Chapter 2
Getting Started

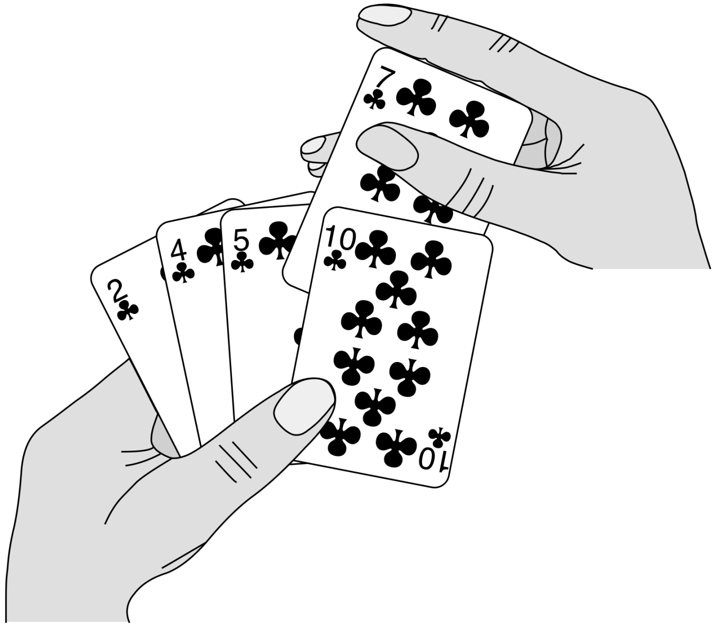
Chi-Yeh Chen
陳奇業
成功大學資訊工程學系

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

2.1 Insertion sort

- **Example:** Sorting problem
  - Input: A sequence of *n* numbers $\langle a_1, a_2, \dots, a_n \rangle$
  - Output: A permutation $\langle a_1', a_2', \dots, a_n' \rangle$ of the input sequence such that $a_1' \le a_2' \le \dots \le a_n'$



- The number that we wish to sort are known as the *keys*.

