2. Getting started

中國文化大學
資訊工程系
副教授　張耀鴻
112學年度第2學期

2. Getting started

- Insertion sort (插入排序法)
- Pseudocode conventions (虛擬碼)
- Analyzing algorithms (演算法分析)
  - ✓ Best case 最佳情況
  - ✓ Worst-case 最差情況
  - ✓ Average-case 平均情況
  - ✓ Order of growth 級數增長
- Designing algorithms (演算法設計)
  - ✓ Incremental approach: Insertion sort 漸進式: 插入排序
  - ✓ Divide-and-conquer: Mergesort 各個擊破法: 合併排序
    - Analysis of divide-and-conquer algorithm 各個擊破法分析

2.1 Insertion sort 插入排序法

- **Example: Sorting problem 排序問題**
  - ✓ Input: A sequence of *n* numbers $< a_1, a_2, ..., a_n >$
    輸入: n 個任意順序的數字 $a_1..a_n$
  - ✓ Output: A permutation $< a'_1, a'_2, ..., a'_n >$ of the input sequence such that $a'_1 \le a'_2 \le ... \le a'_n$
    輸出: n 個由小到大順序排列的數字

The number that we wish to sort are known as the ***keys***. 將鍵值依大小順序排列

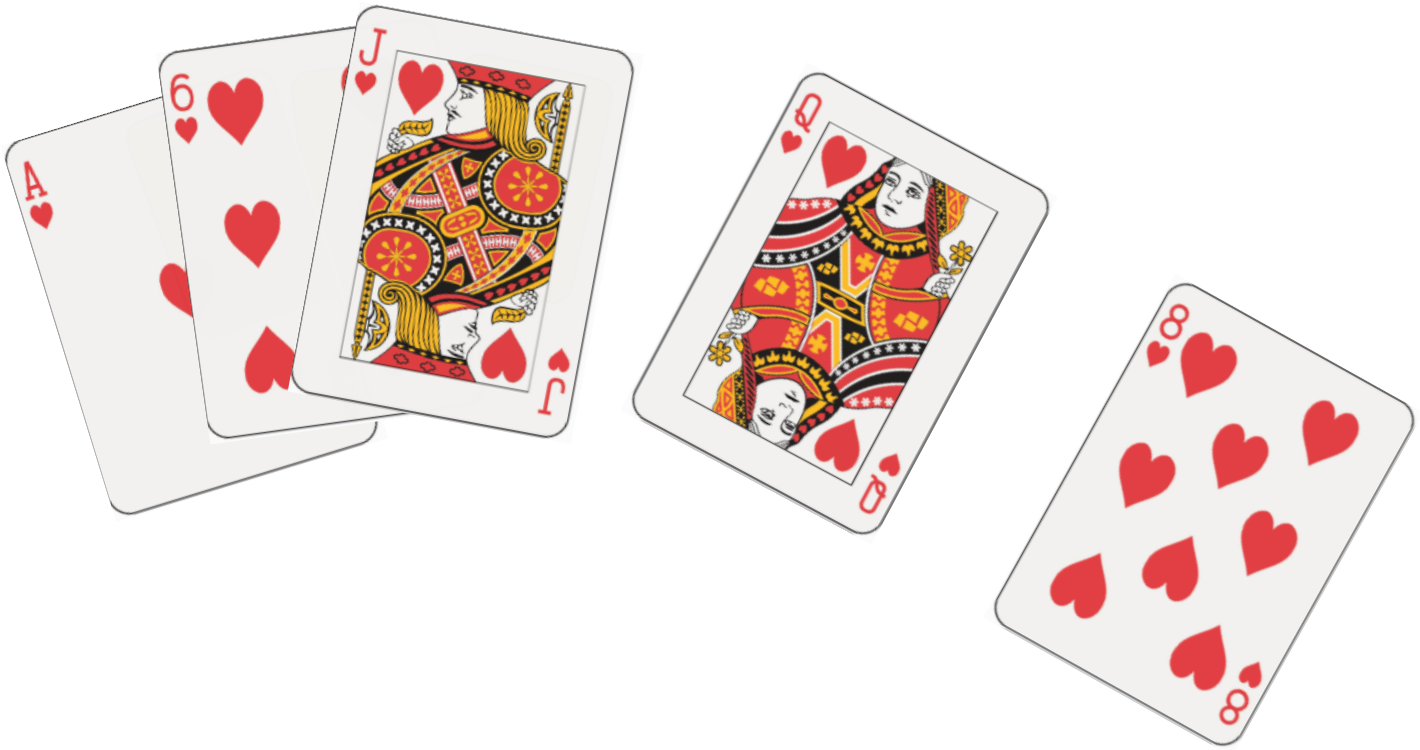
The Insertion Sort Algorithm

1. Sorts the first two elements, which becomes the sorted part of the array
   先將前兩項排序, 此為已排序部份
2. Each remaining element is then **inserted** into the sorted part of the array at the correct location
   再將剩下的一一插入已排序部份

