演算法與程式解題實務

Mong-Jen Kao (高孟駿)

Monday 18:30 – 21:20

Sorting (排序問題)

- 給定 n 個數字 $a_1, a_2, \ldots, a_n$,
  將它們排序，並以非遞減的順序輸出.

- Ex.

  輸入的數字為 3, 2, 8, 6, 10, 12, 2, 1,

  則你的程式需依序輸出 1, 2, 2, 3, 6, 8, 10, 12.

Insertion Sort 演算法

- InsertionSort($A[\,1, 2, \ldots, n\,], n$)

A. For $j \leftarrow 2$ to n, do the following.

   a) $key \leftarrow A[j]$.

   b) $i \leftarrow j - 1$.

   c) While $i > 0$ and $A[i] > key$, do the following.

      1) $A[i+1] \leftarrow A[i]$.

      2) $i \leftarrow i\ - 1$.

   d) $A[i+1] \leftarrow key$.

較直覺且友善的描述方式

- InsertionSort($A[\,1, 2, \ldots, n\,], n$)

A. For $j \leftarrow 2$ to n, do the following.

   a) Find the largest index $i \in [1,2, \ldots, j-1]$ such that $A[i] < A[j]$.
      Set $i \leftarrow 0$ if no such index exists.

   b) Insert $A[j]$ at position $i + 1$
      by moving $A[i+1, \ldots, j-1]$ to $A[i+2, \ldots, j]$.

- InsertionSort($A[\,1, 2, \ldots, n\,], n$)

A. For $j \leftarrow 2$ to n, do the following.

   a) Find the largest index $i \in [1,2, \ldots, j-1]$ such that $A[i] < A[j]$.
      Set $i \leftarrow 0$ if no such index exists.

   b) Insert $A[j]$ at position $i + 1$
      by moving $A[i+1, \ldots, j-1]$ to $A[i+2, \ldots, j]$.

Lemma. (The **Invariant condition** of the algorithm)

At the end of each for-loop in step A.,
the numbers in $A[1,2, \ldots, j]$ are always sorted in order.

Time Complexity (Efficiency) of an Algorithm

- 我們以一個演算法執行過程中，
  所使用到的 **「運算步驟」**數目，來衡量此演算法的**效率**。

  我們將此定義為此演算法的「**時間複雜度 (Time Complexity)**」 / **執行時間**。

  - 由於演算法執行過程中，實際的步驟數會因 input 不同而異，
    我們通常會考慮演算法的 ***Worst-cast running time***.

    亦即，最差情況下的執行時間.

Running Time of Insertion Sort

- InsertionSort($A[\,1, 2, \ldots, n\,], n$)

A. For $j \leftarrow 2$ to n, do the following.

   a) Find the largest index $i \in [1,2, \ldots, j-1]$ such that $A[i] < A[j]$.
      Set $i \leftarrow 0$ if no such index exists.

   b) Insert $A[j]$ at position $i+1$
      by moving $A[i+1, \ldots, j-1]$ to $A[i+2, \ldots, j]$.

Running Time of Insertion Sort

- The *worst-case running time* of insertion sort on n input numbers is

$$\sum_{2\le j\le n}(j-1) \;=\; n(n-1)/2 \;\boxed{=\; O\left(n^2\right).}$$

This says, “roughly at most n^2”.
We will define what this means next lecture.

  – ***The analysis is tight***, as there is indeed an instance that makes InsertionSort to take this number of steps.

Algorithms

What are algorithms?

Why do we need Algorithms?

為什麼我們需要 (更好的) 演算法？

-- 做為可更有效率解決龐大計算問題的工具

- 考慮以下的計算問題.

以字母順序 (Alphabetical order) 排序台灣所有居民的身份證字號.

- 台灣的人口約 2.3×10^7 (二千三百萬).
- 若我們使用 InsertionSort 或是 BubbleSort 演算法,
  那麼排序這些身份證字號所需的計算時間將會**「超過一週」**.
- 然而, 若我們使用更聰明的排序方法, 所需的時間可以降低至**「5秒以內」**.

為什麼我們需要 (更好的) 演算法？

-- 做為可更有效率解決龐大計算問題的工具

- 若我們使用 InsertionSort 或是 BubbleSort 演算法,
  那麼排序這些身份證字號所需的計算時間將會「**超過一週**」.
- 然而, 若我們使用更聰明的排序方法, 所需的時間可以降低至「**5秒以內**」.

■ 因此，
好的演算法在有嚴格時限 (Time-Critical) 的大型計算應用中，是不可或缺的。

- 例如： google map, 導航系統, 火車、航班排程系統等.

