Algorithms
Chapter 8
Sorting in Linear Time

linear time : O(n)

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Lower bounds for sorting** 排序至少要花费的時間，即下界
- Counting sort
- Radix sort
- Bucket sort

不是 comparison sorting

Overview

- Sort n numbers in $O(n \lg n)$ time
  - Merge sort and heapsort achieve this upper bound in the worst case. 在最差的情形下，merge sort 和 heapsort 都需要 O(nlgn) 時間
  - Quicksort achieves it on average. quick sort 所需平均時間也是 O(nlgn)
  - For each of these algorithms, we can produce a sequence of n input numbers that causes the algorithm to run in $\Theta(n \lg n)$ time.
    對於每一種演算法，我們都可以產生一種輸入，使得演算法需要 θ(nlgn)
- **Comparison sorting**
  - The only operation that may be used to gain order information about a sequence is comparison of pairs of elements.
    在 sorting 過程中，只能用兩兩比較得到大小關係
  - All sorts seen so far are comparison sorts: insertion sort, selection sort, merge sort, quicksort, heapsort.
    目前看到的，都是 comparison sorting

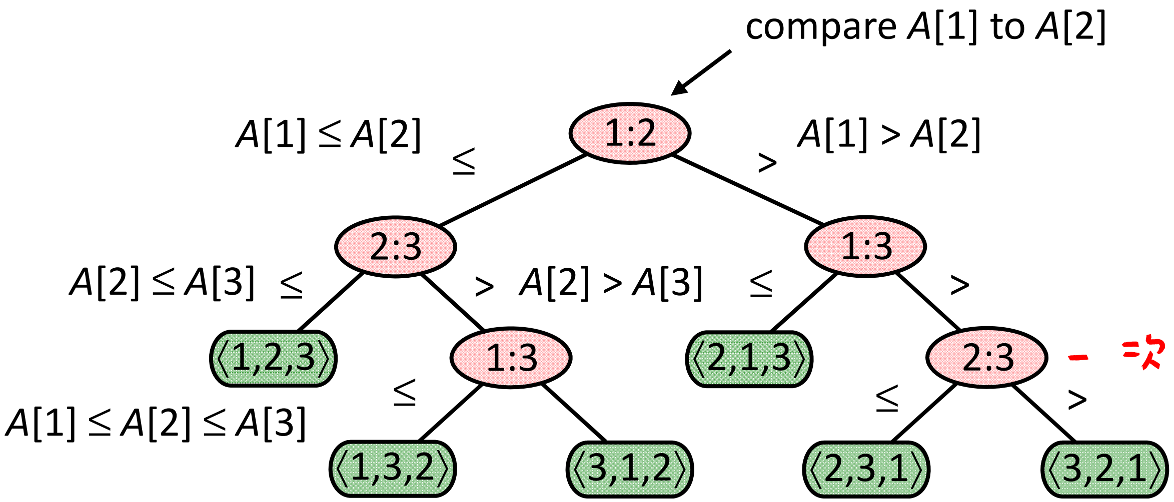
Lower bounds for sorting

- **Lower bounds** 排序最少要花多少時間
  - $\Omega(n)$ to examine all the input. 至少要$\Omega(n)$的時間去看所有input
  - All sorts seen so far are $\Omega(n \lg n)$. 目前看到的排序方法都需要$\Omega(n \lg n)$
  - We'll show that $\Omega(n \lg n)$ is a lower bound for comparison sorts.
    我們要證明"所有"comparison sorting都需要$\Omega(n \lg n)$
- **Decision tree** 對於每一種comparison sort，我們都可以抽象表示為decision tree
  - Abstraction of any comparison sort.
  - A full binary tree. 任何一個點可能是leaf或者有2個child (與原本每層皆滿的定義不同)
  - Represents comparisons made by
    - a specific sorting algorithm
    - on inputs of a given size.
    tree表示一個sorting algorithm在給定的size下的比較過程
  - Control, data movement, and all other aspects of the algorithm are ignored. 演算法過程中的"資料儲存"、"程式流程"都被忽略

Decision tree

- For insertion sort on 3 elements:

表示 insertion sort 在3個 elements 的 decision tree



一次比較

$A[1] \le A[3] \le A[2]$

- How many leaves on the decision tree?

a_1, a_2, a_3 的大小關係有 $3! = 6$ 種

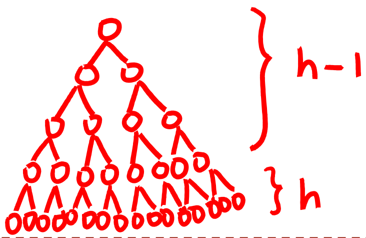
  - There are $\ge n!$ leaves, because every permutation appears at least once.