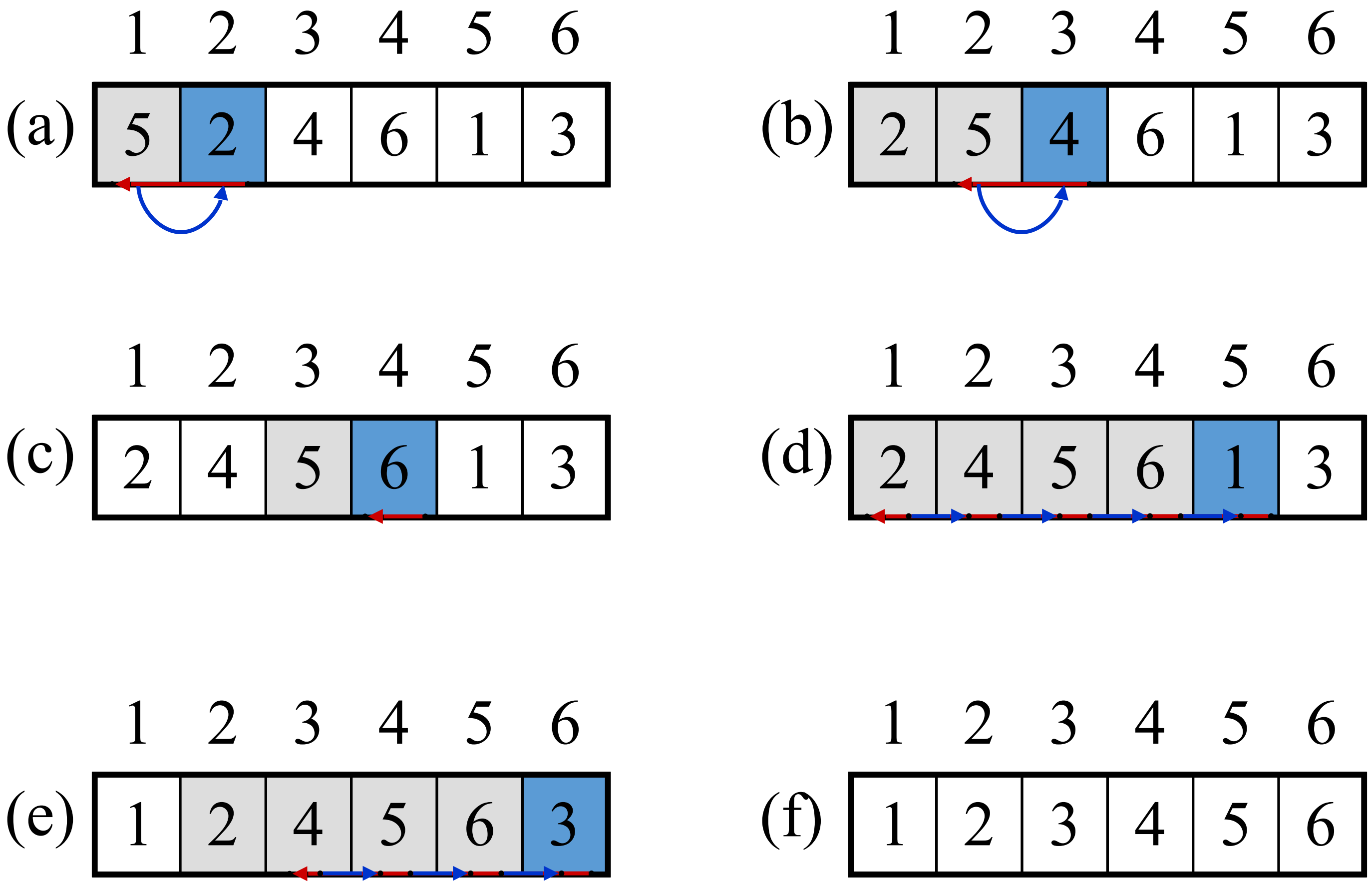
Pseudocode

- Insertion sort

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
```

The operation of Insertion-Sort



- ***Sorted in place:***
  - The numbers are rearranged within the array A, with at most a constant number of them sorted outside the array at any time.
- ***Loop invariant:***
  - At the start of each iteration of the for loop of line 1-8, the subarray $A[1,\ldots, j-1]$ consists of the elements originally in $A[1,\ldots, j-1]$ but in sorted order.

- We use loop invariants to help us understand why an algorithm is correct. We must show three things about a loop invariant:
  - **Initialization**: It is true prior to the first iteration of the loop.
  - **Maintenance**: If it is true before an iteration of the loop, it remains true before the next iteration.
  - **Termination**: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

- Let us see how these properties hold for insertion sort.
