Chap 8. Sorting in Linear Time 孫光天

Merge sort and heapsort achieve the upper bound in worst case O(n lg n). These algorithms sorted data is based on comparison between the input elements. We will introduce three algorithms (not based on comparison) – counting sort, radix sort, and bucket sort – that run in linear time. (=> O(*n*))

8.1 Lower bounds for sorting

For n elements, we assume without loss of generality that all of the input elements are distinct. All comparisons have the form $a_i \le a_j$.

The decision-tree model

Comparison sorts can be viewed abstractly in terms of decision trees. For example, 3 data with index 1, 2 and 3，所有可能出現排序有:

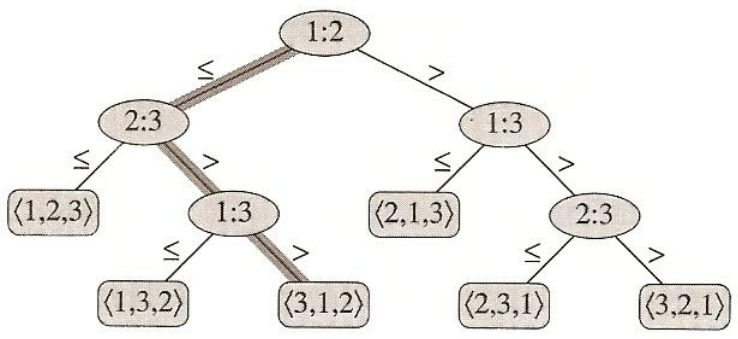

Figure 8.1 The decision tree for insertion sort operating on three elements. An internal node annotated by $i{:}j$ indicates a comparison between a_i and a_j. A leaf annotated by the permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ indicates the ordering $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$. The shaded path indicates the decisions made when sorting the input sequence $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$; the permutation $\langle 3, 1, 2 \rangle$ at the leaf indicates that the sorted ordering is $a_3 = 5 \le a_1 = 6 \le a_2 = 8$. There are $3! = 6$ possible permutations of the input elements, so the decision tree must have at least 6 leaves.

(1, 2 爲指標，非 data) Because any correct sorting algorithm must be able to produce each permutation of its input, a necessary condition for a comparison sort to be correct is that each of the *n*! permutations on *n* elements.

A lower bound for the worst case

The length of the longest path from the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm performs.

Theorem 8.1

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

Pf. Suppose a decision tree of height h with l reachable leaves.

Then,

$n! \le \quad l \quad \le 2^h$ (葉節點數一定包含所有排列數 $n!$，但 ≤ 完整樹 2^h)

Take log,

$$h \ge \lg(n!)$$

$$= \Omega\,(n \lg n)$$

Corollary 8.2

Heapsort and merge sort are asymptotically optimal comparison sorts.

Pf. The $O(n \lg n)$ upper bound for heapsort and merge sort match the $\Omega\,(n \lg n)$ worst-case lower bound.

8.2 Counting sort

Assume each of the n elements is an integer in the range 0 to k, for some integer k. When $k = O(n)$, the sort runs in $\Theta(n)$ time.

The basic idea of counting sort is to determine, for each input element x, the number of elements less than x. (計算小於 x 值的個數) This information can be used to place element x directly into its position in the output array.

Coding method: Three arrays: A[1..n], and length[A] = n; B[1..n] holds the sorted output, and C[0..k]: temporary working storage.

