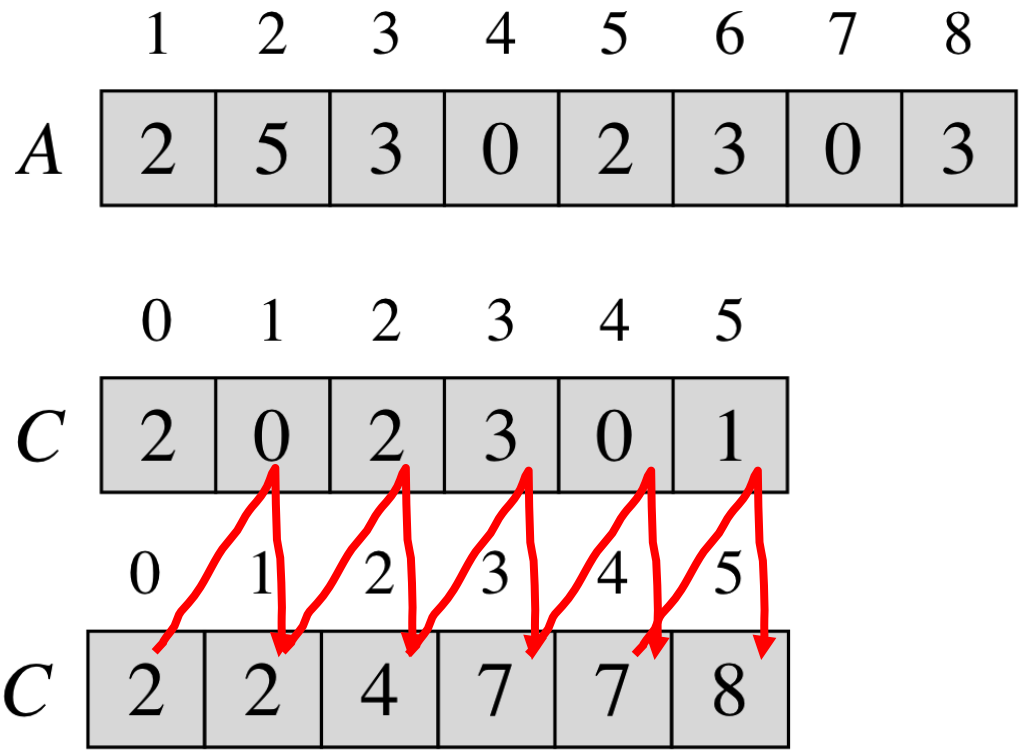
- 如果使用comparison-based sorting平均需要$O(n \log n)$
- 假設超大表格排序法所需時間為$c_1 s$
- 假設comparison-based排序法所需時間為$c_2 n \log n$
- 假設$10c_1 = c_2$
- break even point為 s= 10 n log n
- 假設以台北市電話號碼為例, $s = 10^8$
- $10^8 = 10\, n \log n$, n<10^8
- break even point大約為 n=k×10^6
- 當n大於此數則non-comparison sorting比較快

Counting Sort

假設: 知道可能出現的所有input種類個數K (而且$K = O(n)$)

