排序問題 Sorting Problem

這堂課預計上課重點

- 排序問題定義
- 常見的排序演算法
- 排序演算法時間複雜度分析

排序問題 (Sorting Problem)

- **問題描述 (Problem Statement)**
  - 輸入(Input)：給定一個含有n個數字的序列$< a_1, a_2, \ldots, a_n >$
  - 輸出(Output)：一個含有n個數字排序數列$< a'_1, a'_2, \ldots, a'_n >$，使得$a'_1 \le a'_2 \le, \ldots, \le a'_n$

  PS. 這 n 個數字 $a_1, a_2, \ldots, a_n$ 稱之為 Key

- 問題舉例說明
  - 輸入(Input) : A = <5,2,4,6,1,3>
  - 輸出(Output) : A = <1,2,3,4,5,6>

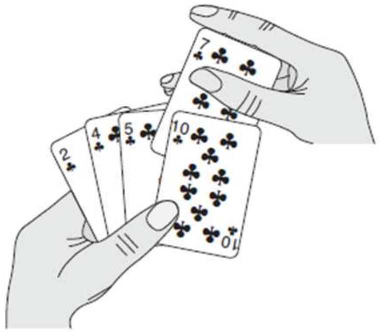
- 演算法表示方式 – 虛擬碼 (Psudocode)
  - 使用任何語言(中英)撰寫，不須侷限於任何語法，只要步驟明確即可。

常見的排序演算法

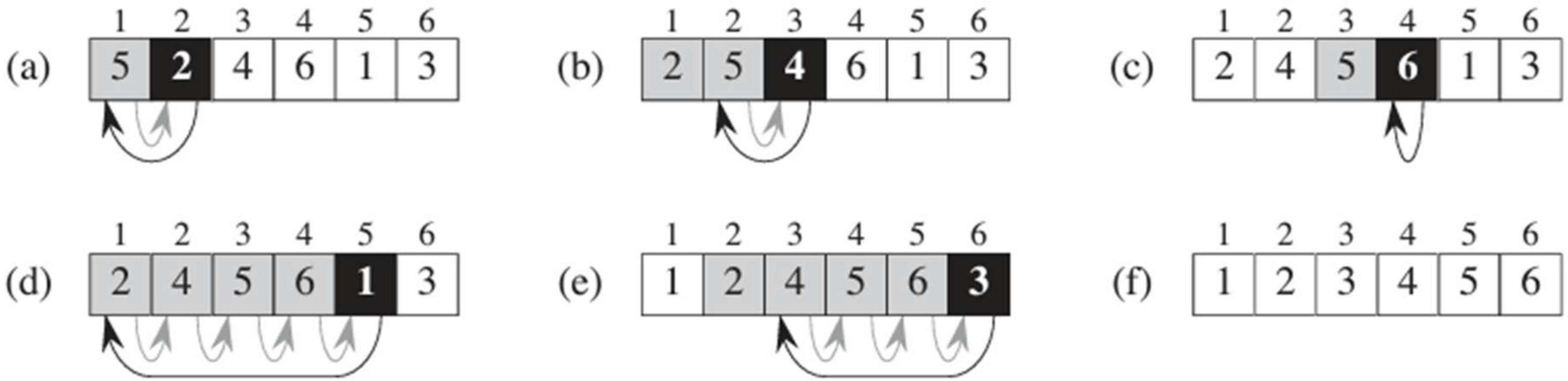
- 插入排序法 (Insertion Sort)
- 合併排序 (Merge Sort)
- 堆積排序 (Heap Sort)
- 快速排序 (Quick Sort)

插入排序法 (Insertion Sort)