  - **Initialization**: When $j = 2$, $A[1,\ldots,\ j-1] = A[1]$ is true.
  - **Maintenance**:

1	2	3	4	5	6
2	5	4	6	1	3

The subarray $A[1,\ldots,\ j]$ then consists of the elements originally in $A[1,\ldots,\ j]$, but in sorted order. Incrementing j for the next iteration of the for loop then preserves the loop invariant.

  - **Termination**: The for loop to terminate is that $j > A.length = n$. When $j = n+1$, the subarray $A[1,\ldots,\ n]$ then consists of the elements originally in $A[1,\ldots,\ n]$, but in sorted order.

2.2 Analyzing algorithms

- Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.
  - ***Resources:*** memory, communication, bandwidth, logic gate, **time**.
  - **Assumption:** one processor, ***RAM***
  - constant-time instruction: **arithmetic** (add, subtract, multiply, divide, remainder, floor, ceiling); **data movement** (load, store, copy); **control** (conditional and unconditional bramch, subroutine call and return)
  - Date type: integer and floating point
  - Limit on the size of each word of data

2.2 Analyzing algorithms

- The best notion for **input size** depends on the problem being studied.
- The **running time** of an algorithm on a particular input is the number of primitive operations or “steps” executed. It is convenient to define the notion of step so that it is as machine-independent as possible.

Analysis of insertion sort

Insertion-sort(A)

1. **for** $j \leftarrow 2$ **to** length[A]
2. **do** key $\leftarrow A[j]$
3. *Insert $A[j]$ into the sorted sequence $A[1,\ldots, j-1]$
4. $i \leftarrow j-1$
5. **while** $i > 0$ and $A[i] >$ key
6. **do** $A[i+1] \leftarrow A[i]$
7. $i \leftarrow i-1$
8. $A[i+1] \leftarrow$ key

Line	cost	times
1	c_1	n
2	c_2	$n-1$
3	0	
4	c_4	$n-1$
5	c_5	$\sum_{j=2}^{n} t_j$
6	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	c_8	$n-1$

- t_j: the number of times the while loop test in line 5 is executed for the value of j.

Analysis of insertion sort

- The running time

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