In lines 6-7, we determine for each i = 0, 1, .., k, how many input

elements are less than or equal to *i* by keeping a running sum of the array C.(將小於等於 ***i*** 值的個數放在 **C[*i*]**中，因此，若 **C[j]=5**，表示存放小於等於 **j** 的個數有 **5** 個，所以，**j** 是第五小，**B[5]**則可以放 **j**)

```
COUNTING-SORT(A, B, k)
1   for $i \leftarrow 0$ to $k$
2       do $C[i] \leftarrow 0$
3   for $j \leftarrow 1$ to $length[A]$
4       do $C[A[j]] \leftarrow C[A[j]] + 1$
5   ▷ $C[i]$ now contains the number of elements equal to $i$.
6   for $i \leftarrow 1$ to $k$
7       do $C[i] \leftarrow C[i] + C[i-1]$
8   ▷ $C[i]$ now contains the number of elements less than or equal to $i$.
9   for $j \leftarrow length[A]$ downto 1
10      do $B[C[A[j]]] \leftarrow A[j]$
11         $C[A[j]] \leftarrow C[A[j]] - 1$
```

倒著放入 **B array** 中
小於等於 **j** 個數減 **1**
(因爲一個以放入 **B array** 中

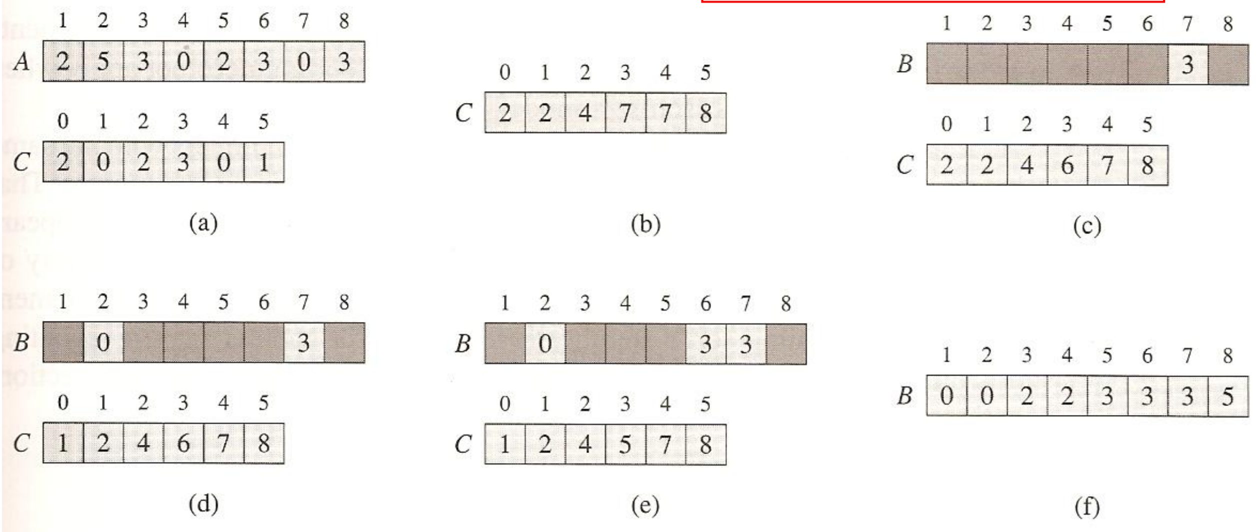

Figure 8.2 The operation of COUNTING-SORT on an input array $A[1..8]$, where each element of A is a nonnegative integer no larger than $k = 5$. **(a)** The array A and the auxiliary array C after line 4. **(b)** The array C after line 7. **(c)–(e)** The output array B and the auxiliary array C after one, two, and three iterations of the loop in lines 9–11, respectively. Only the lightly shaded elements of array B have been filled in. **(f)** The final sorted output array B.

Lines 6~7: 計算小於等於 **i** 的值有幾個。

Lines 9~10: **A** 從後面往前放入 **B**，所以，爲**“stable”**排序。**(後面說明)**

In practice, we usually use counting sort when we have k = O(n), in which case the running time is Θ(n) (= O(n + k)).

An important property of counting sort is that *stable*: numbers with

the same value appear in the output array in the same order as they do in the input array. (例如，兩個 2, input 排在後面的 2，output 也排在後面)

Counting sort's stability property is used as a subroutine in radix sort.