插入排序



- 輸入(Input) : A = <5,2,4,6,1,3>
- 輸出(Output) : A = <1,2,3,4,5,6>



每個回合((a)→(f)) 可以確定目標元素會安插到正確的位置。

將目標元素與已排序的資料列中的元素逐一比較，直到找到比目標元素大的元素進行交換，並往前遞移

灰色: 表示已排序的資料

黑色：表示目前要處理的資料，稱之為目標元素。

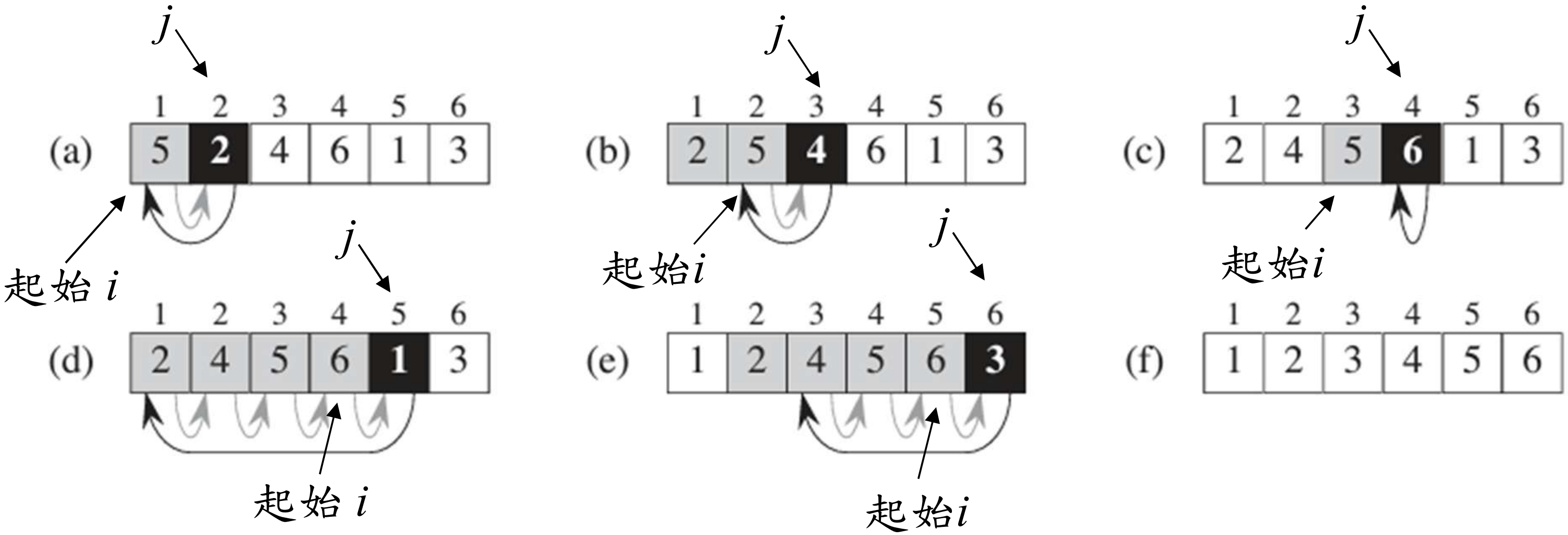
插入排序虛擬碼

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

起始 i

- 將目標元素與已排序的資料列中的元素逐一比較，直到找到比目標元素大的元素進行交換。
- 往前遞移

key的位置 $i+1$



時間複雜度分析

- 執行時間 (Running Time) – 演算法執行基本運算或步驟的數量總和

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

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

Best Case

- 在這個 Case 下執行時間是最少的

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

- Best Case 長甚麼樣子
  - 提示：演算法步驟在這 Case 下某些步驟因為滿足什麼條件可以不執行
  - 花多少時間

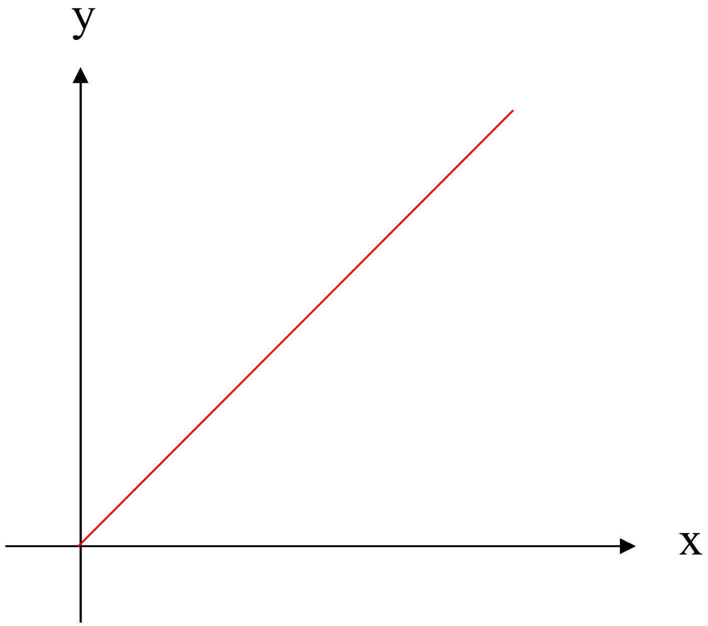
$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

Best Case 分析

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

A 不會因為 n 的大小而改變 **B**