- $t_j = 1$, for $j = 2,3,\ldots,n$: Linear function on n

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_6(n-1) + c_7(n-1) + c_8(n-1)$$
$$= \left(c_1 + c_2 + c_4 + c_5 + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

藏行顯光 成就共好 國立成功大學 九十週年 90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

Analysis of insertion sort

- $t_j = j$, for $j = 2,3,\ldots,n$: Quadratic function on n

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\frac{n(n-1)}{2} + c_8(n-1)$$
$$= \left(\frac{c_5+c_6+c_7}{2}\right)n^2 - \left(c_1+c_2+c_4+\left(\frac{c_5+c_6+c_7}{2}\right)+c_8\right)n - (c_2+c_4+c_5+c_8)$$

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

Worst-case and average-case analysis

- Usually, we concentrate on finding only on the ***worst-case running time***.
- Reason:
  - It is an upper bound on the running time.
  - The worst case occurs fair often.
  - The average case is often as bad as the worst case. For example, the insertion sort. Again, quadratic function.

Order of growth

- In some particular cases, we shall be interested in ***average-case***, or ***expect*** running time of an algorithm.

- It is the ***rate of growth***, or ***order of growth***, of the running time that really interests us.

2.3 Designing algorithms

- There are many ways to design algorithms:
  - **Incremental approach:** having sorted the subarray $A[1 \ldots j-1]$, we inserted the single element $A[j]$ into its proper place, yielding the sorted subarray $A[1 \ldots j]$.
    Ex. insertion sort
  - **Divide-and-conquer:** merge sort
    - recursive:
      - divide
      - conquer
      - combine

藏行顯光 成就
共好 國立成功大學 九十週年 90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

Pseudocode

- Merge sort

Merge(A, p, q, r)

$n_1 \leftarrow q - p + 1$

$n_2 \leftarrow r - q$

create array $L[1,\ldots, n_1 + 1]$ and $R[1,\ldots, n_2 + 1]$

for $i \leftarrow 1$ **to** n_1

    do $L[i] \leftarrow A[p + i - 1]$

for $j \leftarrow 1$ **to** n_2

    do $R[j] \leftarrow A[q + j]$

