Sorting in Linear Time

8.1 Lower bounds for sorting

- 本節探討排序所耗用的時間複雜度下限。

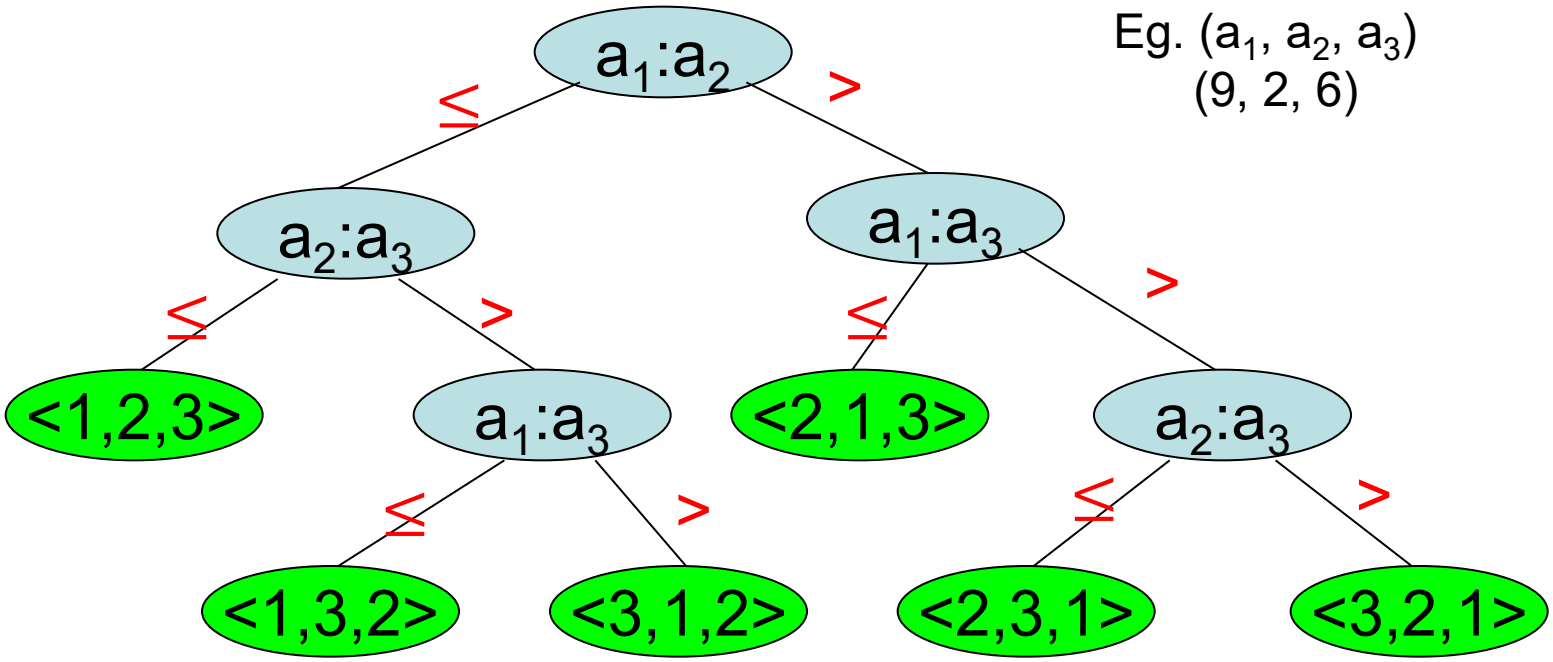
- 任何一個以比較為基礎排序的演算法，排序n個元素時至少耗用$\Omega(n\log n)$次比較。 是以時間複雜度至少為$\Omega(n\log n)$。

- 但不使用比較為基礎 的排序演算法，在某些情形下可在O(n)的時間內執行完畢。

Decision-Tree Model

- 一個以比較為基礎的排序演算法可以按照比較的順序建出一個Decision-Tree。
- 每一個從Root到Leaf的路徑都代表一種排序的結果。
- 任何一個以比較為基礎排序n個元素的演算法，所對應的Decision-Tree高度至少有$\Omega(n \log n)$。