$\Rightarrow$ $T(n) = An + B$ $\Rightarrow$ Linear Function of n $\Rightarrow$ $\Theta(n)$ 或 $O(n)$ 或 $\Omega(n)$



Worst Case

- 在這個 Case 下執行時間是最多的

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

- Worst Case 長甚麼樣子
- 花多少時間

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

Worst Case 分析(1/2)

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

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

所以

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

Worst Case 分析 (2/2)

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

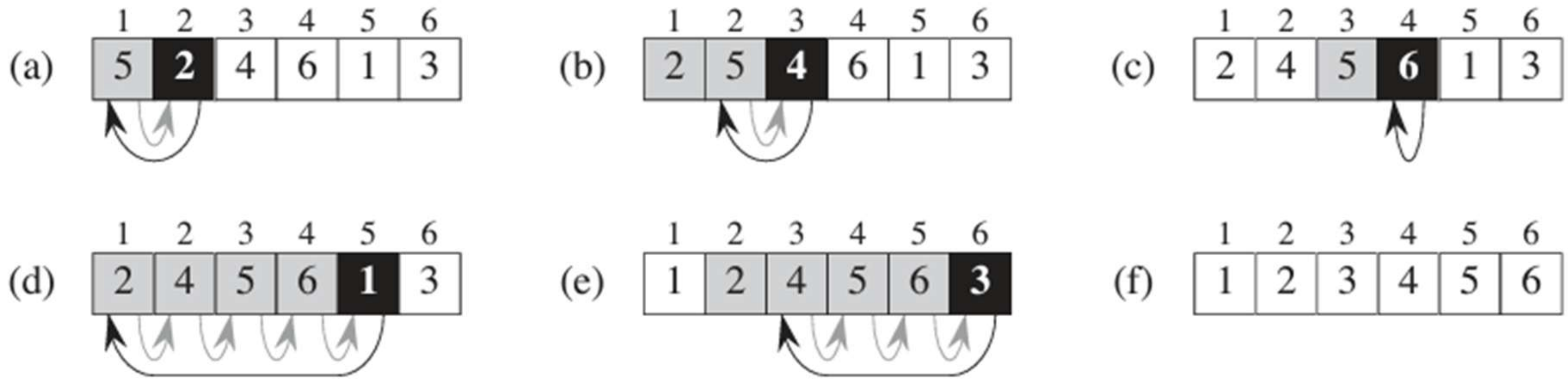
$$\Rightarrow \quad T(n) = \boxed{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)} n^2 + \boxed{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)} n - \boxed{(c_2 + c_4 + c_5 + c_8)} \, .$$

不會因為 n 的大小而改變

$$\Rightarrow \quad T(n) = \mathrm{A}n^2 + \mathrm{B}n + C \quad \Rightarrow \quad \Theta(n^2) \text{ 或 } O(n^2) \text{ 或 } \Omega(n^2)$$