Input array

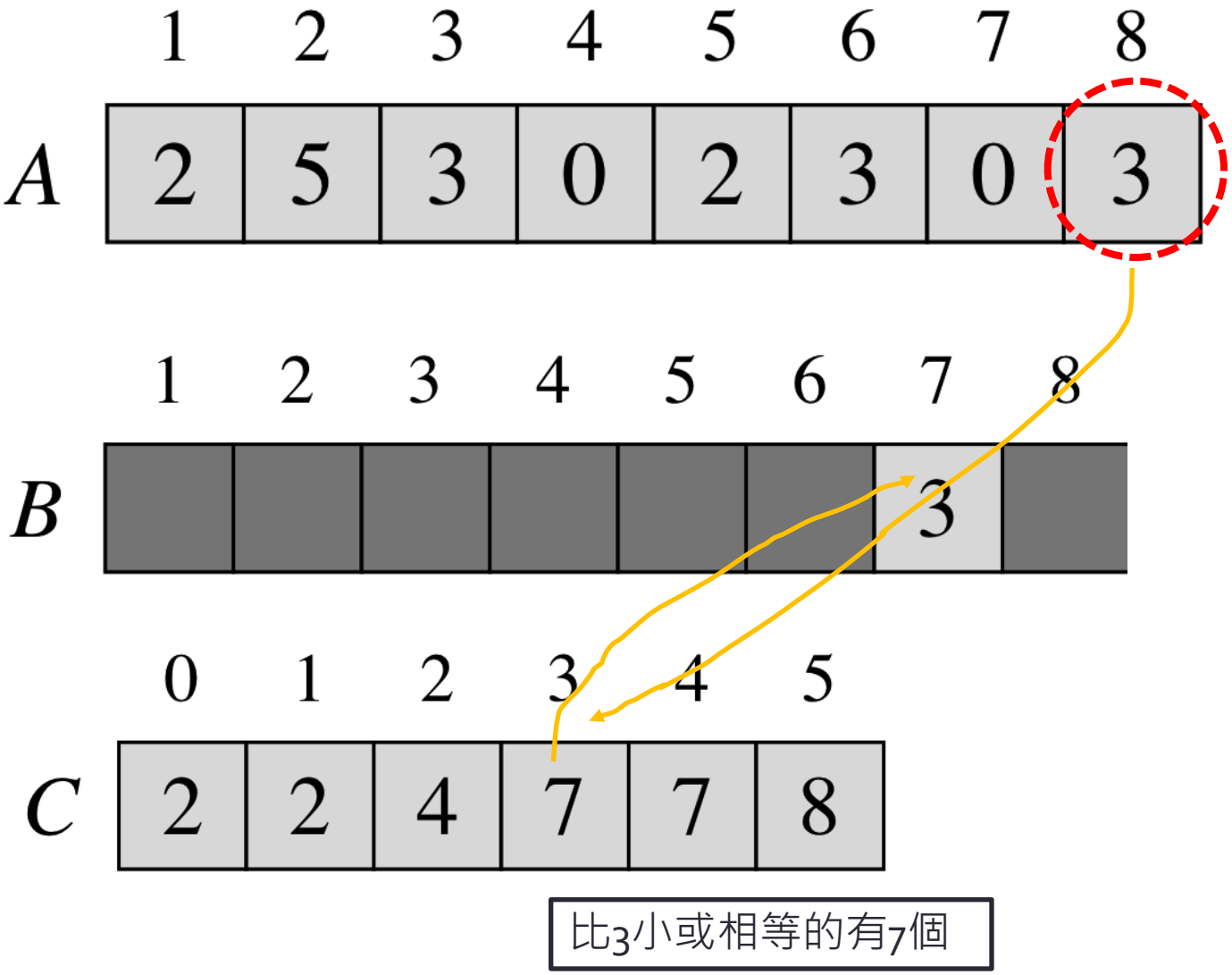
input只會出現{0,1,2,3,4,5}



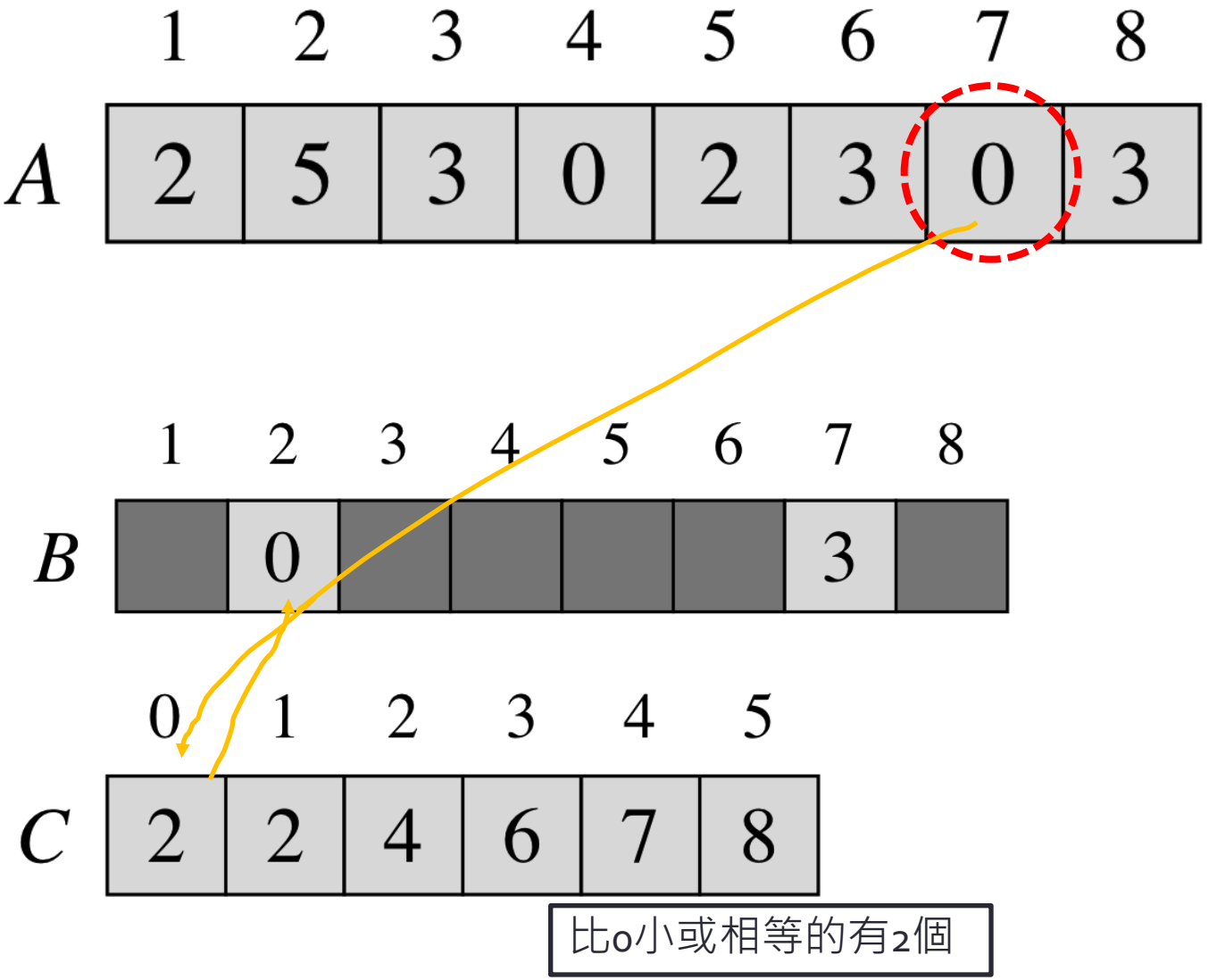
Counting: 算每個數字出現了幾次

"Cumulative Mass Function"
算小於或等於index的數字出現了幾次

Counting Sort

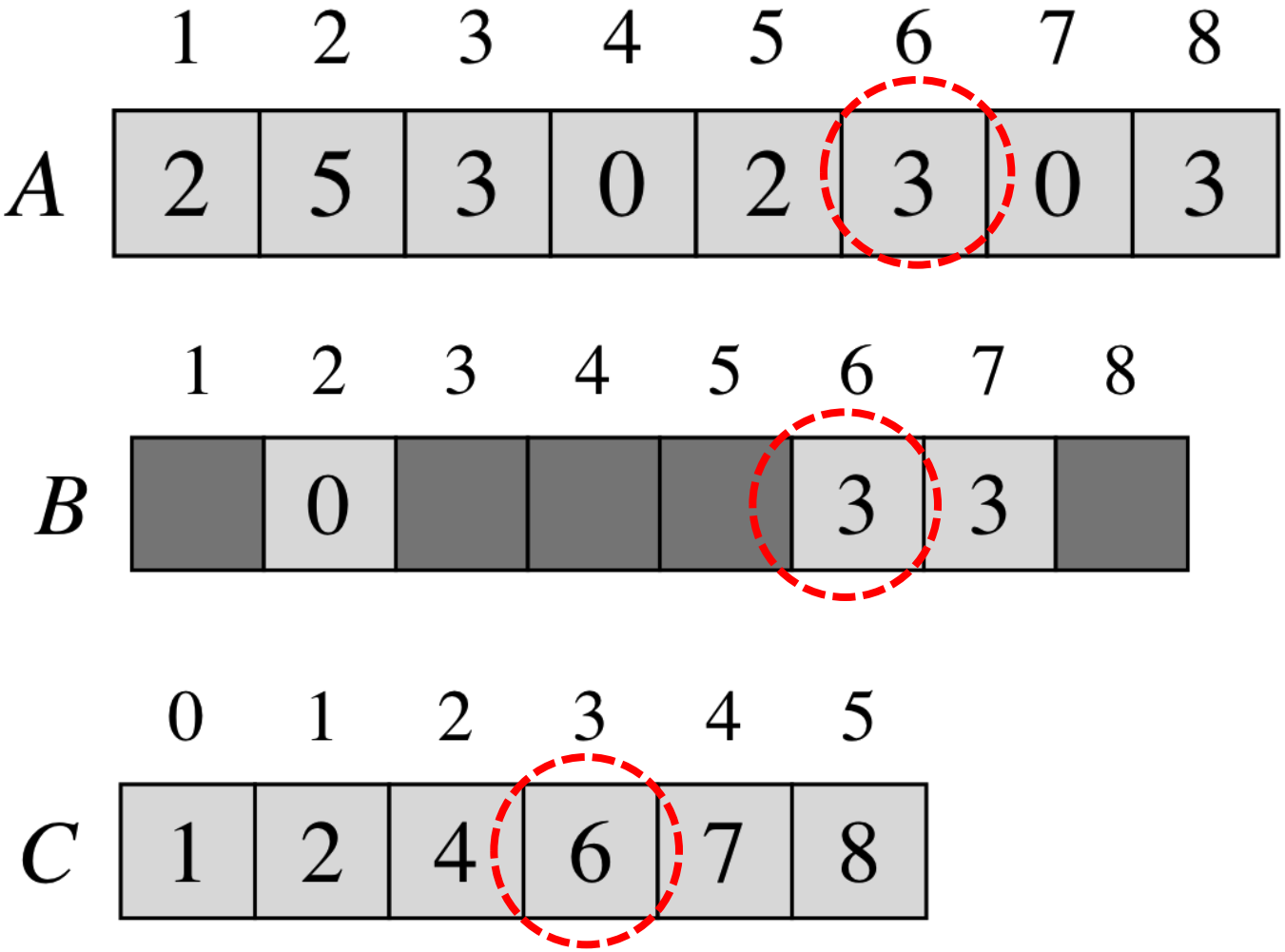


Counting Sort



Counting Sort

注意Counting Sort為stable sort. 因為是從最後面依序放入output array, 因此同樣大小的元素會依原本input array中順序放入.



Counting Sort

A: input array
n: input總個數
B: output array
K: 可能出現的input element總數

```
void CountingSort (int A[], int n, int B[], int K) {
      int C[K], i, j,;
      for (i=0;i<=K;i++)
            C[i]=0
      for (j=1;j<=n;j++)
            C[A[j]]=C[A[j]]+1;
      for (i=1;i<=K;i++)
            C[i]=C[i]+C[i-1];
      for (j=n;j>=1;j--) {
            B[C[A[j]]]=A[j];
            C[A[j]]--;
      }
}
```

Counting Sort

A: input array
n: input總個數
B: output array
K: 可能出現的input element總數

```
void CountingSort (int A[], int n, int B[], int K) {
      int C[K], i, j,;
O(K)  for (i=0;i<=K;i++)
            C[i]=0
O(n)  for (j=1;j<=n;j++)
            C[A[j]]=C[A[j]]+1;
O(K)  for (i=1;i<=K;i++)
            C[i]=C[i]+C[i-1];
O(n)  for (j=n;j>=1;j--) {
            B[C[A[j]]]=A[j];
            C[A[j]]--;
      }
}
```