Decision-Tree Model



Decision-Tree Model

- 證明：因為可能有$n!$種可能的排序結果，故對應的Decision tree至少有$n!$個leaf nodes。而高度為h的二元樹最多有2^h個leaf nodes。
  因此$h \geq \log_2(n!) = \Theta(n\log n)$。(後者由Stirling's approximation得證：$n! > (n/e)^n$)

- Heapsort與Mergesort是asymptotically optimal之比較排序法。

8.2 Counting Sort

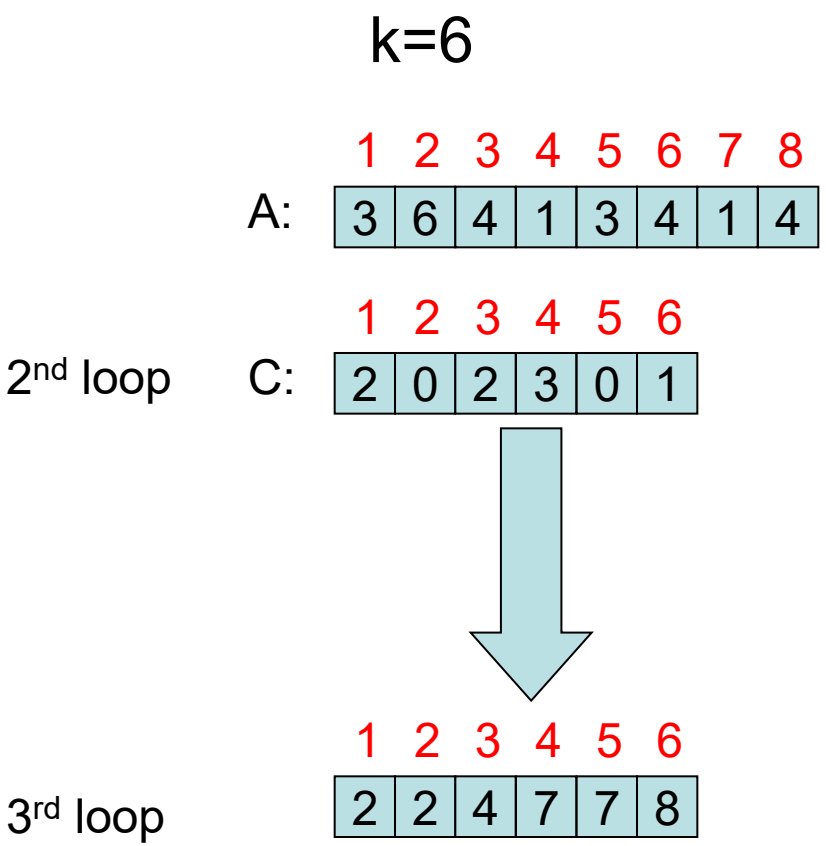
- Counting Sort (記數排序法) 不需要藉由比較來做排序。
- 必須依賴一些對於待排序集合中元素性質的假設。(如：所有待排序元素均為整數，介於1到k之間)
- 時間複雜度：$O(n+k)$
- 主要的關鍵在於，統計1到k之間每個數值出現過幾次，然後想辦法將排序好的數列輸出。

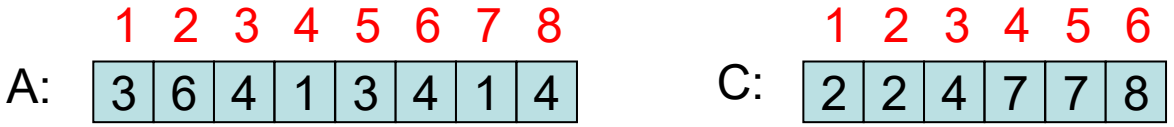
Input: A[1..n], where A[j]∈{1,2,…,k}

Output:B[1..n], sorted

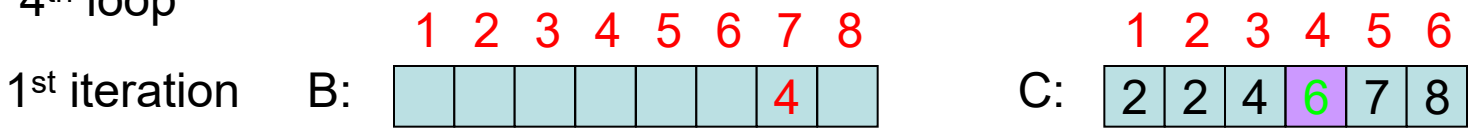
COUNTING-SORT(A, B, k)