Summary

- 插入排序
  - 時間
    - Best Case : 全部是正向排序，Running Time $\Theta(n)$
    - Worst Case : 全部是反向排序，Running Time $\Theta(n^2)$
    - Average Case : Average Case = Worst Case
  - 空間
    - In-place 演算法 : 演算法執行時不須額外的空間來佔存排序的過程



演算法設計方法
- Divide-and-Conquer Approach

Divide-and-Conquer 方法

- 將一個問題切割成數個很直覺被解決的小問題，每個小問題解決完後再將結果進行合併即完成解決。

- 三大步驟
  - 分割 (Divide) : 將問題切割成數個子問題 (Subproblem)。
  - 擊破 (Conquer)：每個子問題以遞迴(Recursively)的方式解決之，直到每個問題都能夠很直覺被解決。
  - 合併(Combine)：將所有子問題的結果合併。

合併排序 (Merge Sort)

設計理念

自己呼叫自己(遞迴)

- Divide-and-conquer approach
  - 分割：將 1 個 含有 n 個數字的序列，切割成 2 個含有 $\frac{n}{2}$ 個數字的序列。
  - 擊破：使用合併排序演算法將此 2 個含有 $\frac{n}{2}$ 個數字的序列排序。
  - 合併：合併此 2 個已排序含有 $\frac{n}{2}$ 個數字的序列。

合併排序概念

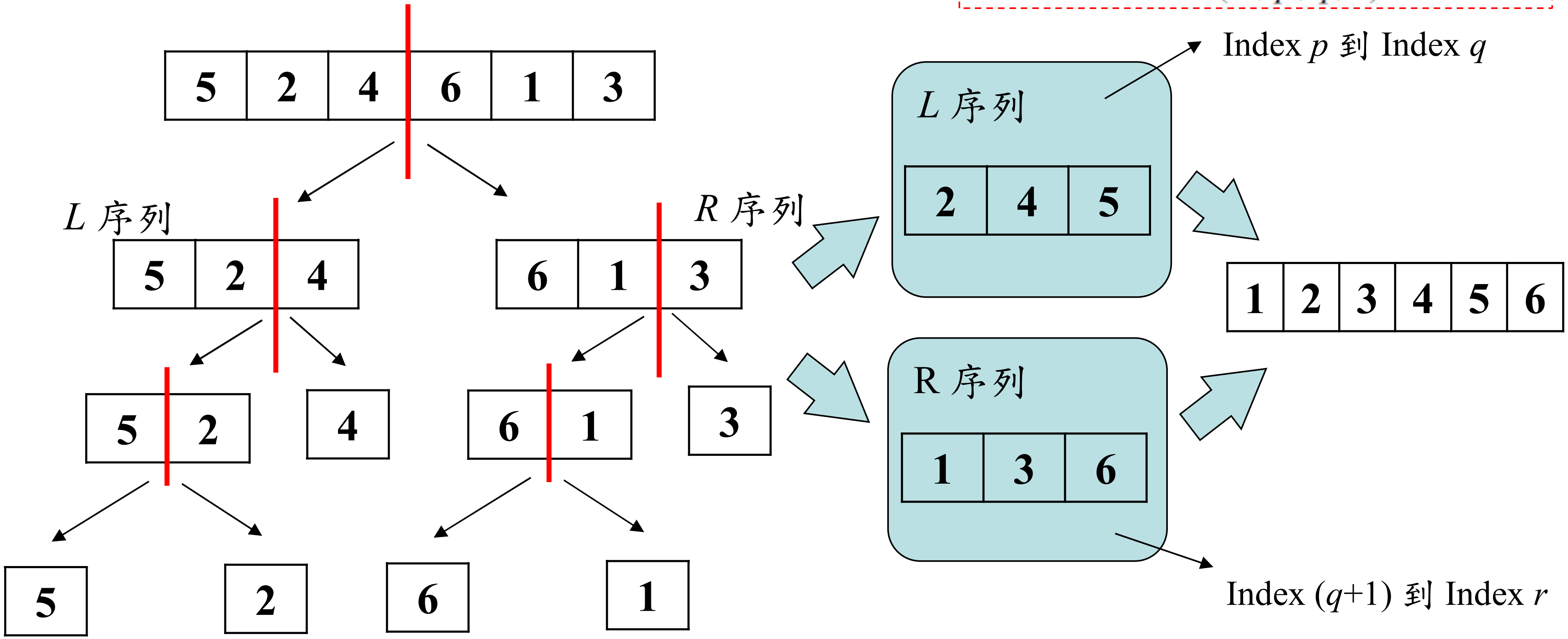
- 輸入(Input) : A = <5,2,4,6,1,3>
- 輸出(Output) : A = <1,2,3,4,5,6>