每一種結果都是一個 leaf

leaves 的個數 $\ge n!$，有 $n!$ 種可能性

Properties of decision trees$_{1/3}$

- **Lemma 1** Any binary tree of height h has $\leq 2^h$ leaves.
  - *Proof*: By induction on h. 若高度為h，leaves的個數最多有2^h個
  - **Basis:**
    - $h = 0$. Tree is just one node, which is a leaf. $2^h = 1$. h=0時只有一個node
  - **Inductive step:**
    - Assume true for height = $h - 1$. 令h-1層時正確
    - Extend tree of height $h - 1$ by making as many new leaves as possible. h-1層的每一個leaf都長出2個子節點
    - Each leaf becomes parent to two new leaves.
    - # of leaves for height h = 2 · (# of leaves for height $h - 1$)



$= 2 \cdot 2^{h-1}$ (induction hypothesis)

$= 2^h$.

Properties of decision trees$_{2/3}$

- **Theorem 1** Any decision tree that sorts n elements has height $\Omega(n \lg n)$.

任何 decision tree 高度至少 $\Omega(n \lg n)$

⇒ 每一個 node 為一次比較

至少要比較 $\Omega(n \lg n)$ 次

Proof:

- $\ell \geq n!$, where ℓ = # of leaves.
- By lemma 1, $n! \leq \ell \leq 2^h$ or $2^h \geq n!$.
- Take logs: $h \geq \lg(n!)$.

若高度為 h，leaves 的個數最多有 2^h 個

- Use Stirling's approximation: $n! > (n/e)^n$

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

(Ω : $\geq$)

Properties of decision trees$_{3/3}$

- **Corollary 1** Heapsort and merge sort are asymptotically optimal comparison sorts.

heapsort 和 merge sort 的時間複雜度與最佳的 comparison sorting 的時間只差常數倍

Proof:

- The $O(n \lg n)$ upper bounds on the running times for heapsort and merge sort match the $\Omega(n \lg n)$ worst-case lower bound from Theorem 1.

merge sort 和 heapsort 最多花 $O(n \lg n)$ 時間

任何 comparison sorting 至少花 $\Omega(n \lg n)$ 時間

Outline

- Lower bounds for sorting
- **Counting sort**
- Radix sort
- Bucket sort

不是 comparison sorting

Counting sort

- Non-comparison sorts. 不是用比較的方式得到大小的關係
- Depends on a **key assumption:** numbers to be sorted are integers in $\{0, 1, . . . , k\}$. 主要假設：數字大小 0~k 之間的整數
- **Input:** $A[1 . . n]$, where $A[j] \in \{0, 1, . . . , k\}$ for $j = 1, 2, . . . , n$.
  Array A and values n and k are given as parameters.
  n和k是給定的參數，用n和k描述演算法複雜度
- **Output:** $B[1 . . n]$, sorted. B is assumed to be already allocated and is given as a parameter. 假設B的空間已經有了，是給定的
- **Auxiliary storage:** $C[0 . . k]$. 額外所需的空間
- **Worst-case running time:** $\Theta(n+k)$. 最差情況下所需的時間

The COUNTING-SORT procedure

```
COUNTING-SORT(A, B, k)
    for i ← 0 to k
        do C[i] ← 0
    for j ← 1 to length[A]
        do C[A[j]] ← C[A[j]] + 1
    /* C[i] now contains the number of elements equal to i. */
    for i ← 1 to k
        do C[i] ← C[i] + C[i – 1]
    /* C[i] now contains the number of elements less than or equal to i. */
    for j ← length[A] downto 1
        do B[C[A[j]]] ← A[j]
           C[A[j]] ← C[A[j]] – 1
```

Lines 1–2: $\Theta(k)$ 將計數器清為0

Lines 3–4: $\Theta(n)$ 計算有幾個

Lines 6–7: $\Theta(k)$ ≤i 的有幾個

Lines 9–11: $\Theta(n)$ 放到正確的位置，由後往前放

- **The running time:** $\Theta(n+k)$.