Insertion Sort

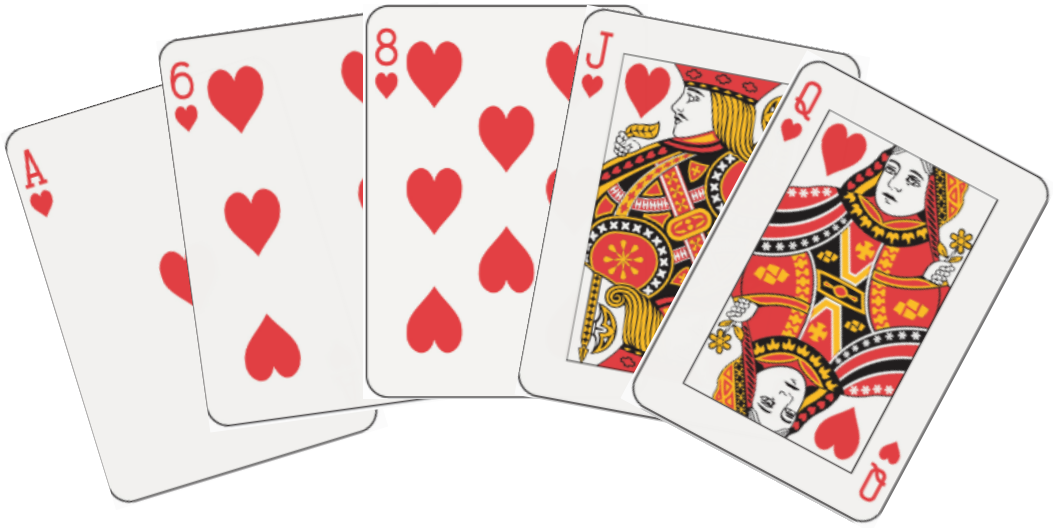
To insert 8, we need to make room for it by moving first Q and then J.
欲插入8, 首先移動Q, 再移動J, 以挪出空間