MERGE-SORT(A, p, r)

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

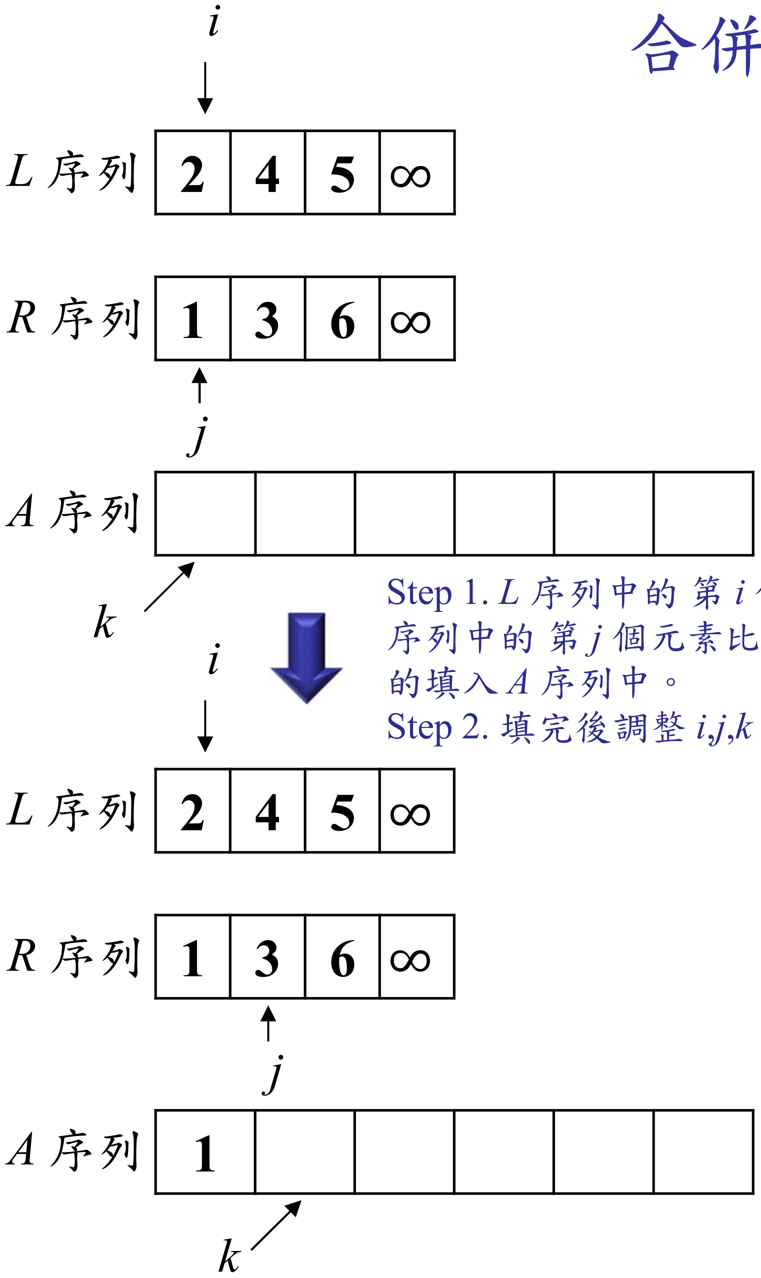
Divide 階段

Merge 階段



合併排序虛擬碼

練習：請大家依照演算法第12-17行步驟將後續拆解圖完成。



Step 1. L 序列中的 第 i 個元素 和 R 序列中的 第 j 個元素比較，數值小的填入 A 序列中。
Step 2. 填完後調整 i,j,k 的位置

MERGE(A, p, q, r)