A

1	2	3	4	5	6	7	8
2	5	3	0	2	3	0	3

C

0	1	2	3	4	5
2	0	2	3	0	1

計算有幾個

C

0	1	2	3	4	5
2	2	4	7	7	8

≤i的有幾個

放到正確的位置，由後往前放

B

1	2	3	4	5	6	7	8
						3	

C

0	1	2	3	4	5
2	2	4	6	7	8

B

1	2	3	4	5	6	7	8
	0					3	

C

0	1	2	3	4	5
1	2	4	6	7	8

B

1	2	3	4	5	6	7	8
	0				3	3	

C

0	1	2	3	4	5
1	2	4	**5**	7	8

B

1	2	3	4	5	6	7	8
0	0	2	2	3	3	3	5

Properties of counting sort

- A sorting algorithm is said to be **stable** if keys with same value appear in same order in output as they did in input.
- **Counting sort is stable** because of how the last loop works.
  因為由後往前放的原因
- Counting sort will be used in radix sort.
  radix sort 要用到 counting sort
- counting sort does not sort "inplace". counting sort 會用到額外的記憶体 <即 Barray, carray>

3' 2 3'' $\xrightarrow{\text{排序}}$ 2 3' 3'' 為 stable

3' 2 3'' $\longrightarrow$ 2 3'' 3' 非 stable (nonstable)

大小相同的 value，排序前在前方的，排序後仍在前方，稱為 stable

Outline

- Lower bounds for sorting
- Counting sort
- **Radix sort**
- Bucket sort

Radix sort 使用在多個欄位的排序，如：年，月，日

- **Key idea:** Sort **least** significant digits first. 從最不重要的數字排起

RADIX-SORT(A, d)

使用stable sort 去排每個數字

1. **for** $i \leftarrow 1$ **to** d
2. **do** use a stable sort to sort array A on digit i

- An example:

sorted

百十個		個位		十位		百位
326	⇨	690	⇨	704	⇨	326
453		751		608		435
608		453		326		453
835		704		835		608
751		835		435		690
435		435		751		704
704		326		453		751
690		608		690		835

先從個位開始排 → 十位排序 → 百位排序

百位　　十.個位
Ex:　i　　$i-1$，要排百位時，十位，個位已排序

Correctness of radix sort

- **Proof:** By induction on number of passes (*i* in pseudocode).
- **Basis:** 排序次數 Ex: p15 的 passes 是 3（個，十，百位）
  - *i* = 1. There is only one digit, so sorting on that digit sorts the array.

    只有一個數字，將此數字排序，相當於將 array 排序
- **Inductive step:**
  - Assume digits 1, 2,…, *i* – 1 are sorted.
  - Show that a stable sort on digit *i* leaves digits 1, 2,…, *i* sorted:
    - If 2 digits in position *i* are different, ordering by position *i* is correct, and positions 1,…, *i* – 1 are irrelevant.
    - If 2 digits in position *i* are equal, numbers are already in the right order (by inductive hypothesis). The stable sort on digit *i* leaves them in the right order.

（前 $i-1$ 已排序）　　（在前依然在前）

兩個數字的大小 { 相同：因為 induction hypothesis + stable sort ⇒ 所以是对的
　　　　　　　　不相同：百位比十位和個位重要，小的在前是对的，大的在後

Time complexity of radix sort

- Assume that we use counting sort as the intermediate sort.
- When each digit is in the range 0 to $k-1$, each pass over n d-digit number takes time $\Theta(n + k)$.

  數字大小是0~k-1，有n個數，排一個數字所需時間為$\theta(n+k)$
- There are d passes, so the total time for radix sort is $\Theta(d(n + k))$.

  共d欄
- If $k = O(n)$, time = $\Theta(dn)$.

- **Lemma 2:** Given n d-digit numbers in which each digit can take on up to k possible values, RADIXSORT correctly sorts these numbers in $\Theta(d(n + k))$ time.