Counting: 數每種element共幾個

"CMF": 數比每種element小或相等的共幾個

從input array最後一個開始依序放入output array

$O(n+K)=O(n)$

Sorting on several keys

- 假設K有很多個sub-key
- $K = (K_1, K_2, \ldots, K_r)$

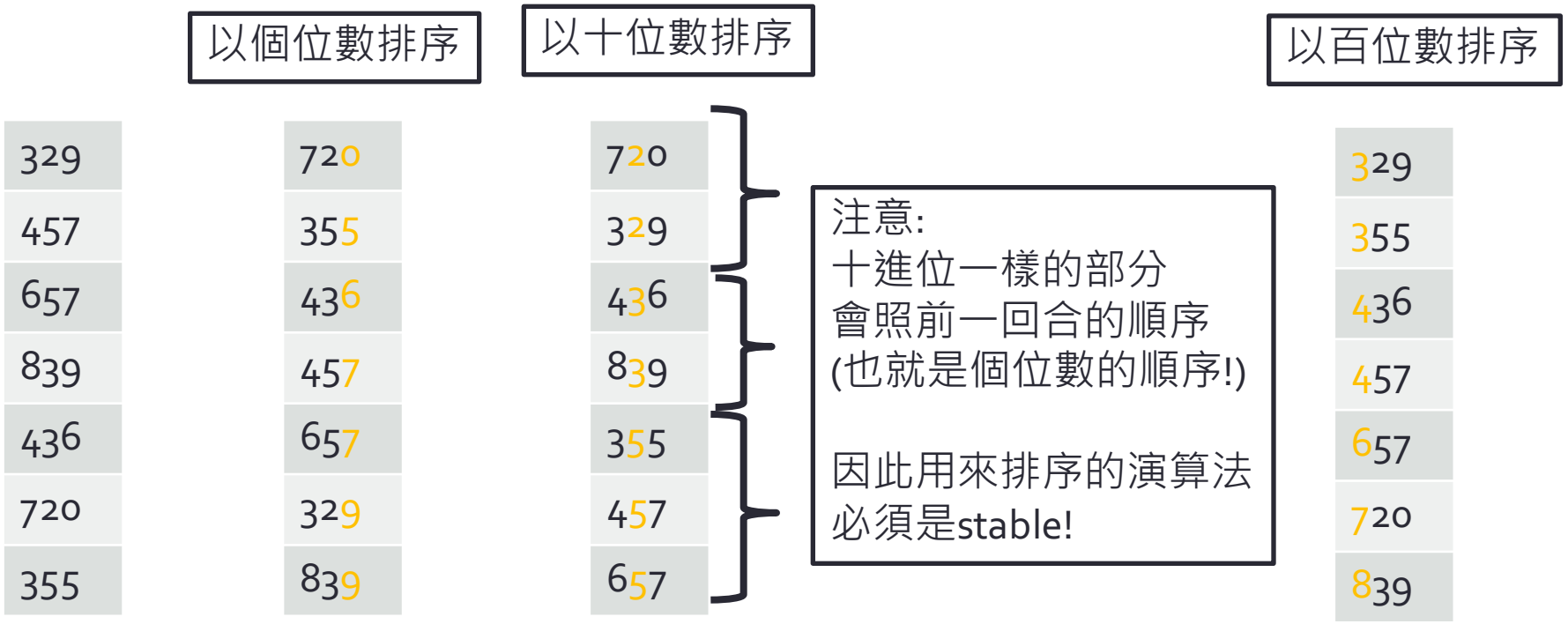
Most significant key　　Least significant key

- 則$K_x \leq K_y$ iff
- $K_{x,i} = K_{y,i}, 1 \leq i \leq j \; and \; K_{x,j+1} < K_{y,j+1}$ for some $j < r$, or
- $K_{x,i} = K_{y,i}, 1 \leq i \leq r$

- 則我們可以有以下的sorting 方法.
- Most Significant Digit first (MSD) sorting
  - 先依照most significant key sort, 然後依序往least significant key sort過去
- Least Significant Digit first (LSD) sorting
  - 先依照least significant key sort, 然後依序往most significant key sort過去

Radix Sort

假設:有"多個key"



先依照least significant digit sort, 然後依序往most significant key sort
過去: Least Significant Digit first (LSD) sorting