Insertion Sort



Insertion Sort

Insertion Sort

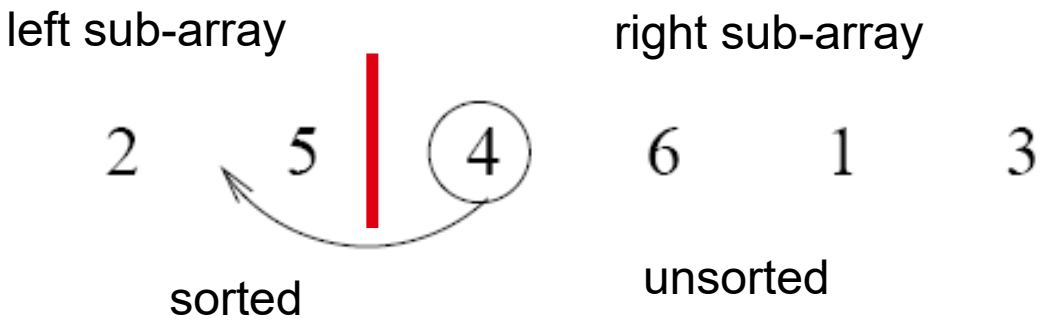


Insertion Sort

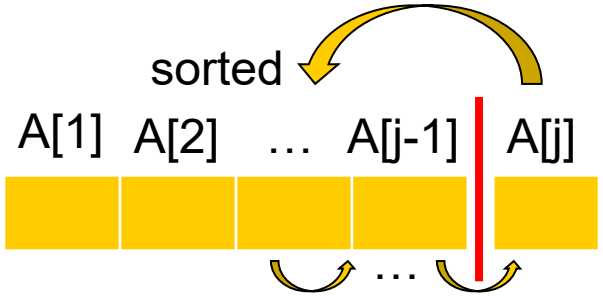
input array

5 2 4 6 1 3

at each iteration, the array is divided in two sub-arrays:
每次疊代過程, 陣列皆分為兩部份

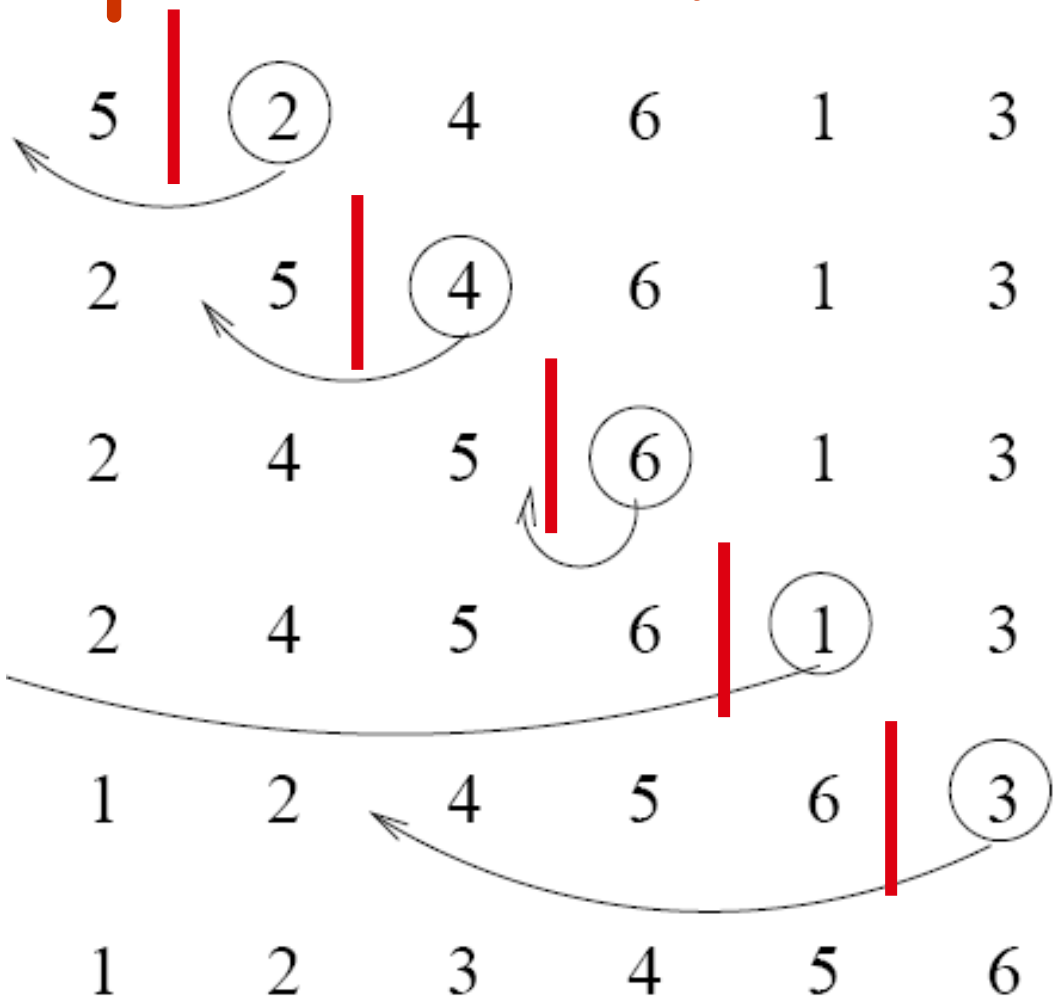


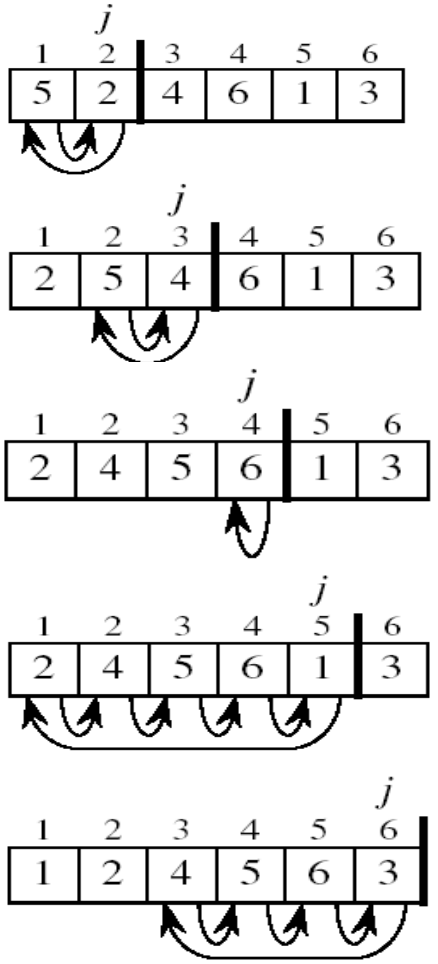
Pseudocode of Insertion Sort




```
Insertion-sort(A)
 1 for j ←2 to length[A]
 2     key←A[j]
 3     // Insert A[j] into the sorted sequence A[1..j-1]
 4      i ← j - 1
 5      while i>0 and A[i]>key
 6           do A[i+1] ←A[i]
 7              i ← i - 1
 8      A[i +1] ←key
```

Operations of Insertion Sort





```
Insertion-sort(A)
1   for j ←2 to length[A]
2       key←A[j]
3       *Insert A[j] into the sorted sequence A[1..j-1]
4        i ← j – 1
5        while i>0 and A[i]>key
6            do A[i+1] ←A[i]
7                 i ← i – 1
8       A[i +1] ←key
```

- *Sorted in place*：在array內重新安排順序
  - ✓ 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*： 迴圈不變量
  - ✓ At the start of each iteration of the for loop of line 1-8, the subarray A[1..j-1] consists of the elements originally in A[1..j-1] but in sorted order.
  - ✓ Check for **Initialization**, **Maintenance**, and **Termination** conditions.
    每次迴圈開始時, 子陣列A[1..j-1]皆已排序.