EXERCISE: Coding "Counting-sort (*A*, *B*, *k*)" (上面 副程式 方式)
Data: 1, 8, 4, 9, 7, 21, 33, 32, 6, 5, 25, 18
(下週報告: 1.演算法 與 Source code ; 2. 執行過程(要印出)；3.結果)

8.3 Radix sort

Radix sort is the algorithm used by the card-sorting machines you now find only in computer museums.

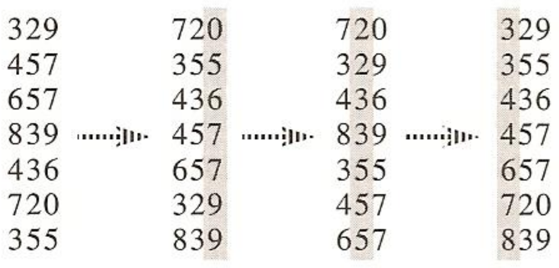

Figure 8.3 The operation of radix sort on a list of seven 3-digit numbers. The leftmost column is the input. The remaining columns show the list after successive sorts on increasingly significant digit positions. Shading indicates the digit position sorted on to produce each list from the previous one.

For decimal digits, only 10 places are used in each column. A *d*-digit number occupies a field of *d* columns.

Intuitively (直覺), one might want to sort numbers on their most significant digit. This procedure generates many intermediate piles (堆) of cards that must be kept track of. (排第二位，必須記住第一位大小)

Radix sort solves the problem of card sorting counter-intuitively by sorting on the least significant digit first. Then the entire deck is sorted again on the second-least significant digit and recombined in a like manner.

For example, we might wish to sort dates by three keys: year, month, and day.

```
RADIX-SORT(A, d)
For i =1 to d
    do use a stable sort to sort array A on digit i. (因爲 data 大小固定)
```

Lemma 8.3

Given n d-digit numbers in which each digit can take on up to k possible values RADIX-SORT correctly sort these numbers in $\Theta(d\,(n + k))$ time.

Lemma 8.4

Given n b-bit numbers and any positive integer $r \le b$ (each number is r-bit (每個數字用 r-bit)), RADIX-SORT correctly sort these numbers in $\Theta((b/r)\,(n + 2^r))$ time.

For example, we have a 32-bit word as having 4 8-bit digits, so that

$b = 32$, $r=8$, $k = 2^r - 1 = 255$, and $d = b/r = 4$.

8.4 Bucket sort

Bucket sort runs in linear expected time when the input is drawn from a uniform distribution. The bucket sort assumes that the input is generated by a random process that distributes elements uniformly over the interval [0, 1)

The basic idea of bucket sort is to divide the interval [0, 1) into n equal-sized subintervals, or buckets for an n-element array A. An auxiliary array B[0..n-1] of linked lists (buckets) and assume that there is a mechanism for maintaining such lists.

```
BUCKET-SORT(A)
1  $n \leftarrow length[A]$
2  for $i \leftarrow 1$ to $n$
3      do insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
4  for $i \leftarrow 0$ to $n-1$
5      do sort list $B[i]$ with insertion sort
6  concatenate the lists $B[0], B[1], \ldots, B[n-1]$ together in order
```

Lines 2~3: 將 A data 放入 B 中。O(n)

Line 3: B array index = A 的 data 乘以 n 倍。

If A[i] and A[j] are placed into the same bucket, then the for loop of lines 4-5 put them into the proper order (insert sort). Line 5 takes $O(n^2)$ time in the worst case. The total time takes n calls (line 4) to insertion sort.(共 $O(n^3)$) For each bucket *i*, there is n_i data. The running time of bucket sort is: (粗估)

