Chap 2. Getting Started

孫光天

2.1 Insertion sort

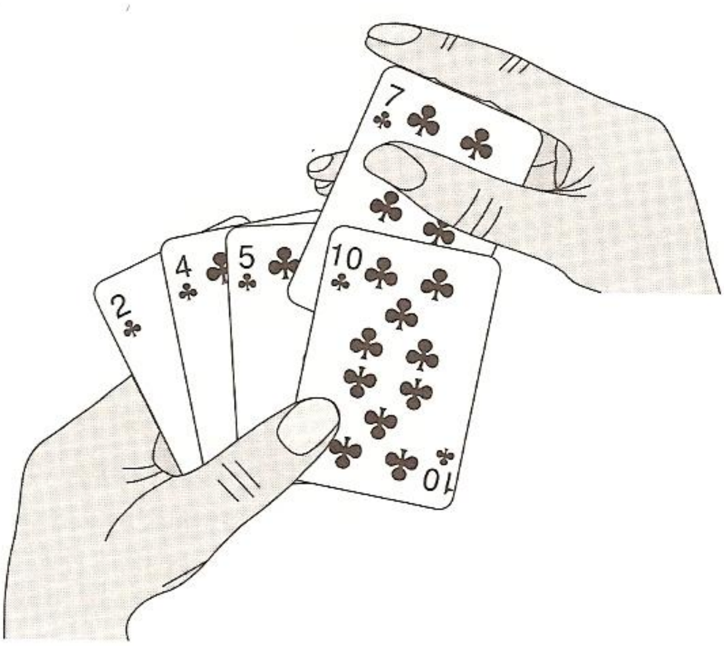
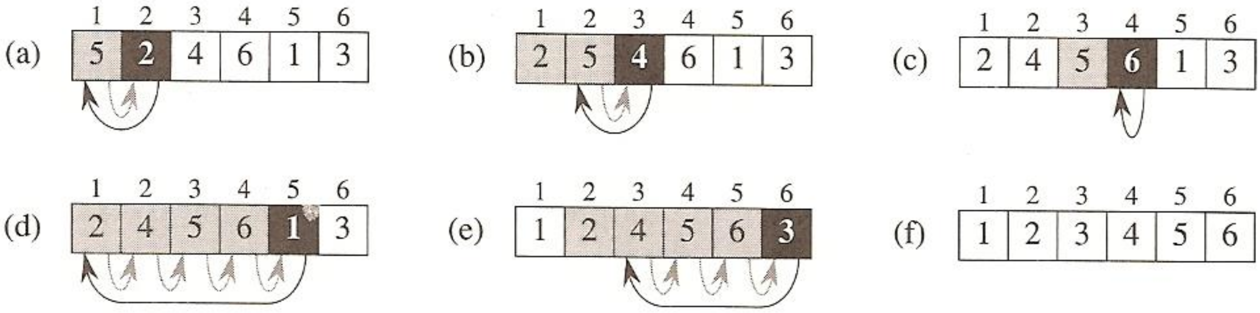


Figure 2.1 Sorting a hand of cards using insertion sort.

State description:

2.1 Insertion sort



Algorithm:

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

Data 比較大，往後移

Loop invariants and correctness of insertion sort:

Loop invariants: 資料不會因爲 **loop** 的執行而改變；可以利用 **loop invariant** 來證明 **algorithm** 的正確性，必須符合下列三個條件：

1. **Initialization: It is true prior to the first iteration of the loop.**

   (初始時，資料符合演算法特性（只有一筆 **A[1]**）)

2. **Maintenance: It is true before an iteration of the loop, it remains true before the next iteration.**

   (維持性，**loop** 執行前 (**A[1] ~ A[k]**) 與執行後 (**(A[1] ~ A[k+1])**)，資料符合演算法特性（**insertion sort:** 由小排到大）)

3. **Termination: Loop terminates** 也符合特性.