Break each key into digits

b個 bit，每r個分一組

- **Lemma 3:** Given n b-bit numbers and any positive integer $r \leq b$, RADIX-SORT correctly sorts these numbers in $\Theta((b/r)(n + 2^r))$ time.
- **Proof**
  - We view each key as having $\underbrace{d}_{d組} = \lceil b/r \rceil$ digits of r bits each.
  - Each digit is an integer in the range 0 to $2^r - 1$, so that we can use counting sort with $k = 2^r$. 數字大小為 $0 \sim 2^r-1$ $k = 2^r$
  - Each pass of counting sort takes time $\Theta(n+k) = \Theta(n+2^r)$.每個數字
  - A total running time of $\Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$.
- **For example:**

  $\underbrace{32 \cdots 25}_{8\ bit} \underbrace{\cdots\cdots}_{8\ bit} \underbrace{\cdots\cdots}_{8\ bit} \underbrace{8 \cdots\cdots 1}_{8\ bit}$
  - 32-bit words, 8-bit digits.
  - $b = 32$, $r = 8$, $d = 32/8 = 4$, $k = 2^8 - 1 = 255$.

每8個 bit sort 一次 共排4次 大小為 $0 \sim 255$

The main reason

- How does radix sort violate the ground rules for a comparison sort? 為何radix sort可以打破comparison sort的限制

(I) - Using counting sort allows us to gain information about keys by means other than directly comparing 2 keys. 不是用兩兩比較大小關係

(II) - Used keys as array indices.

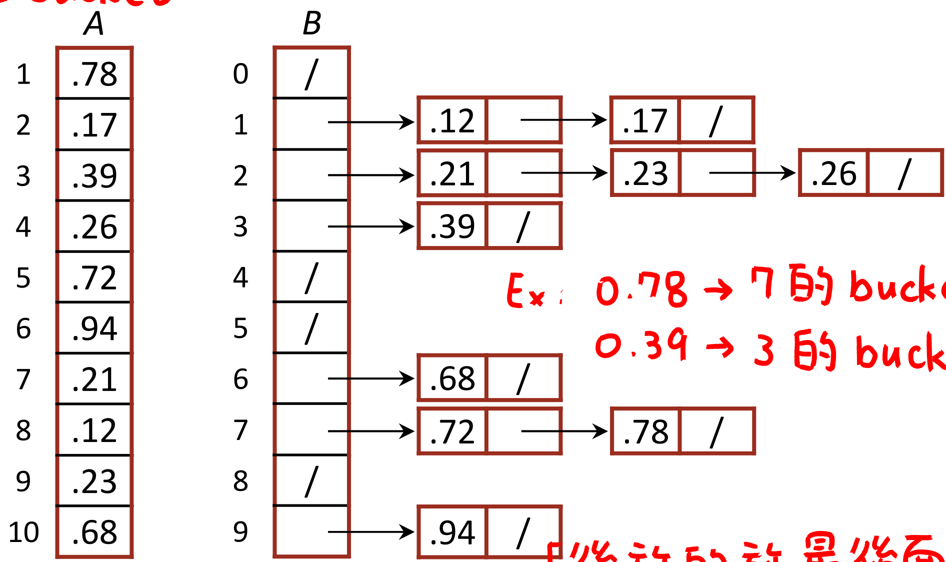
使用鍵值來當array的index (中間使用counting sort)

Outline

- Lower bounds for sorting
- Counting sort
- Radix sort
- **Bucket sort**

Bucket sort

- Assumes the input is generated by a random process that distributes elements uniformly over [0, 1).
  假設所有key值都在[0,1)之間隨机分布(包含0但不包含1)
- **Key idea:**
  - Divide [0, 1) into *n* equal-sized **buckets**. 建立n個buckets
  - Distribute the *n* input values into the buckets. 將value分配到相对应的bucket
  - Sort each bucket. sort每一個bucket
  - Then go through buckets in order, listing elements in each one.
    將list由小到大連起來



Ex: 0.78 → 7的bucket
0.39 → 3的bucket

[後放的放最後面]
維持Stable的性质

頭 先放 ⟷ 後放 chain

The BUCKET SORT procedure

- **Input:** $A[1..n]$, where $0 \le A[i] < 1$ for all i.
- **Auxiliary array:** $B[0..n-1]$ of linked lists, each list initially empty.