The Merge-Sort Algorithm

The Merge-Sort (合併排序) Algorithm

- Let $a_1, a_2, \dots, a_n$ be the input numbers.
- The merge-sort algorithm works as follows.

1. 將 input 約略切為兩等份

$$L = \{a_1, \dots, a_{\lfloor n/2 \rfloor}\} \text{ and } R = \{a_{\lfloor n/2 \rfloor + 1}, \dots, a_n\}.$$

2. 分別使用「合併排序法」排序 L 與 R.
3. 將 L 與 R 合併為一個排序好的序列.

The Merge-Sort Algorithm

- A more detailed pseudo-code for this algorithm.

<ins>Algorithm MergeSort</ins>($A[1,2,\dots,n], \mathrm{left}, \mathrm{right}$)

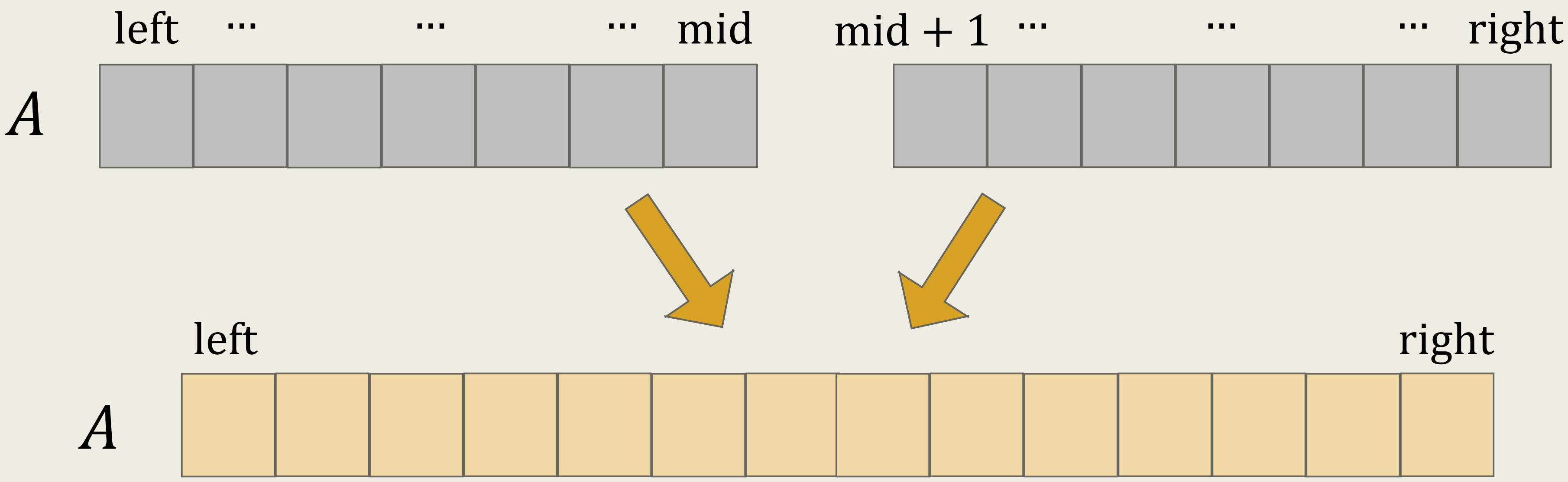
1. If $\mathrm{left} = \mathrm{right}$, then return.
2. Let $\mathrm{mid} \leftarrow \lfloor(\mathrm{left} + \mathrm{right})/2\rfloor$.
3. Call MergeSort(A, left, mid) and MergeSort(A, mid+1, right).
4. Merge $A[\mathrm{left},\dots,\mathrm{mid}]$ and $A[\mathrm{mid}+1,\dots,\mathrm{right}]$
   with the procedure Merge($A, \mathrm{left}, \mathrm{mid}, \mathrm{right}$).

The Procedure Merge$(A, \text{left}, \text{mid}, \text{right})$

- The procedure takes two sorted lists

$$L \coloneqq A[\text{left}, \dots, \text{mid}] \quad \text{and} \quad R \coloneqq A[\text{mid}+1, \dots, \text{right}]$$