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

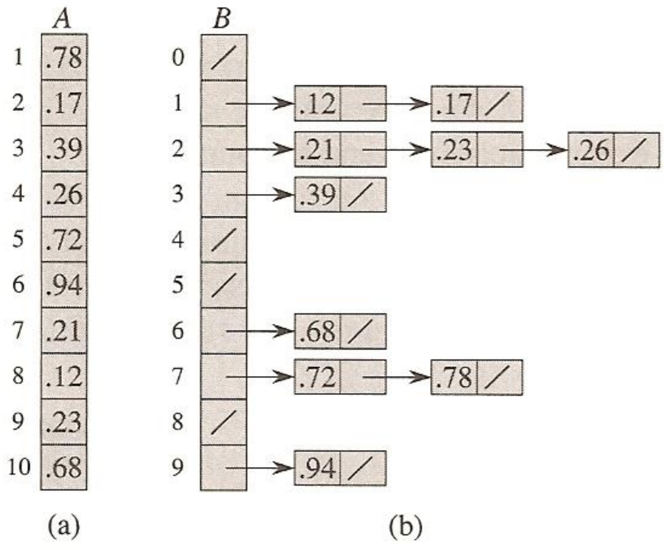

Figure 8.4 The operation of BUCKET-SORT. **(a)** The input array $A[1..10]$. **(b)** The array $B[0..9]$ of sorted lists (buckets) after line 5 of the algorithm. Bucket i holds values in the half-open interval $[i/10, (i+1)/10)$. The sorted output consists of a concatenation in order of the lists $B[0], B[1], \ldots, B[9]$.

Taking expectations of both sides and using linearity of expectation, we have

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \qquad \text{(by linearity of expectation)} \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) \qquad \text{(by equation (C.21))} \ . \end{aligned} \tag{8.1}$$

We claim that

$$\mathrm{E}[n_i^2] = 2 - 1/n \tag{8.2}$$

for $i = 0, 1, \ldots, n-1$. It is no surprise that each bucket i has the same value of $\mathrm{E}[n_i^2]$, since each value in the input array A is equally likely to fall in any bucket. To prove equation (8.2), we define indicator random variables

$$X_{ij} = \mathrm{I}\{A[j] \text{ falls in bucket } i\}$$

for $i = 0, 1, \ldots, n-1$ and $j = 1, 2, \ldots, n$. Thus,

$\mathrm{E}[n_i^2] = 2 - 1/n$ **(每個 link list n_i 個元素 sorting 花費時間期望值)**

Pf: **Let** $X_{ij} = \mathrm{I}\{A[j] \text{ falls in bucket } i\}$

for $i = 0, 1, \ldots, n-1$ and $j = 1, 2, \ldots, n$. Thus,

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

To compute $\mathrm{E}[n_i^2]$, we expand the square and regroup terms:

$$\begin{aligned}
\mathrm{E}[n_i^2] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} X_{ij}X_{ik}\right] \\
&= \sum_{j=1}^{n} \mathrm{E}[X_{ij}^2] + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} \mathrm{E}[X_{ij}X_{ik}] , \qquad (8.3)
\end{aligned}$$

where the last line follows by linearity of expectation. We evaluate the two summations separately. Indicator random variable X_{ij} is 1 with probability $1/n$ and 0 otherwise, and therefore

$$\begin{aligned}
\mathrm{E}[X_{ij}^2] &= 1\cdot\frac{1}{n} + 0\cdot\left(1-\frac{1}{n}\right) \\
&= \frac{1}{n} .
\end{aligned}$$

When $k \ne j$, the variables X_{ij} and X_{ik} are independent, and hence

$$\begin{aligned}
\mathrm{E}[X_{ij}X_{ik}] &= \mathrm{E}[X_{ij}]\,\mathrm{E}[X_{ik}] \\
&= \frac{1}{n}\cdot\frac{1}{n} \\
&= \frac{1}{n^2} .
\end{aligned}$$

Substituting these two expected values in equation (8.3), we obtain

$$\begin{aligned}
\mathrm{E}[n_i^2] &= \sum_{j=1}^{n}\frac{1}{n} + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} \frac{1}{n^2} \\
&= n\cdot\frac{1}{n} + n(n-1)\cdot\frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} \\
&= 2 - \frac{1}{n} ,
\end{aligned}$$

Then, the expect time for bucket sort is $\Theta(n) + n \cdot O(2 - 1/n) = \Theta(n)$