不是 in place，需要額外空間　是 Stable

BUCKET-SORT(A, n)

1. **for** $i \leftarrow 1$ **to** n
2. **do** insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$

分配到相对应的 bucket

3. **for** $i \leftarrow 0$ **to** $n-1$
4. **do** sort list $B[i]$ with insertion sort

(insertion sort，是 Stable)
sort 每一個 bucket

5. concatenate lists $B[0]$, $B[1]$, . . . , $B[n-1]$ together in order
6. **return** the concatenated lists

連起來

Correctness of bucket sort

- Consider $A[i]$, $A[j]$. 有2個 value
  Assume without loss of generality that $A[i] \le A[j]$.
  第i個value的bucket 第j個value的bucket
- Then $\lfloor n \cdot A[i] \rfloor \le \lfloor n \cdot A[j] \rfloor$.
- So $A[i]$ is placed into the same bucket as $A[j]$ or into a bucket with a lower index.
- If same bucket, insertion sort fixes up.
- If earlier bucket, concatenation of lists fixes up.

i的bucket較j小或相同 (I) 相同：用 insertion sort 排好
(II) 不相同：連的時候是由小到大連，也是对的

Time complexity of bucket sort

- Relies on no bucket getting too many values.
  重点是同一個 bucket 不能有太多的 element
- All lines of algorithm except insertion sorting take $\Theta(n)$ altogether. 除了 sort 外，其他的時間都是 O(n)
- Intuitively, if each bucket gets a constant number of elements, it takes $O(1)$ time to sort each bucket ➔ $O(n)$ sort time for all buckets.
  如果每個只有常數個 bucket ⇒ sort 每一個 bucket 為 O(1) ⇒ sort 全部為 O(n)
- We "expect" each bucket to have few elements, since the average is 1 element per bucket.
  希望同一個 bucket 不要有太多的 element，因為平均是一個

Time complexity of bucket sort

- Define a random variable: ***n**$_i$* = the number of elements placed in bucket $B[i]$. 排每個bucket的時間 insertion time $O(n_i^2)$

  第i個bucket n_i個

- Because insertion sort runs in quadratic time, bucket sort time is $T(n) = \theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$.

  建n個bucket的時間+丟到相對應的bucket+連起來的時間

- Take expectations of both sides:

取期望值 兩相加取期望=個別取期望再相加

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

$$= \theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)]$$ — linearity of expectation

$$= \theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$ — E[*a*X] = *a*E[X]

乘常數取期望值
=取期望後再乘常數

Claim that $E[n_i^2] = 2 - 1/n$ for $0 \le i \le n-1$.
宣稱

Therefore, $E[T(n)] = \theta(n) + \sum_{i=0}^{n-1} O(2 - 1/n)$

$$= \theta(n) + O(n)$$

$$= \theta(n).$$ $\sum_{i=0}^{n-1} (2) = O(n)$

Proof of claim

- **Claim:** $E[n_i^2] = 2 - 1/n$ for $0 \le i \le n-1$.
- **Proof** element 落在每一個 bucket 的机率為 $\frac{1}{n}$
  - $\Pr\{A[j] \text{ falls in bucket } i\} = p = 1/n$.
  - The probability that $n_i = k$ follows the binomial distribution $b(k; n, p)$. 第i個 bucket 有k個的机率是 binomial distribution
  - So, $E[n_i] = \underset{n \cdot \frac{1}{n}}{np} = 1$ and variance 变異数 $\mathrm{Var}[n_i] = np(1-p) = 1 - 1/n$.
  - For any random variable X, we have $E[n_i^2] = \mathrm{Var}[n_i] + E^2[n_i]$

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

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

b(k, n, p)

有k個的机率 丟n次 每一個bucket的机率

Notes 非兩兩相比較所得的結果

- Again, not a comparison sort. Used a function of key values to index into an array. 用鍵值放到相对应的 array
- This is a **probabilistic analysis**. We used probability to analyze an algorithm whose running time depends on the distribution of inputs. 這是一個机率分析 Input 的 distribution 不同，時間複雜度也不同
- Different from a **randomized algorithm**, where we use randomization to impose a distribution. (擲骰子為一隨机過程，不是 randomization，因為沒有擲骰子來決定下一步， bucket sort 每一步皆確定
- With bucket sort, if the input isn't drawn from a uniform distribution on [0, 1), all bets are off (performance-wise, but the algorithm is still correct).
  如果 element 分佈不均，除了正確性外都是錯的