產生 2 個暫存序列

```
1  n1 = q − p + 1
2  n2 = r − q
3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
4  for i = 1 to n1
5      L[i] = A[p + i − 1]
6  for j = 1 to n2
7      R[j] = A[q + j]
8  L[n1 + 1] = ∞
9  R[n2 + 1] = ∞
```

初始化 i 和 j

```
10 i = 1
11 j = 1
```

決定要填入 A 序列數值

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

回合 1

L 序列 = [2,4,5,∞], key(i) = 2
R 序列 = [1,3,6,∞], key(j) = 3
A 序列 = [1]

合併排序執行時間分析

- 令 T(n) 表示排序 1 個 含有 n 個數字的序列所需要的時間

n = 1 時，執行的時間成本

➡ $T(n) = \Theta(1),\ \text{if } n = 1$

n > 1 時，執行的時間成本

- ✓ 每次分割會將原本序列拆解成 2 個子序列(L 和 R)，每個子序列含有 $\frac{n}{2}$ 個數字 ，因此，2 個子序列 × 排序1個含有$\frac{n}{2}$ 個數字序列時間 = $2\times T(\frac{n}{2})$
- ✓ 每次分割的執行時間 $\Theta(1)$
- ✓ 每次合併的執行時間 $\Theta(n)$

➡ $T(n) = 2\times T(\frac{n}{2}) + \Theta(1) + \Theta(n) = 2\times T(\frac{n}{2}) + \Theta(n)\ ,\ \text{if } n > 1$

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

合併排序執行時間分析 (Recursive Tree 分析法)

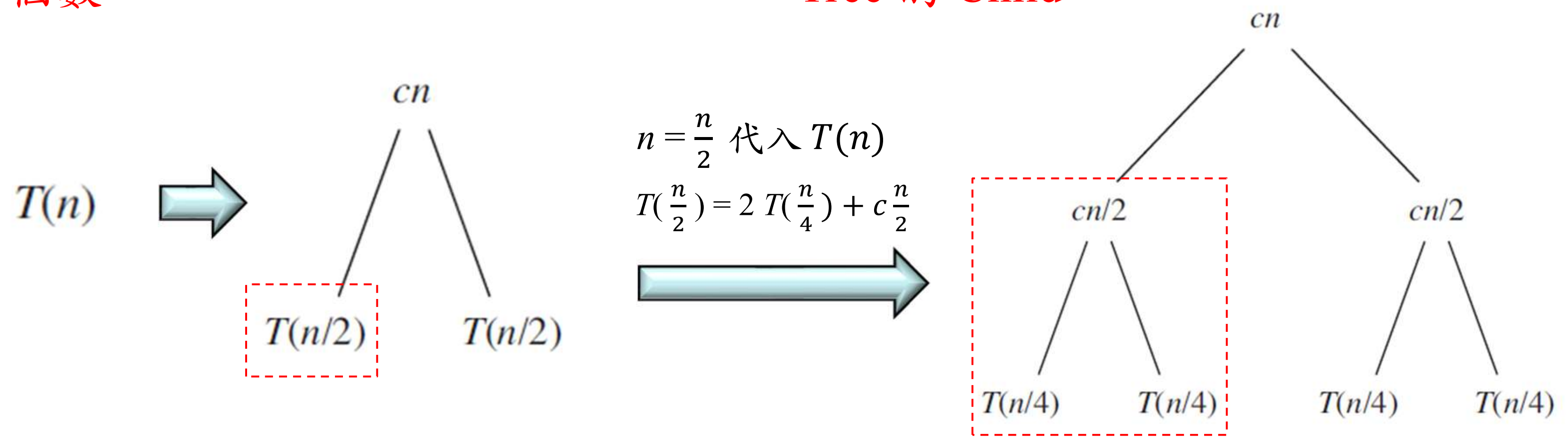
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