   (最終 **loop** 會結束且符合演算法特性)

The loop invariant gives us a useful property that helps us show that an algorithm is correct. (證明方法之"正確性")

(It is similar to mathematical induction.(數學歸納法))

Pseudocode conventions: **< see p.19 of the text book. >**

For example, symbol "▷" indicate that the remainder of the line is a comment.

Parameters are passed to a procedure *by <u>value</u>*. (其他：***name, address, reference***)

2.2 Analyzing algorithms

Random-access machine (RAM) model is used.

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*[A]	c_1	n
do *key* $\leftarrow A[j]$	c_2	$n-1$
$\triangleright$ Insert $A[j]$ into the sorted sequence $A[1\,.\,.\,j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

The running time of the algorithm is the sum of running times for each sta executed; a statement that takes c_i steps to execute and is executed n tim contribute $c_i n$ to the total running time.[5] To compute $T(n)$, the running t INSERTION-SORT, we sum the products of the *cost* and *times* columns, obta

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

where *n* is the number of input (input length, not the input value).

Case 1. The best case, if the array is already sorted. $t_j = 1$.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

This is a linear function.

Case 2. The worst case, if the array is in reverse sorted order.

$t_j = j$, and

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

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \; . \end{aligned}$$

This is a quadratic function.

We only concentrate on "worst case". Three reasons:

1. **Worst case is the upper bound on the running time that guarantees this algorithm will never take any longer.**
2. **The worst case occurs fairly often. When we search a data in a database, we often search all database for an absent data.**
3. **The "average case" is often roughly as bad as the worst case.**
   (In some particular cases, we shall be interested in the average-case or expected running time of an algorithm by using the probabilistic analysis.)

Order of growth

Rate of growth or order of growth: We only consider the leading term of a formula (e.g., an^2 in $an^2 + bn + c$) The notation "Θ" is used in this book. (e.g., $\Theta(n^2)$)

2.3 Divide-and-conquer approach

Divide the problem into a number of subproblem.

Conquer the subproblems by solving them recursively.

Combine the solutions to the subproblems into the solution of the original problem.

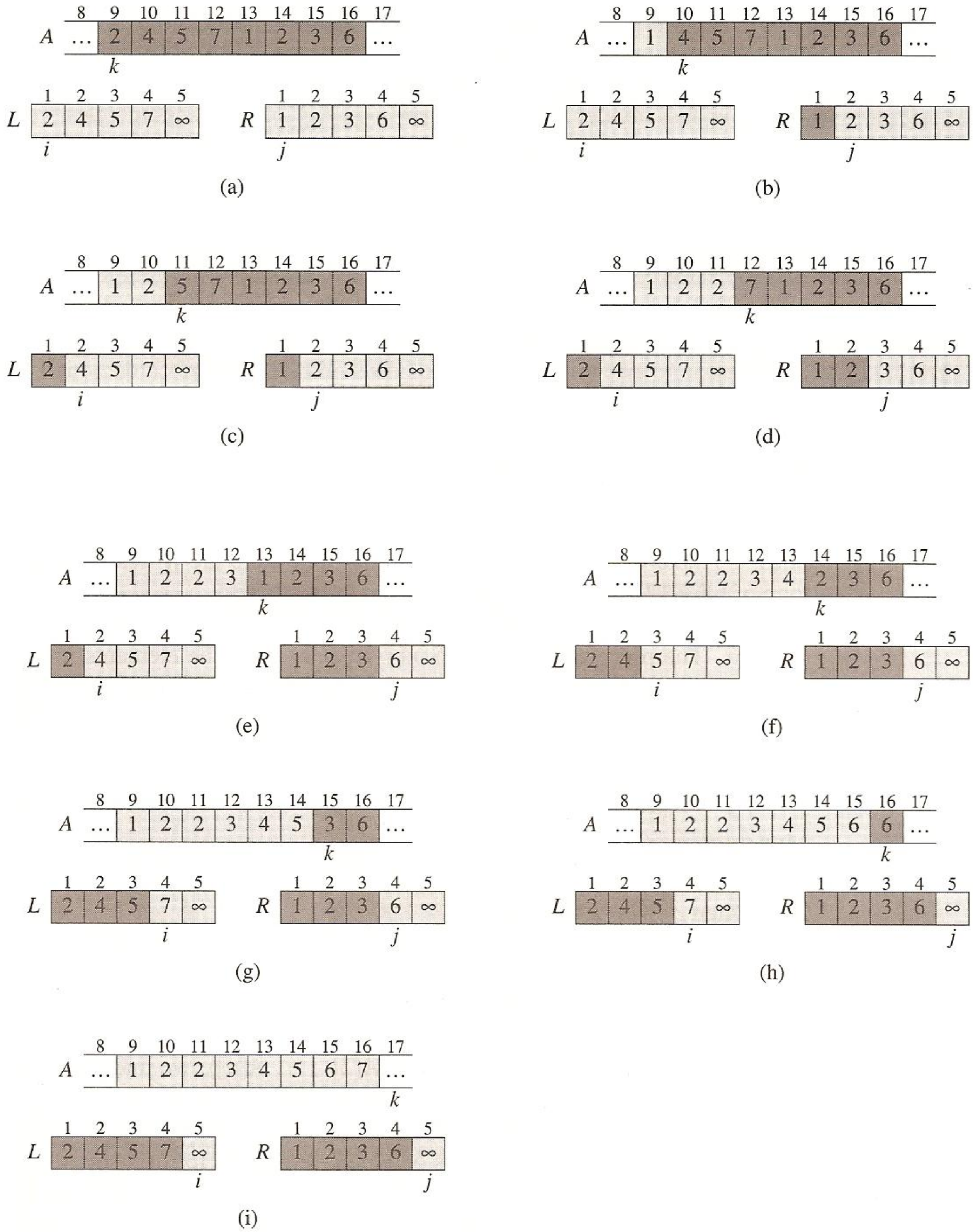
A: array; p, q, r : indices & p ≤ q < r. *A*[p..q] & *A*[q+1 .. r] put into two sub-array *L* & *R* (sorted). Then merge into an array *A*.