Pseudocode conventions 虛擬碼慣例

1. 縮排: Indentation indicates block structure.
2. 迴圈: Looping constructs are like in C/C++/Java.
3. 註解: // indicates a comment.
4. 變數: Variables are local.
5. 屬性: Attribute `attr` of object `x`, we write `x.attr`.
6. 參數: Parameters are passed by value.
7. 布林運算:The boolean operators "`and`" and "`or`" are ***short-circuiting***.

Exercise

Linear Search Problem: 線性搜尋

Input: A sequence of n numbers and a value v.

$$A = \langle a_1, a_2, \ldots, a_n \rangle$$

Output: An index i s.t. $v = A[i]$, or NIL

Write pseudocode for ***linear search***.
寫出linear search的虛擬碼

```
for each item in the list:
    if that item has the desired value,
        stop the search and return the item's location.
return NIL.
```

```
for each 陣列元素
    if 符合所求,
        停止搜尋並傳回該元素位置.
return NIL.
```

2.2 Analyzing algorithms 演算法分析

Analyzing an algorithm has come to mean **predicting** the resources that the algorithm requires. 預測該演算法所需資源

- ✓ ***Resources:*** memory, communication, bandwidth, logic gate, **time**.
- ✓ **Assumption**: one processor, ***RAM*** **(Random-access Machine)**
- ✓ (We shall have occasion to investigate models for parallel computers and digital hardware.)

2.2 Analyzing algorithms

- The best notion for **input size** depends on the problem being studied. 以輸入資料多少決定問題大小
- The **running time** of an algorithm on a particular input is the number of primitive operations or "**steps**" executed. 執行時間以所需步驟表示
- It is convenient to define the notion of step so that it is as machine-independent as possible.

Pseudo code of Insertion-Sort

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*$[A]$	c_1	
2 **do** *key* $\leftarrow A[j]$	c_2	
3 ▷ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	
4 $i \leftarrow j-1$	c_4	
5 **while** $i > 0$ and $A[i] >$ *key*	c_5	
6 **do** $A[i+1] \leftarrow A[i]$	c_6	
7 $i \leftarrow i-1$	c_7	
8 $A[i+1] \leftarrow$ *key*	c_8	

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

j值不同時, 迴圈中第5行所執行的次數.

Best case running time 最佳情况

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

$$+ c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

∵already sorted $\quad\therefore t_j = 1 \text{ for } j = 2,3,\ldots n$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$
$$= an + b$$

Linear function on *n*

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*[A]	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Worst case running time 最糟情況

- ✓ $t_j = j$ for j = 2,3,…,n : quadratic function on *n*.

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

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*[A]	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Worst-case and average-case analysis 平均情況

- Usually, we concentrate on finding only on the ***worst-case running time*** *演算法分析重點放在最糟情況*
- Reason:
  - ✓ It is an upper bound on the running time 上界
  - ✓ The worst case occurs fair often 最糟情況常發生
  - ✓ The average case is often as bad as the worst case. For example, the insertion sort. Again, quadratic function. 平均情況通常也好不到哪裡

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.
  主要針對問題大小成長時, 執行時間所增加的速度做分析

Exercise

Write pseudocode for ***selection sort*** algorithm.

寫出「選擇排序法」演算法

```
SELECTION-SORT(A)
n = A.length
for j = 1 to n − 1                    從第1個位置到第n-1個位置依序檢查
    smallest = j                      若目前檢查至第j個位置
    for i = j + 1 to n                則找出第j+1個到最後一個中最小的所在位置
        if A[i] < A[smallest]
            smallest = i
    exchange A[j] with A[smallest]    再與第j個元素的位置對調
```

SELECTION-SORT(A)

$n = A.length$
for $j = 1$ **to** $n - 1$
  $smallest = j$
  for $i = j + 1$ **to** n
    if $A[i] < A[smallest]$
      $smallest = i$
  exchange $A[j]$ with $A[smallest]$

```
void selection_sort(vector<int> & A)
{
  for (unsigned j = 0; j < A.size() - 1; ++j) {
    int smallest = j;
    for (unsigned i = j + 1; i < A.size(); ++i) {
      if (A[i] < A[smallest])
        smallest = i;
    } // end of for
    exchange(A, j, smallest);
  } // end of for
} // end of selection_sort
```

```
void exchange(vector<int> & A, unsigned x, unsigned y)
{
  A[x] ^= A[y];
  A[y] ^= A[x];
  A[x] ^= A[y];
}
```

Exercise

- How to modify any algorithms to have a good best-case running time?