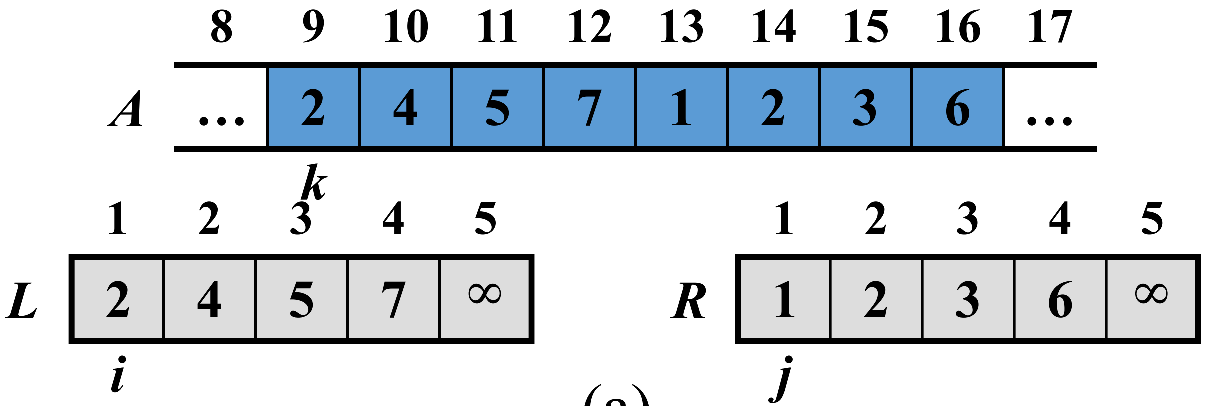
$L[n_1 + 1] \leftarrow \infty$

$R[n_2 + 1] \leftarrow \infty$

Pseudocode

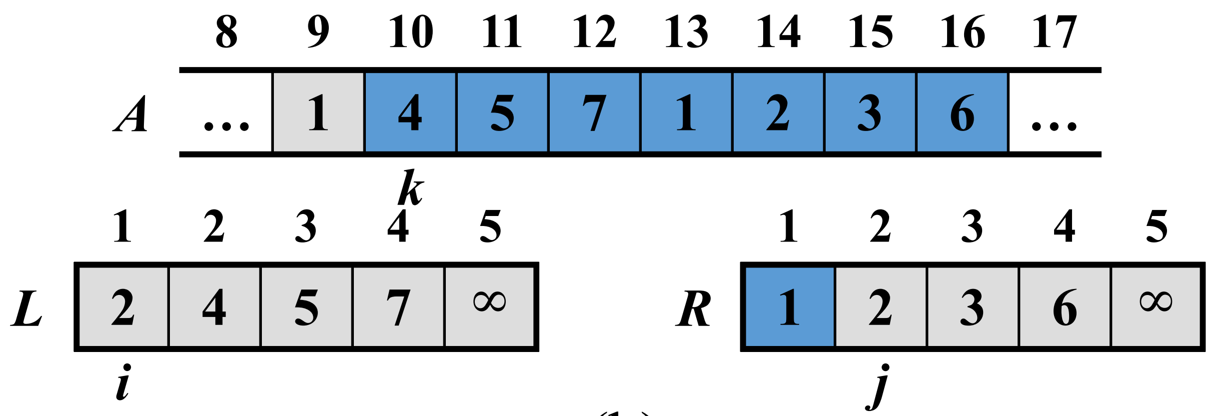
```
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
          then A[k] ← L[i]
               i ← i + 1
          else A[k] ← R[j]
               j ← j + 1
```

The operation of lines 10-17



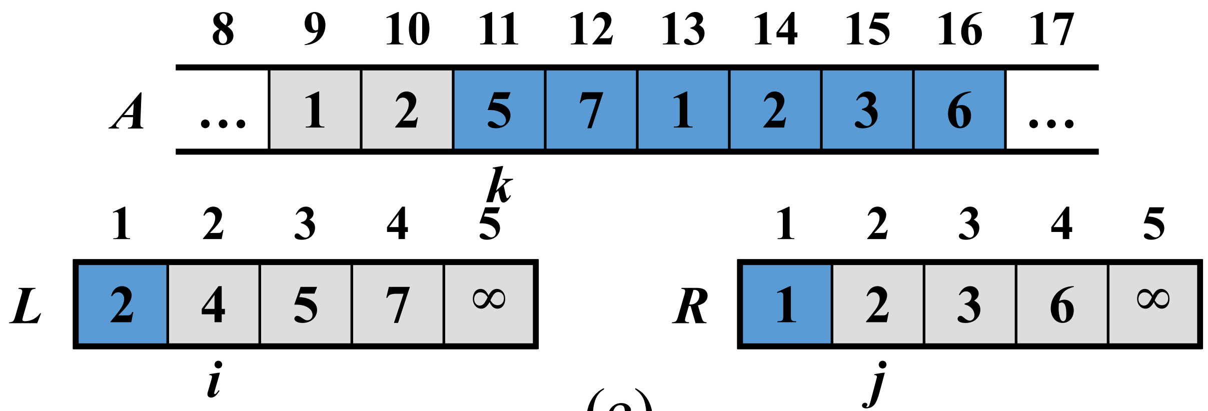
(a)

The operation of lines 10-17



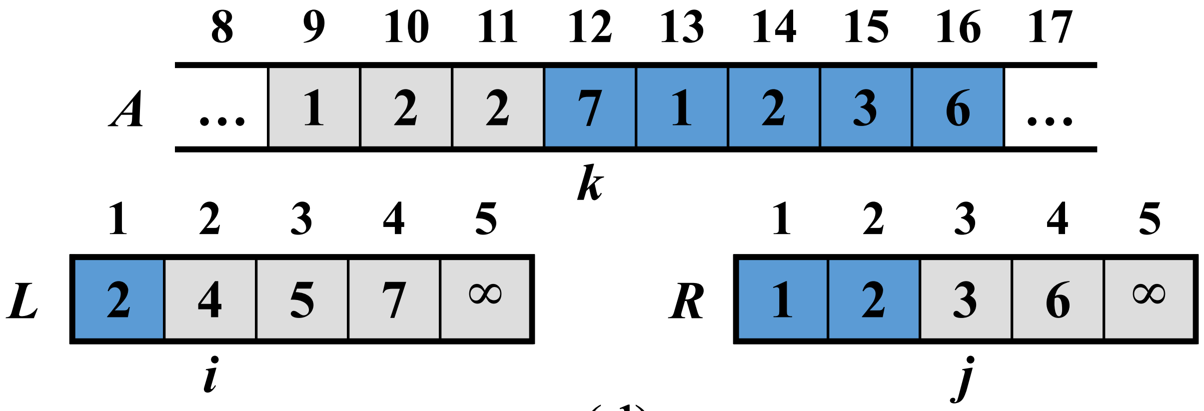
(b)

The operation of lines 10-17



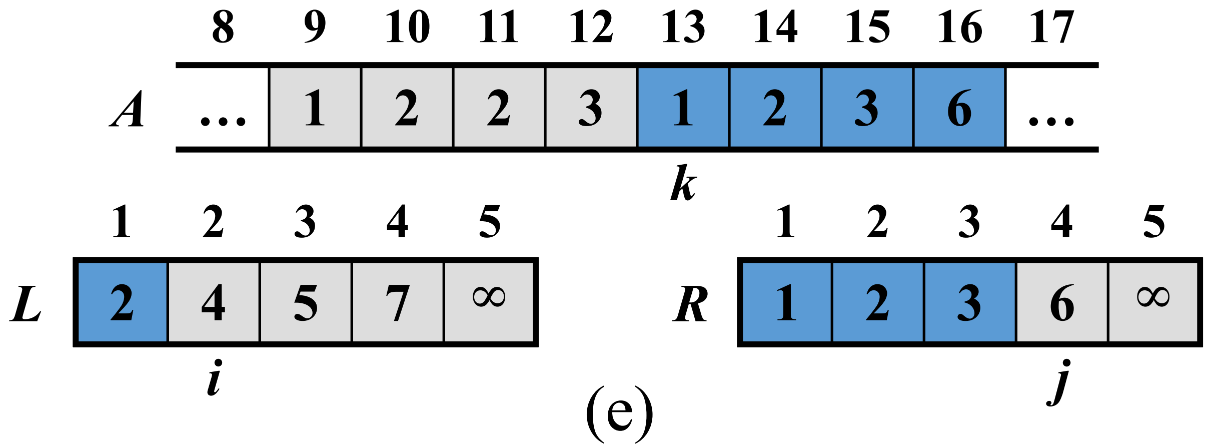
(c)

The operation of lines 10-17

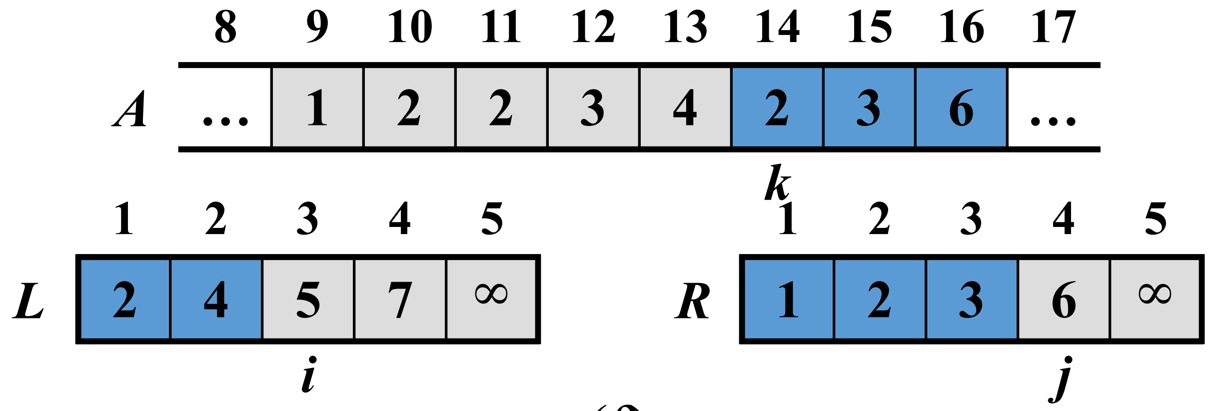


(d)

The operation of lines 10-17

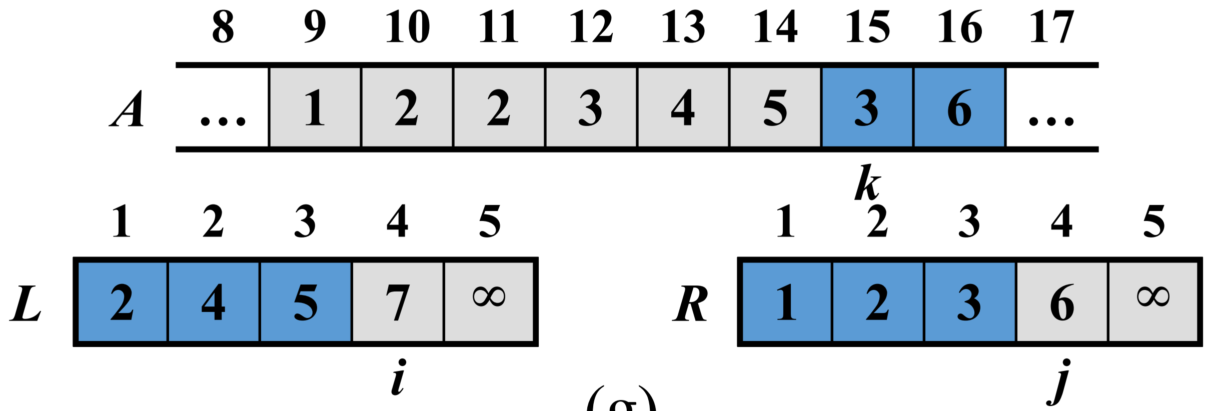


The operation of lines 10-17



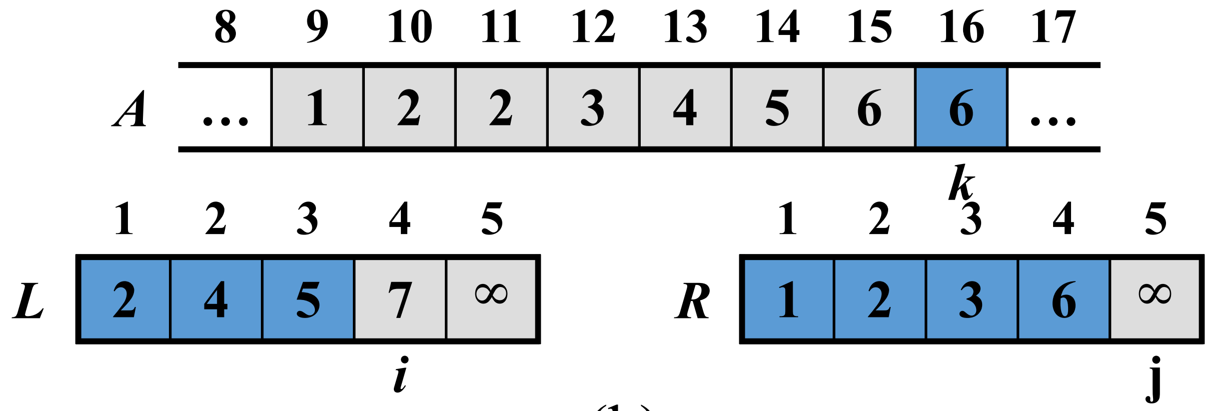
(f)

The operation of lines 10-17



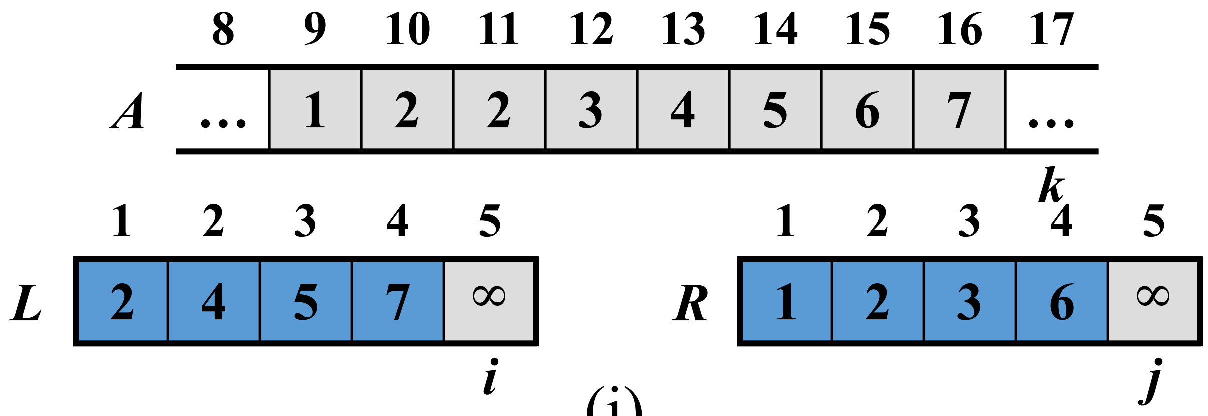
(g)

The operation of lines 10-17



(h)

The operation of lines 10-17

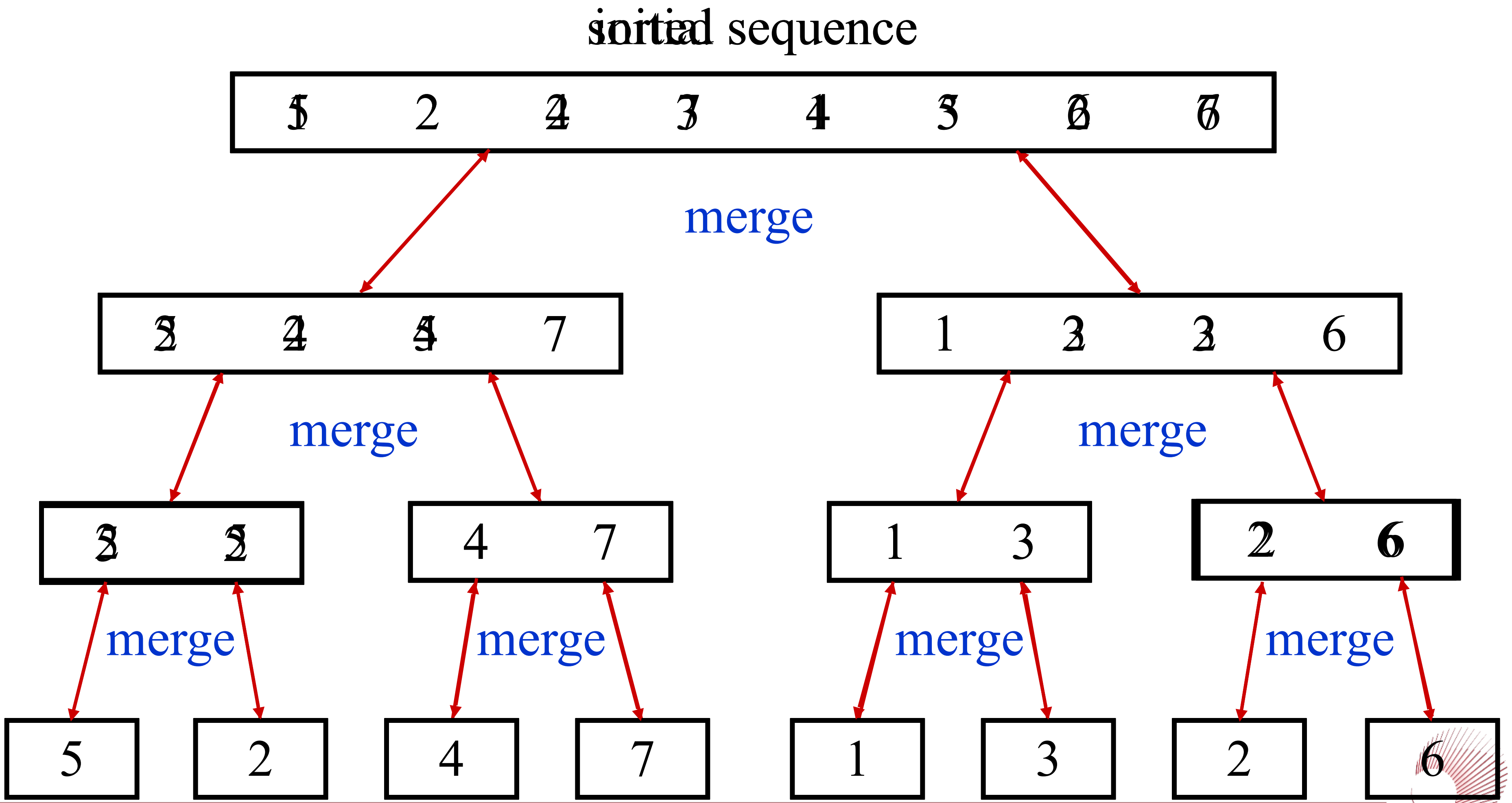


(i)

Pseudocode