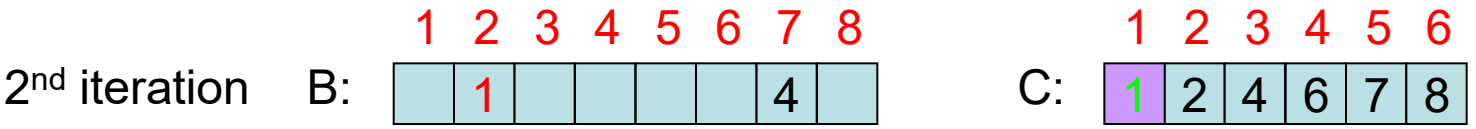
```
1   let C[0..k] be a new array
2   for i = 0 to k
3       C[i] = 0
4   for j = 1 to A.length
5       C[A[j]] = C[A[j]] + 1
6   // C[i] now contains the number of elements equal to i.
7   for i = 1 to k
8       C[i] = C[i] + C[i − 1]
9   // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

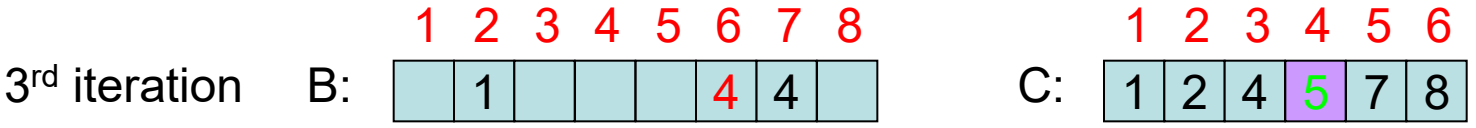

k=6
1 2 3 4 5 6 7 8
A:
3 6 4 1 3 4 1 4
1 2 3 4 5 6
2nd loop
C:
2 0 2 3 0 1
1 2 3 4 5 6
3rd loop
2 2 4 7 7 8


1 2 3 4 5 6 7 8
A:
3 6 4 1 3 4 1 4