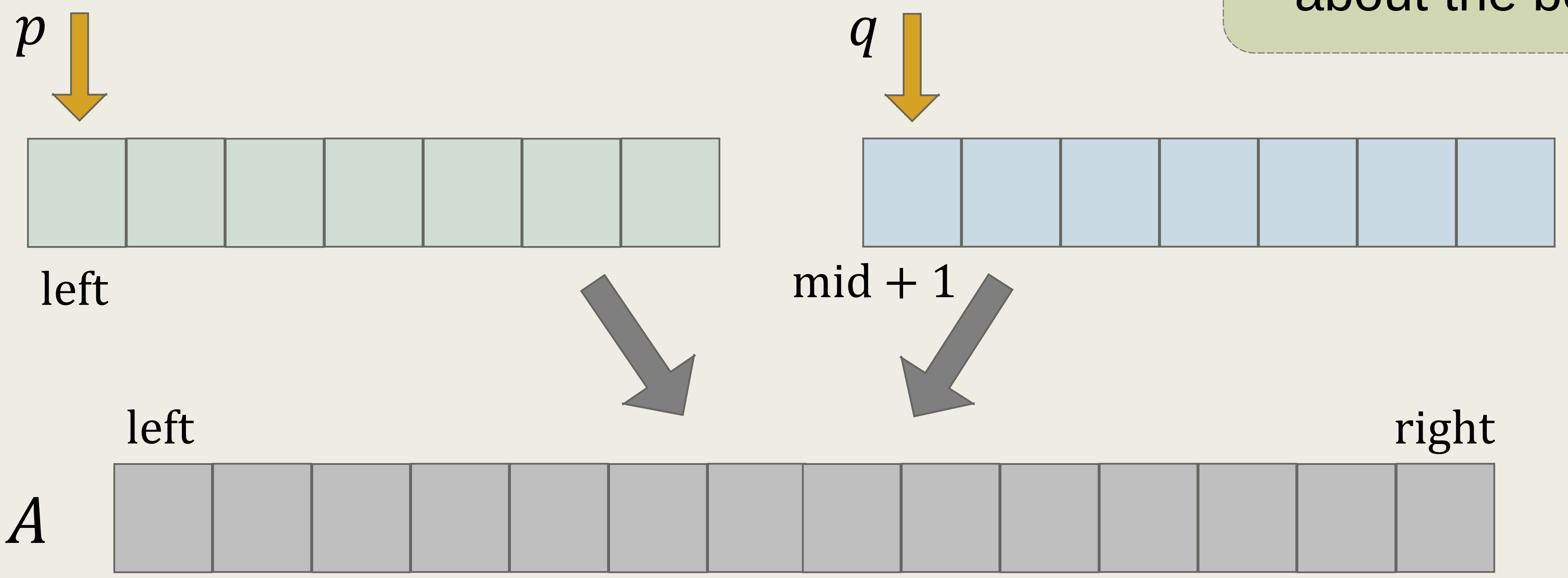
and merge them into one sorted list $A[\text{left}, \dots, \text{right}]$.



- The procedure uses two pointers p and q to iterate over L and R.
  - In each iteration, it picks the smaller between p and q to the new sequence and advances it.
  - Repeats until L and R are scanned.

The idea is simple.

Have to be careful
about the boundary cases.



- The procedure Merge($\cdot$) uses an extra array $\mathrm{temp}[1, \ldots, n]$.

<u>Procedure Merge</u>($A[1,2, \ldots, n]$, left, mid, right)

1. Copy $A[\mathrm{left}, \ldots, \mathrm{right}]$ to $\mathrm{temp}[\mathrm{left}, \ldots, \mathrm{right}]$.

   $p \leftarrow \mathrm{left}, \ q \leftarrow \mathrm{mid} + 1, \ \mathrm{pos} \leftarrow \mathrm{left}$.

2. While $p \leq \mathrm{mid}$ and $q \leq \mathrm{right}$, do the following.

   - If $\mathrm{temp}[p] < \mathrm{temp}[q]$, then set $A[\mathrm{pos}++] \leftarrow \mathrm{temp}[p++]$.

     Otherwise, set $A[\mathrm{pos}++] \leftarrow \mathrm{temp}[q++]$.

3. While $p \leq \mathrm{mid}$, set $A[\mathrm{pos}++] \leftarrow \mathrm{temp}[p++]$.

4. While $q \leq \mathrm{right}$, set $A[\mathrm{pos}++] \leftarrow \mathrm{temp}[q++]$.

Analysis of the Procedure Merge(·)

- 正確性 – 為什麼此程序可正確合併 L 與 R?
  - Provided that L and R are already sorted, the smaller of $\mathrm{temp}[p]$ and $\mathrm{temp}[q]$ must be the smallest element among $\mathrm{temp}[p, \ldots, \mathrm{mid}]$ and $\mathrm{temp}[q, \ldots, \mathrm{right}]$.

- The time complexity of this procedure is

$$2 \cdot (\mathrm{right} - \mathrm{left} + 1) = O(\mathrm{right} - \mathrm{left} + 1),$$

i.e., linear in the number of elements.