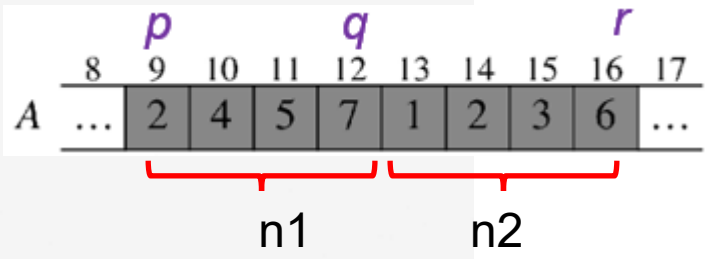
  如何改寫任何演算法，使得當遇到最佳狀況時可以很快的執行完畢?

2.3 Designing algorithms 演算法設計

- There are many ways to design algorithms:

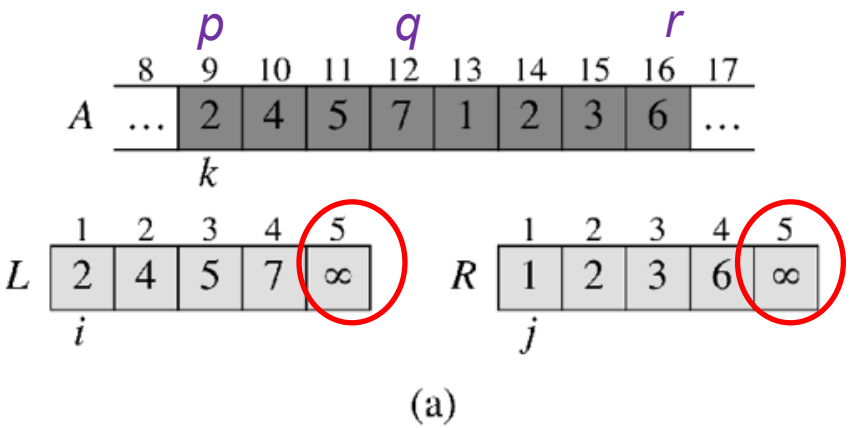
漸進法

- **Incremental approach**: insertion sort
- **Divide-and-conquer**: merge sort 合併排序
  - ✓ recursive: 個別擊破法
    - *divide* 分化
    - *conquer* 征服
    - *combine* 合併




```
MERGE(A, p, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i ← 1 to n1
 5       do L[i] ← A[p + i − 1]
 6  for j ← 1 to n2
 7       do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13       do if L[i] ≤ R[j]
14            then A[k] ← L[i]
15                 i ← i + 1
16            else A[k] ← R[j]
17                 j ← j + 1
```

Example of Merge(A,p,q,r)



(a)

```
1   n1 ← q − p + 1
2   n2 ← r − q
3   create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
```

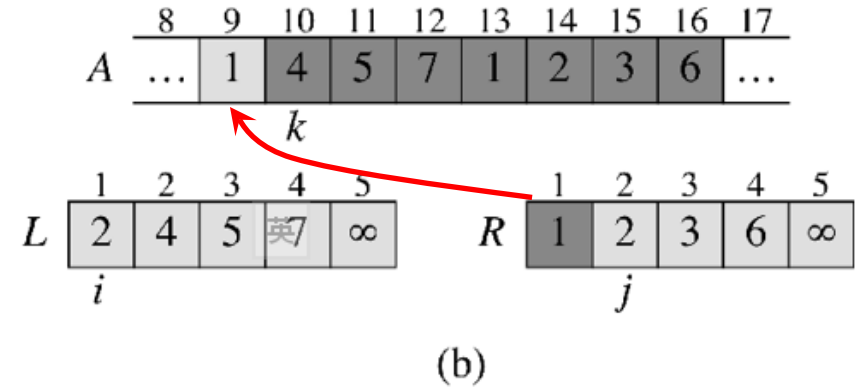
將原陣列分成左右兩個子陣列

```
4   for i ← 1 to n1
5       do L[i] ← A[p + i − 1]
6   for j ← 1 to n2
7       do R[j] ← A[q + j]
```

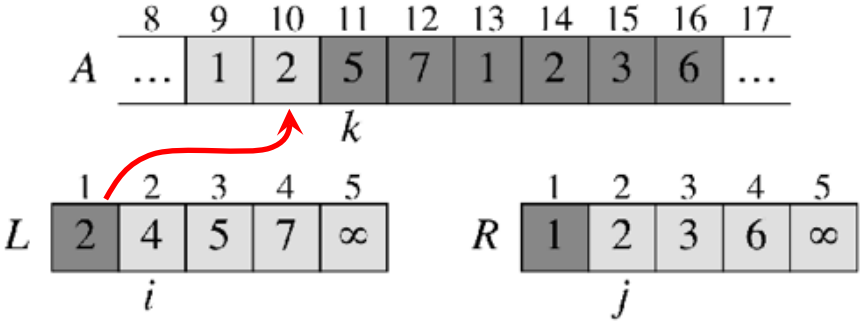
把資料複製到L和R陣列

```
8   L[n1 + 1] ← ∞
9   R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
```