1 2 3 4 5 6
C:
2 2 4 7 7 8

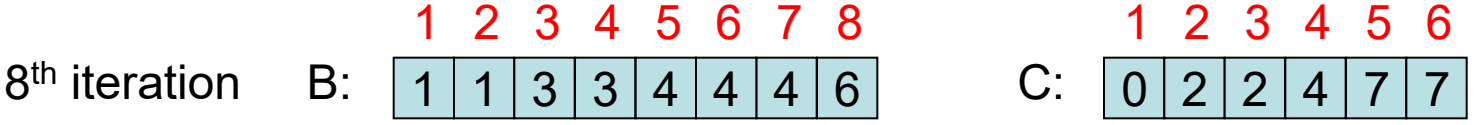
4th loop


1st iteration
B:
1 2 3 4 5 6 7 8
4
C:
1 2 3 4 5 6
2 2 4 6 7 8


2nd iteration
B:
1 2 3 4 5 6 7 8
1
4
C:
1 2 3 4 5 6
1 2 4 6 7 8


3rd iteration
B:
1 2 3 4 5 6 7 8
1
4 4
C:
1 2 3 4 5 6
1 2 4 5 7 8

……

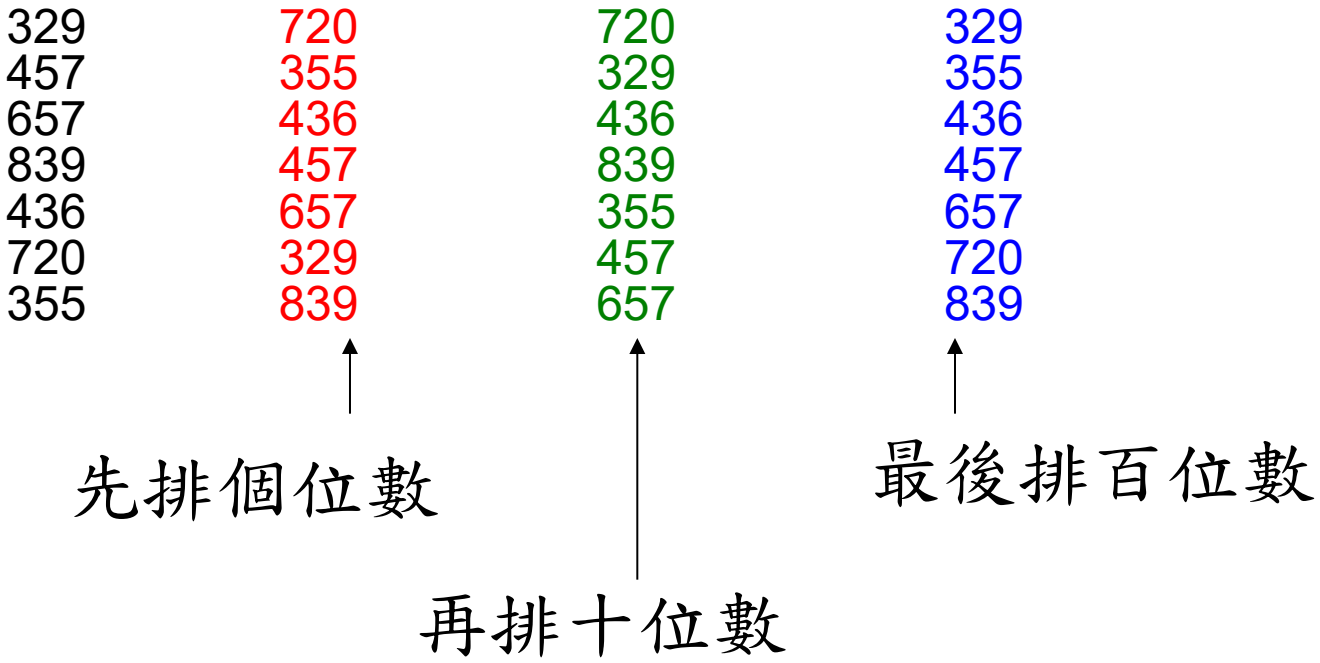

8th iteration
B:
1 2 3 4 5 6 7 8
1 1 3 3 4 4 4 6
C:
1 2 3 4 5 6
0 2 2 4 7 7

8.3 Radix Sort

- Radix Sort(基數排序法)無需利用元素間的比較排序。
- 必須依賴一些對於待排序集合中元素性質的假設。(所有待排序元素均為整數，至多d位)

Radix Sort

- 關鍵想法：利用記數排序法由低位數排到高位數。