```
MERGE-SORT(A, p, r)
if p < r
    then q ← ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

The operation of Merge sort



Analysis of Merge sort

- Analyzing divide-and-conquer algorithms
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$
  - See Chapter 4.
- Analysis of merge sort
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$
  - $T(n) = \Theta(n \lg n)$

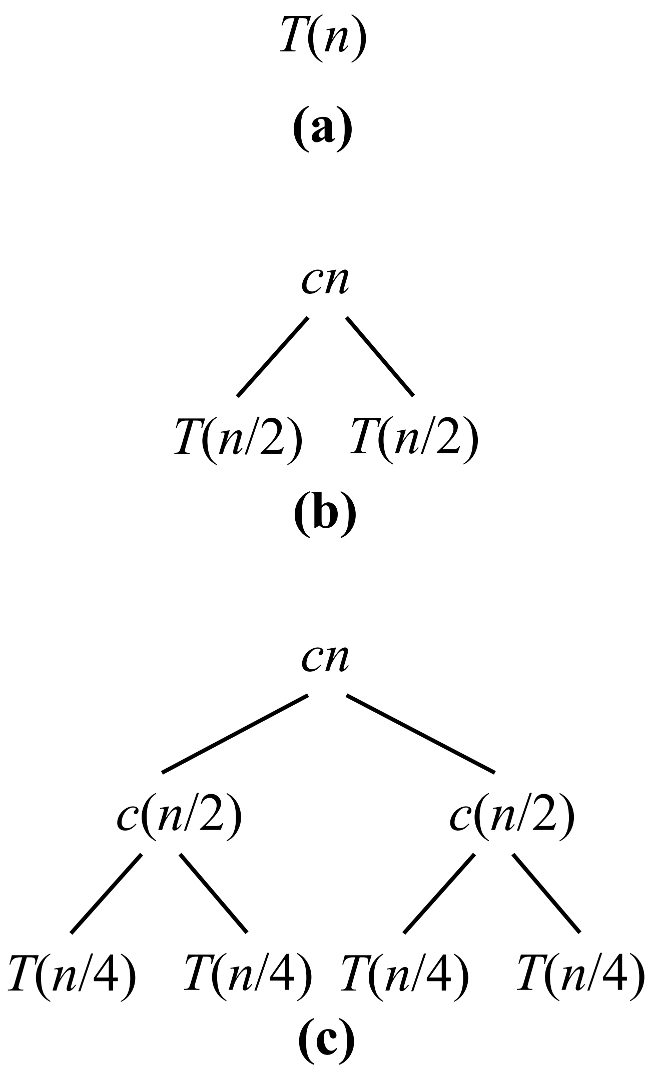
Analysis of Merge sort

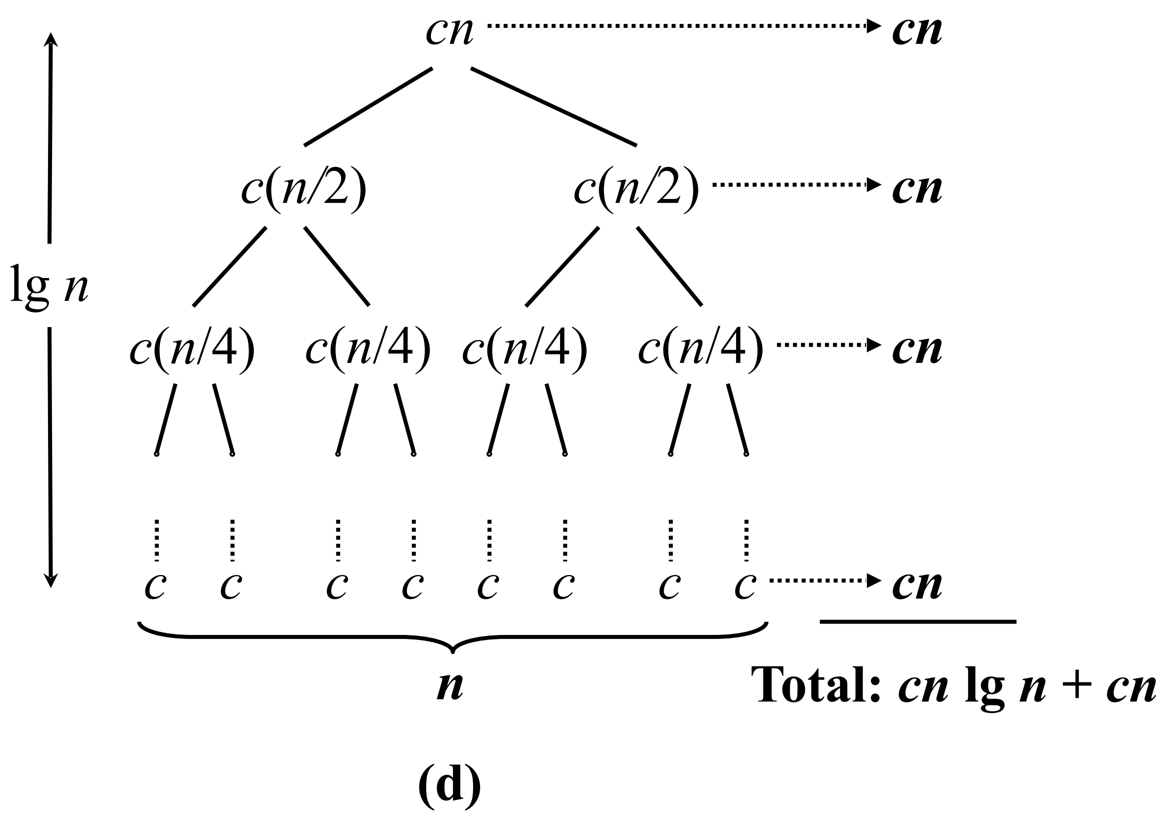
- $$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \end{cases}$$

where the constant *c* represents the time require to solve problem of size 1 as well as the time per array element of the divide and combine steps.

藏行顯光 成就 共好
國立成功大學 九十週年
90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

The construction of a recursion tree





The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated)

- Outperforms insertion sort!

藏行顯光
成就共好

Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

國立成功大學
National Cheng Kung University

1931