標示L和R陣列的結束記號,
並指出兩陣列中將要比較的位置



(b)

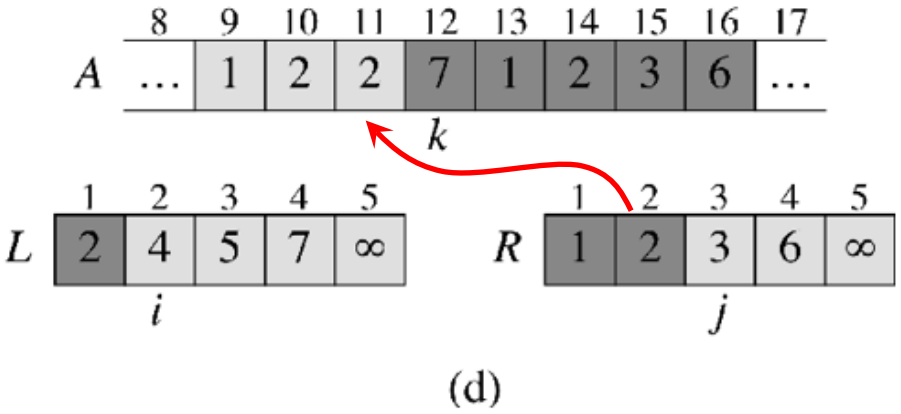


(c)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14  英          then A[k] ← L[i]
15                   i ← i + 1
16          else  A[k] ← R[j]
17                   j ← j + 1
```

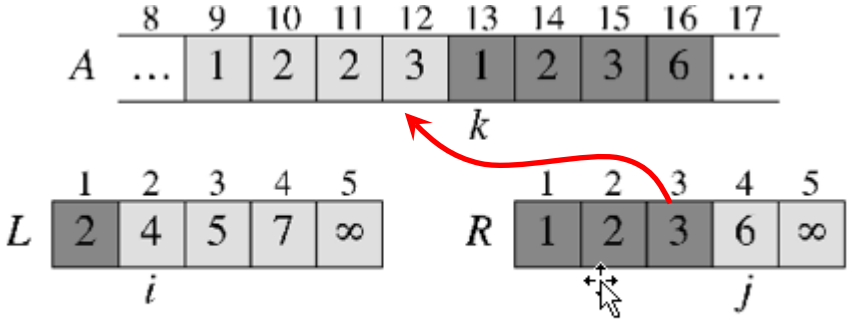
執行第12~17行的迴圈.
把較小的資料放回A陣列.
再準備比較下一組. (依此類推…)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14  英          then A[k] ← L[i]
15                   i ← i + 1
16          else  A[k] ← R[j]
17                   j ← j + 1
```



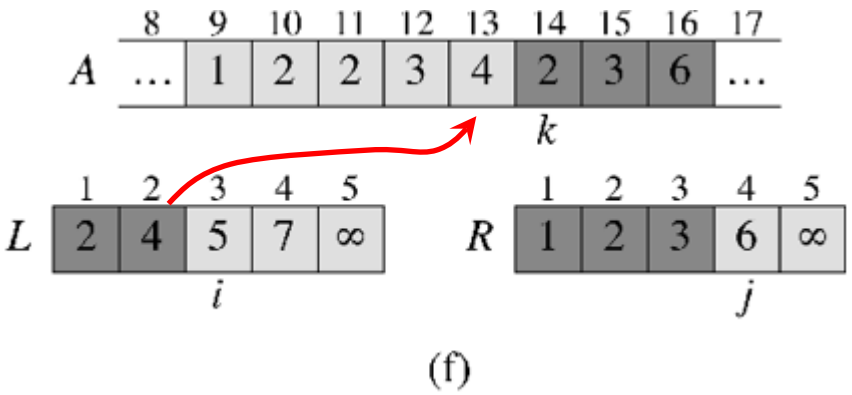
(d)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```



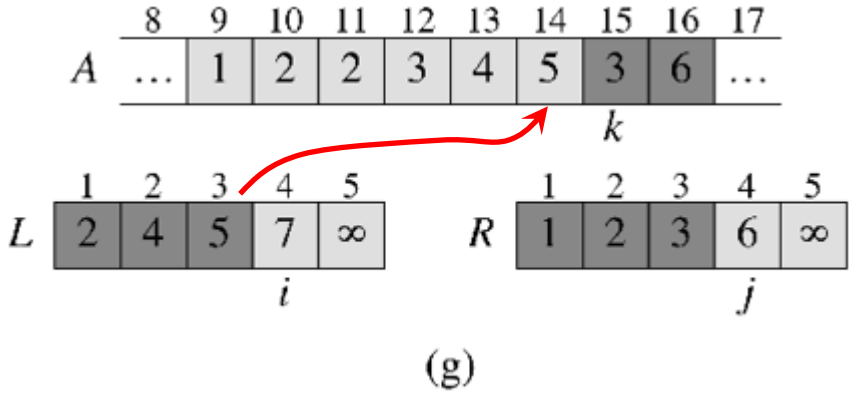
(e)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```