```
Radix-Sort(A,d)
{ for i = 1 to d
      do use stable sort to sort A on digit i
}
```

- 此處使用的stable sort如果使用Counting Sort則每個Iteration只需花$\Theta(n+10)$的時間。

- 因此總共花費$O(d(n+10))$的時間。

- 如果d是常數，則Radix Sort為一個可以在Linear time完成的排序演算法。

8.4 Bucket Sort

- 當元素均勻分布在某個區間時，Bucket sort平均能在O(n)的時間完成排序。

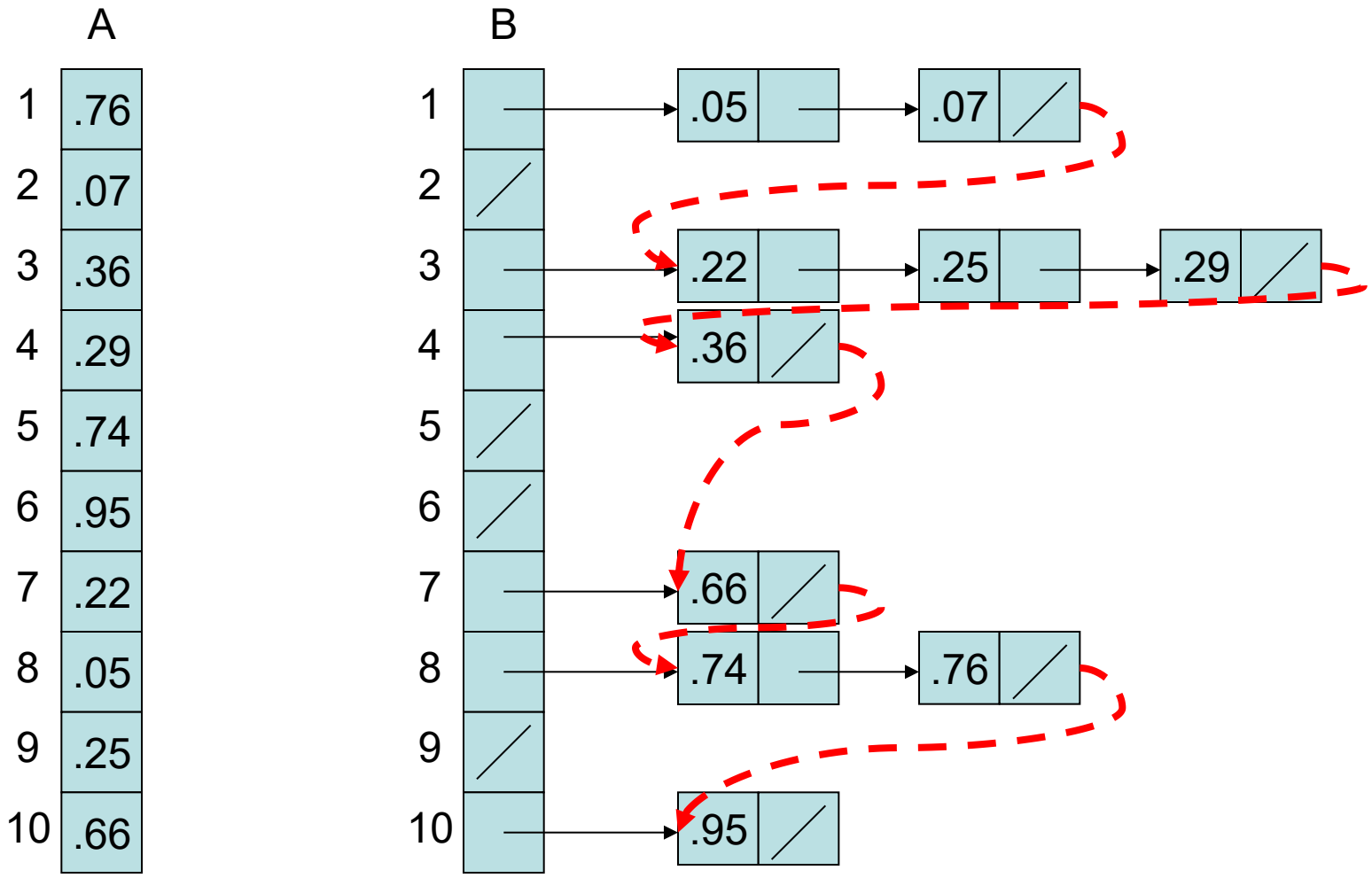
- 假定要排序n個元素A[1..n]均是介於[0,1]之間的數值。

- 準備n個籃子(bucket)，B[1..n]，將元素x依照x所在的區間放進對應的籃子：即第$\lceil xn \rceil$個籃子 。

Bucket Sort

- 元素放進籃子時，使用Linked list來儲存，並利用插入排序法排序(Insertion sort)。

- 只要依序將Lined list串接起來，即得到已排序的n個元素。



紅色虛線串起的即是最後排序好的List

時間複雜度分析

- 假定分到第i個籃子的元素個數是n_i。
- 最差情形：$T(n) = O(n) + \sum_{1 \le i \le n} O(n_i^2)$
  $= O(n^2).$

- 平均情形：$T(n) = O(n) + \sum_{1 \le i \le n} O(E[n_i^2])$
  $= O(n) + \sum_{1 \le i \le n} O(1)$
  $= O(n)$
- $E[n_i^2] = \Theta(1)$?

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

- n_i: 隨機變數 n次Bernoulli trials 落在 bucket $B[i]$的次數 ，每次成功的機率: $p = 1/n$
- $E[n_i] = np = 1$
- $Var[n_i] = np(1-p) = E\left[n_i^2\right] - E[n_i]^2$
- $E\left[n_i^2\right] = Var[n_i] + E[n_i]^2 = \left(1 - \frac{1}{n}\right) + 1$
  $= 2 - 1/n$