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

Tree 的 Root

Child 個數

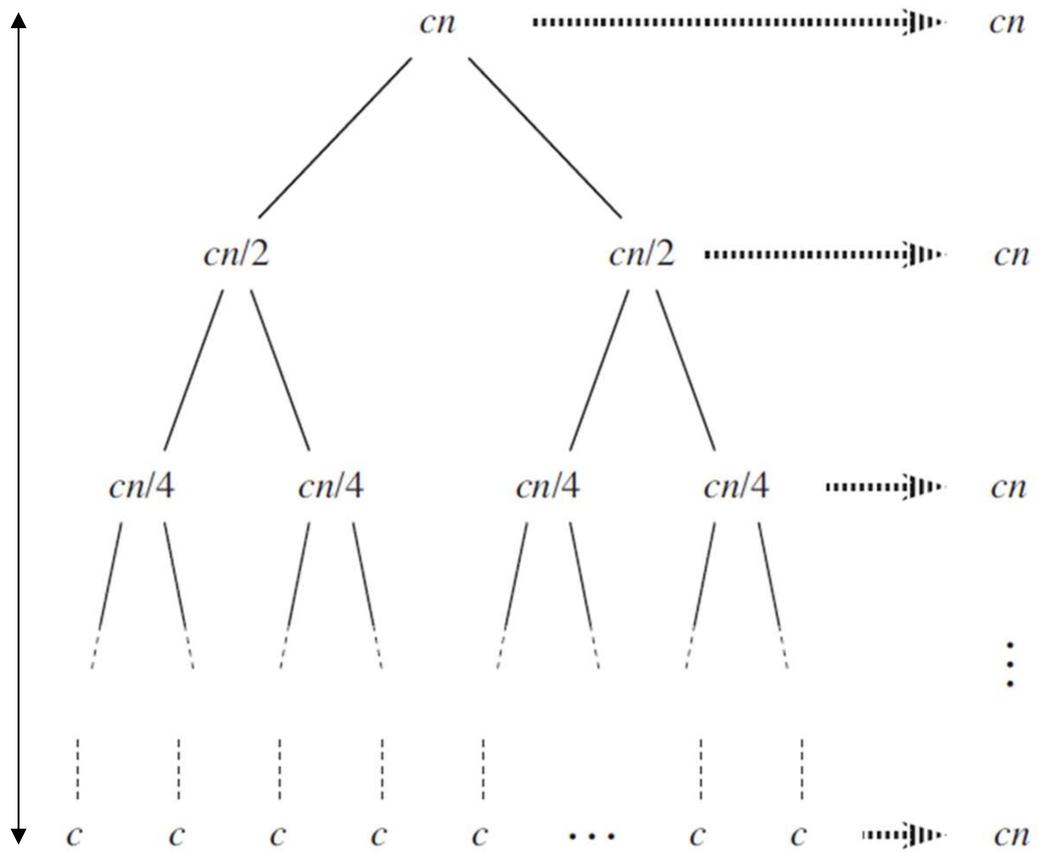
Tree 的 Child



練習：請大家依照此規則畫出第4和5 層的Recursive Tree 。

合併排序執行時間分析 (Recursive Tree 分析法)

Complete Binary Tree
高度 $K = log_2(n+1)$
➔有 (K+1) 層



$$T(n) = cn * (K+1) = cn * log_2(n+1) + cn$$

對數運算公式

(1) $\log_\alpha MN = \log_\alpha M + \log_\alpha N$

(2) $\log_\alpha x = \dfrac{\log_\beta x}{\log_\beta \alpha}$

(3) $\log_\alpha \theta = \dfrac{1}{\log_\theta \alpha}$

Summary

- 合併排序
  - 時間
    - Best Case : N/A
    - Worst Case : N/A
    - Average Case : Θ (cn + cn* log_2n) = Θ ($n$log$n$)
  - 空間
    - Not-In-place 演算法