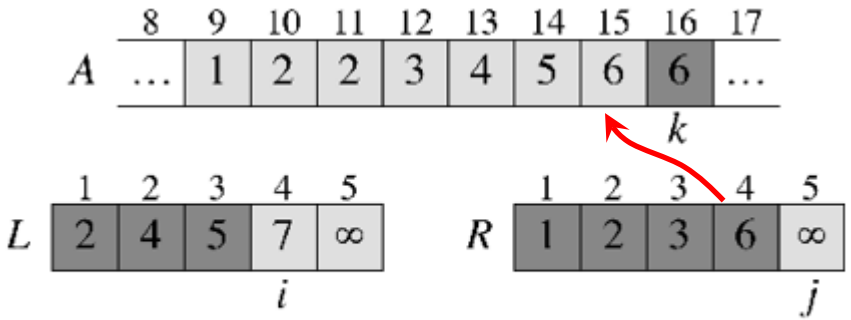
(f)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```



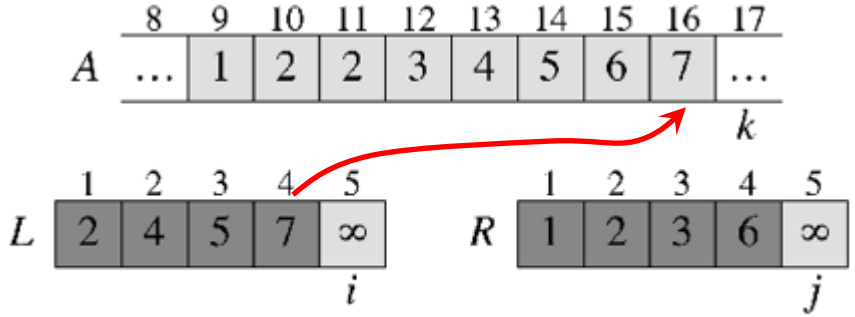
(g)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```



(h)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```



(i)

```
12  for k ← p to r
13      do if L[i] ≤ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```

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)
```

sorted sequence

initial sequence

Analysis of merge sort 合併排序分析

- Analyzing divide-and-conquer algorithms

  只要是遇到分析任何各個擊破演算法, 首先列出通式(遞迴式):

$$T(n) = \begin{cases} \Theta(1) & if \quad n \le c \\ aT(n/b) + D(n) + c(n) & otherwise \end{cases}$$

See Chapter 4

- Analysis of merge sort 合併排序法分析:

$$T(n) = \begin{cases} \Theta(1) & if \quad n = 1 \\ 2T(n/2) + \Theta(n) & if \quad n > 1 \end{cases}$$