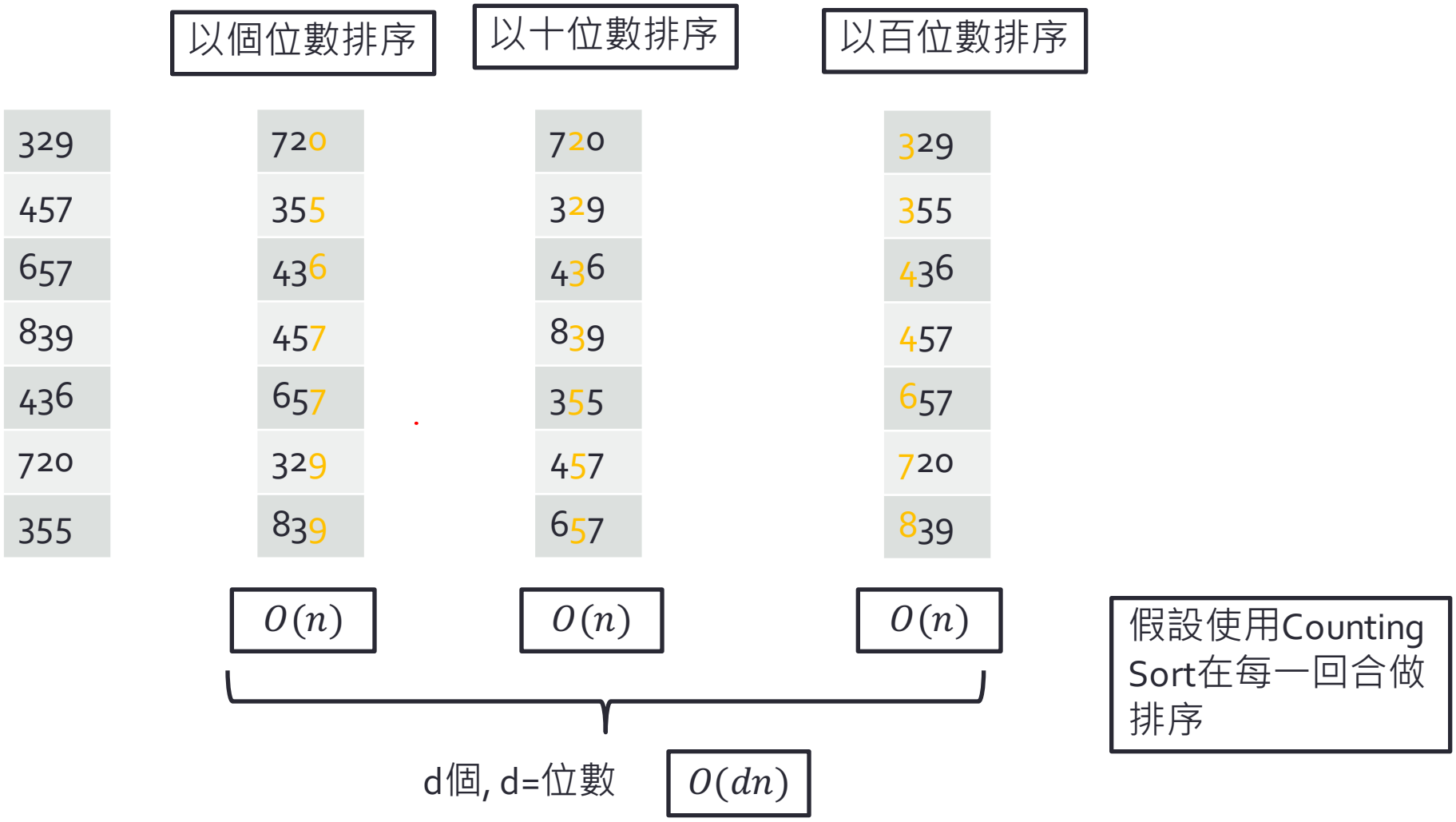
Radix Sort

如果先依照most significant key sort, 然後依序往least significant key sort過去: Most Significant Digit first (MSD) sorting?

	以百位數排序	以十位數排序	正確的做法必須切分成小的subset再去排序! (因此MSD比較麻煩)
329	329	329	329
457	355	720	355
657	457	436	436
839	436	839	457
436	657	355	657
720	720	457	720
355	839	657	839