Analysis of the Algorithm MergeSort(·)

- MergeSort (合併排序法) 的正確性
  - Proved by induction on $m \coloneqq \mathrm{right} - \mathrm{left} + 1$.
  - When $m = 1$, the procedure MergeSort(A, left, right) clearly sorts $A[\mathrm{left}]$ correctly.
  - When $m > 1$,
    by induction hypothesis, MergeSort sorts L and R correctly.
    Then, we have shown that the procedure Merge(·) merges L and R into a sorted list.

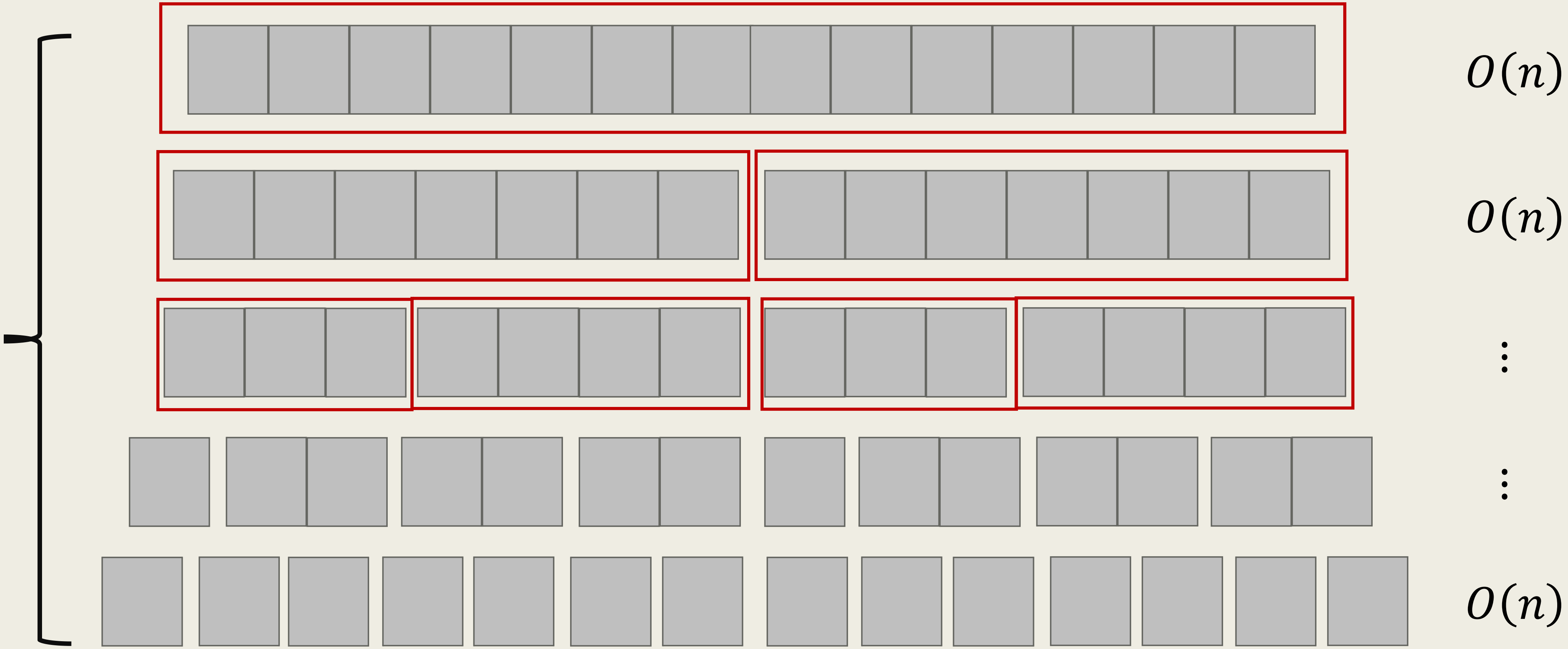
Analysis of the Algorithm MergeSort(·)

- Time complexity of Merge-Sort.
  - For any $n \geq 1$,
    let $T(n)$ be the number of steps required by MergeSort algorithm.
  - Then, we have

$$T(n) = \begin{cases} O(1), & \text{if } n \leq 1, \\ 2 \cdot T\left(\frac{n}{2}\right) + O(n), & \text{otherwise.} \end{cases}$$

$O(\log n)$ levels in total

Total time taken by Merge$(\cdot)$



In total, it takes $O(n \log n)$ time.

Analysis of the Algorithm MergeSort(·)

- Time complexity of Merge-Sort.
  - For any $n \geq 1$,
    let $T(n)$ be the number of steps required by MergeSort algorithm.
  - Then, we have

$$T(n) = \begin{cases} O(1), & \text{if } n \leq 1, \\ 2 \cdot T\left(\frac{n}{2}\right) + O(n), & \text{otherwise.} \end{cases}$$

And $T(n) = O(n \log n)$.