(先看圖解說明 （下頁），再看步驟)

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
for i ← 1 to n1
      do L[i] ← A[p + i − 1]
for j ← 1 to n2
      do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
      do if L[i] ≤ R[j]
            then A[k] ← L[i]
                 i ← i + 1
            else A[k] ← R[j]
                 j ← j + 1
```

結尾放入一個很大值

Steps:



Merge-sort (*A*, p, r)

1. if p < r

2. then q <- ⌊(p+r) /2⌋

3. Merge-sort (*A*, p, q)

4. Merge-sort (*A*, q+1, r)

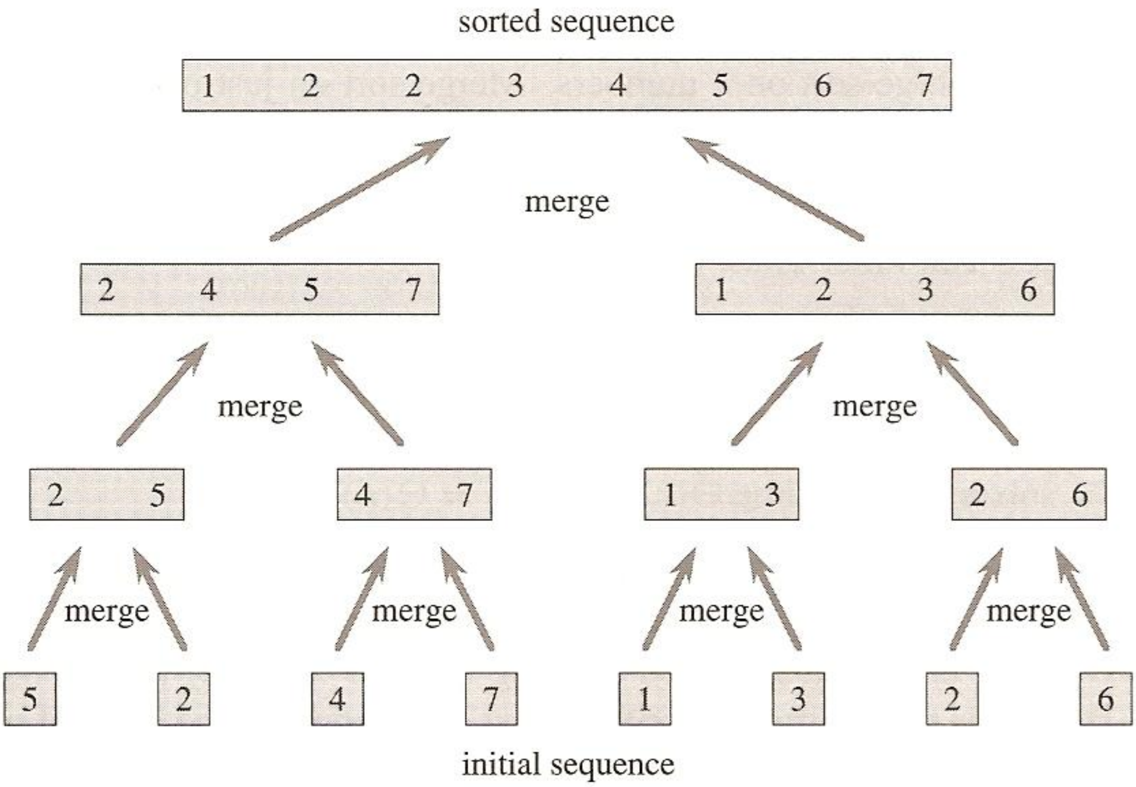
5. Merge (*A*, p, q, r)

/* step 3 & 4 are recurrence processes */

(The following figure is the merge-sort process. The input sequence is divided and merged from the bottom.)

Input *A* = (5, 2, 4, 7, 1, 3, 2, 6) (at the bottom)

Sorting process:



EXERCISE: Coding "Merge-sort (*A*, p, r)" (上面 recurasive 方式)