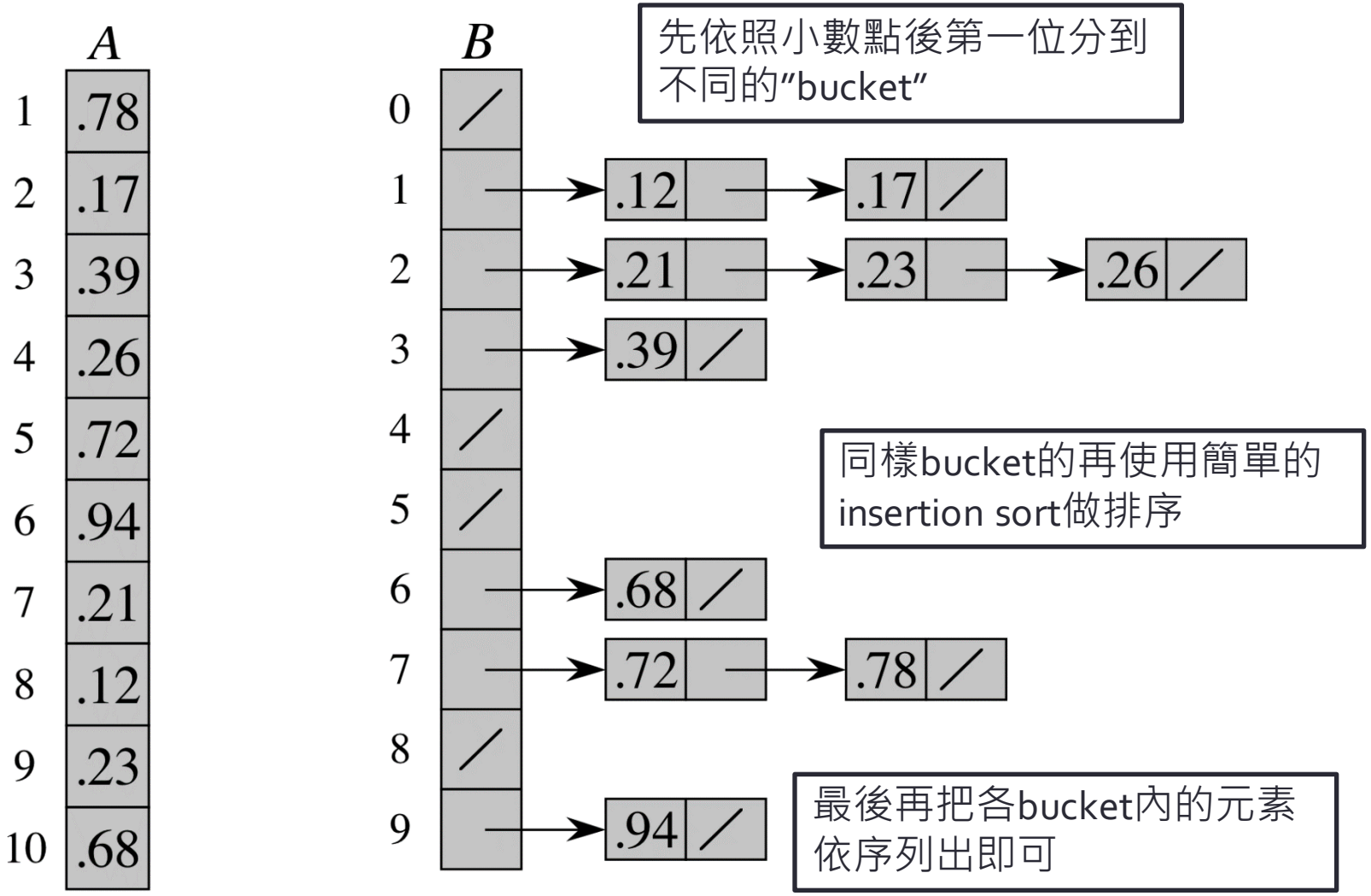
如果直接使用接下來的十位數排序會亂掉!

Radix Sort


以個位數排序
以十位數排序
以百位數排序
329
457
657
839
436
720
355
720
355
436
839
329
457
657
720
329
355
436
457
657
839
$O(n)$
$O(n)$
$O(n)$
d個, d=位數
$O(dn)$
假設使用Counting Sort在每一回合做排序

Bucket Sort

假設: input是從uniform distribution取出來的



Bucket Sort

- 假設: input是從uniform distribution取出來的
- 因此分到各個bucket的數量會差不多.

- 大略計算一下所需的時間:

走過每個bucket的時間

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

總時間

每個bucket內insertion sort所需時間

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

$$E\left[n_i^2\right] = 2 - \frac{1}{n}$$

(證明過程請見Cormen p202-203)

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

Related Course Book Chapter

- Cormen 8.2, 8.3, 8.4