$$T(n) = \Theta(n \log n)$$

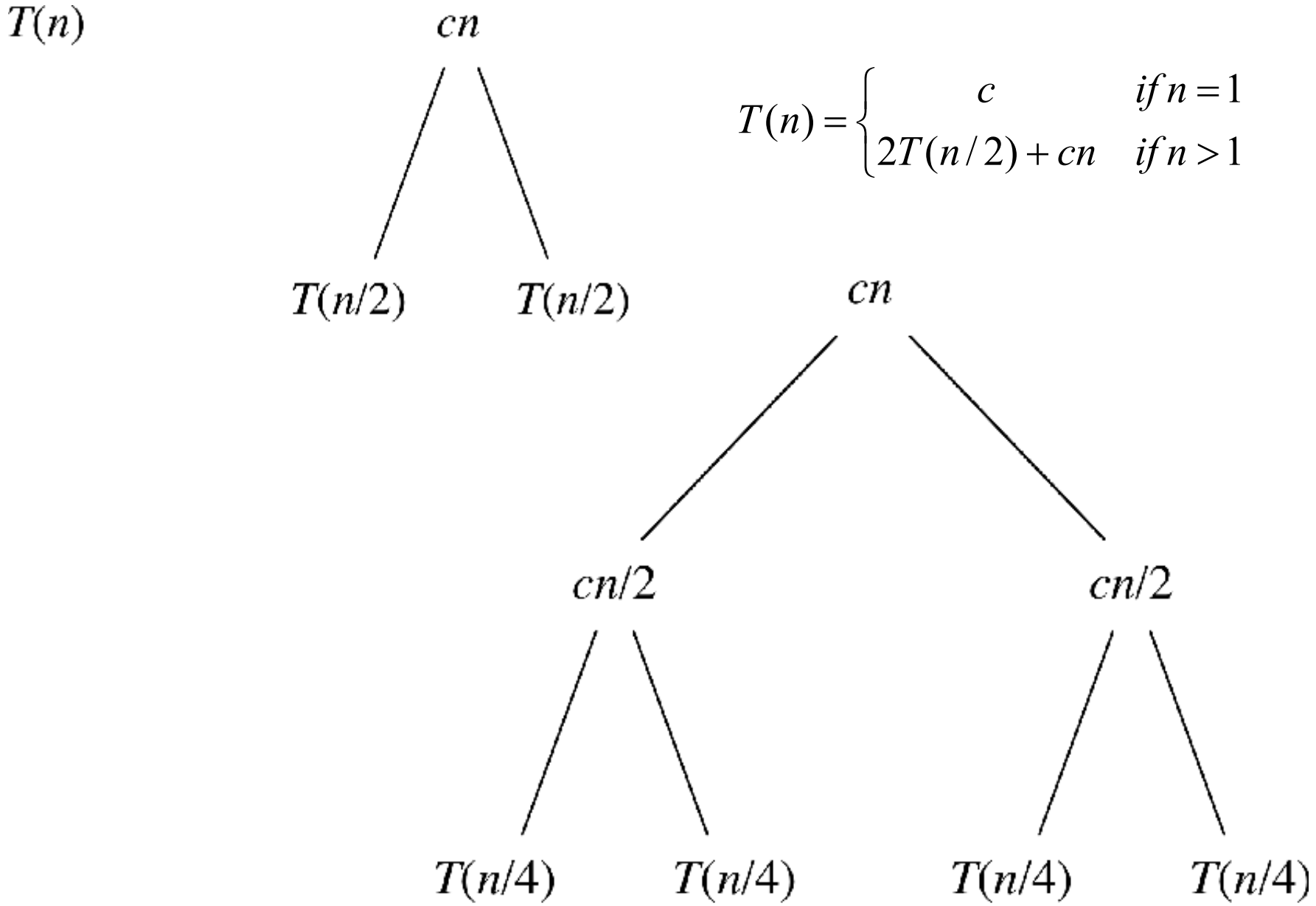
合併排序的執行時間 = 陣列左右兩邊合併排序的時間 + 合併的時間

Analysis of merge sort

$$T(n) = \begin{cases} c & if\ n = 1 \\ 2T(n/2) + cn & if\ n > 1 \end{cases}$$

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

其中 c 為執行一次的(常數)時間.



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

cn

Total: $cn \lg n + cn$

Outperforms insertion sort!

比插入排序快!

Exercise

- 實作 Merge_Sort 演算法

```
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)
```

```
void merge_sort(vector<int> & A, unsigned p, unsigned r)
{
  if (p < r) {
    unsigned q = static_cast<unsigned>(floor((p + r) / 2.0)) + 1;
    merge_sort(A, p, q-1);
    merge_sort(A, q, r);
    merge(A, p, q-1, r);
  } // end of if
} // end of merge_sort
```

```
void merge(vector<int> & A, unsigned p, unsigned q, unsigned r)
{
  unsigned n1, n2, i, j, k;
  n1 = q - p + 1;
  n2 = r - q;
  // create arrarys L[1..n1+1] and R[1..n2+1]
  vector<int> L(n1+1), R(n2+1);
  for (i = 0; i < n1; ++i)
    L[i] = A[p + i];
  for (j = 0; j < n2; ++j)
    R[j] = A[q + 1 + j];
  L[i] = INT_MAX;
  R[j] = INT_MAX;
  i = 0;
  j = 0;
  for (k = p; k <= r; ++k) {
    if (L[i] <= R[j]) {
      A[k] = L[i];
      ++i;
    }
    else {
      A[k] = R[j];
      ++j;
    }
  } // end of for
} // end of merge
```

MERGE(A, p, q, r)

1 $n_1 \leftarrow q - p + 1$
2 $n_2 \leftarrow r - q$
3 create arrays $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$
4 **for** $i \leftarrow 1$ **to** n_1
5 **do** $L[i] \leftarrow A[p + i - 1]$
6 **for** $j \leftarrow 1$ **to** n_2
7 **do** $R[j] \leftarrow A[q + j]$
8 $L[n_1 + 1] \leftarrow \infty$
9 $R[n_2 + 1] \leftarrow \infty$
10 $i \leftarrow 1$
11 $j \leftarrow 1$
12 **for** $k \leftarrow p$ **to** r
13 **do if** $L[i] \le R[j]$
14 **then** $A[k] \leftarrow L[i]$
15 $i \leftarrow i + 1$
16 **else** $A[k] \leftarrow R[j]$
17 $j \leftarrow j + 1$

Summary

- Insertion sort
- Pseudocode conventions
- Analyzing algorithms
  - Best case
  - Worst-case
  - Average-case
  - Order of growth
- Designing algorithms
  - Incremental approach: Insertion sort
  - Divide-and-conquer: Mergesort
    - Analysis of divide-and-conquer algorithm