Data: 1, 8, 4, 9, 7, 21, 33, 32, 6, 5, 55, 22, 17, 26, 36, 24

(下週 報告 1. 演算法 與 Source code ; 2. 執行過程(要印出)； 3.結果)

Analysis of merge sort:

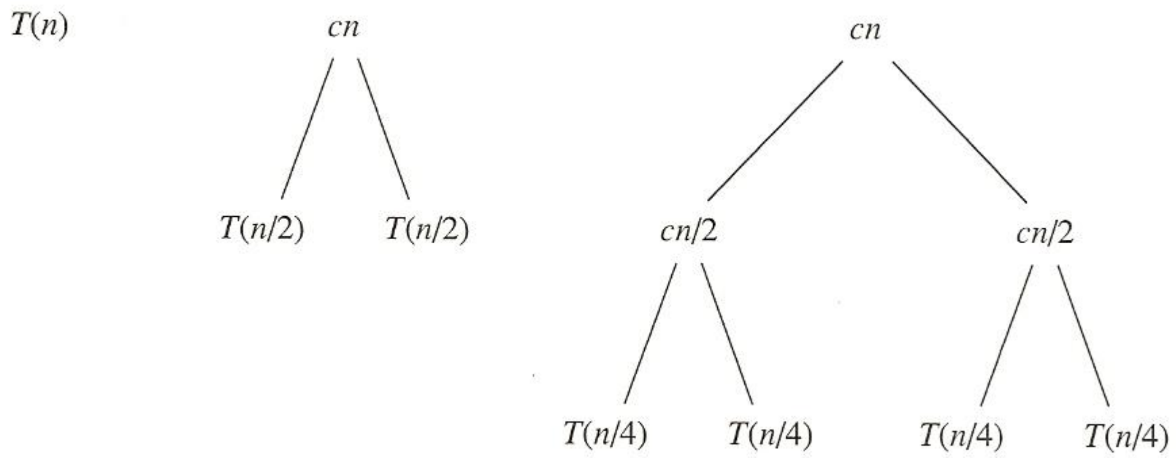
$$T(n) = T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + cn \qquad n \geq 2$$

T(1)=*a*, where *a* & *c* are constants.

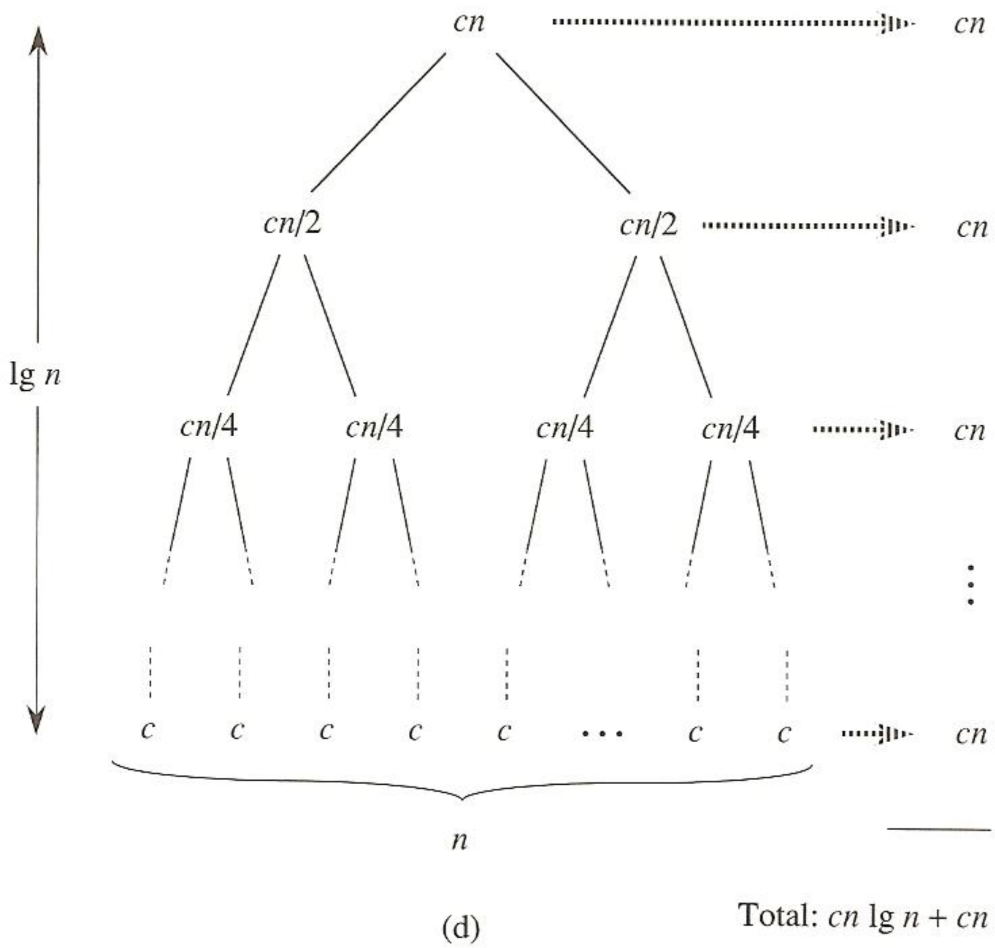
令 $n = 2^k$ **(i.e. n is a power of 2)**

$$\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + cn \\
&= 2[2T(\frac{n}{4}) + c \cdot \frac{n}{2}] + cn \\
&= 2^2 T(\frac{n}{2^2}) + cn + cn \\
&= 2^2 T(\frac{n}{2^2}) + 2cn \\
&= \ldots\ldots\ldots\ldots \\
&= 2^k T(\frac{n}{2^k}) + kcn \\
&= 2^k T(1) + kcn \\
&= an + c \cdot n \cdot \log n
\end{aligned}$$

=> $T(n) = O(n \log n)$



(a) (b) (c)



(d)

Height: lg *n*; levels: lg *n* + 1. Total computing = cn (lg *n* + 1) = c*n* lg *n* + c*n*