



# 計算思維與人工智慧應用導論

## 功能模組 & 演算邏輯

Dr. Chih-Hsun Wu

吳致勳 助理教授

國立政治大學人工智慧跨域研究中心

Date: 2023/9/26

本投影片僅供教學用途，  
所用圖檔都盡量附上原始來源，  
如有侵權煩請告知，將立即修正

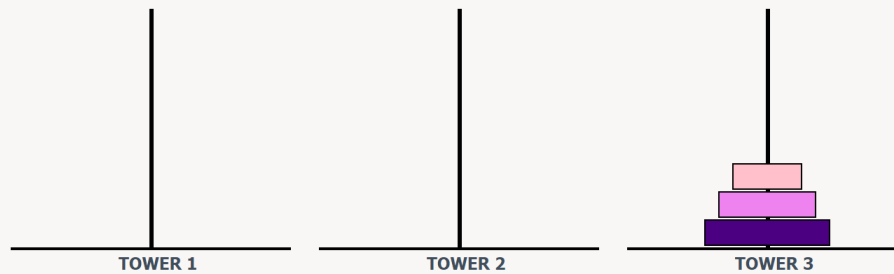
# 課程進度

週次	課程主題	課程內容與指定閱讀
1	計算思維簡介	社會情境脈絡與未來發展 書目：1, 2, 3
2	計算思維	基本內涵與核心概念 書目：1, 2, 3
3	功能模組	問題拆解與型態辨認 書目：4, 5, 6
4	功能模組	抽象思考與演算邏輯 書目：4, 5, 6
5	國慶日	國定假日
6	類比至數位轉換 & 電腦運算架構	類比與數位訊號的基礎概念及類比轉換至數位訊號的原理 & 電腦組成元件與其運算架構 書目：7 chapter 1 & 4 & 5
7	大數據應用	大數據中資料科學的基礎分析概念與商業相關應用
8	學習成果測試	期中評量/作業活動
9	運算思維測驗	國際運算思維挑戰賽
10	人工智慧發展	人工智慧發展歷程與未來趨勢
11	人工智慧技術與應用	人工智慧各式技術與應用案例 書目：8
12	人工智慧應用場景	人工智慧跨域應用
13	人工智慧學習模型實作	Nocode AI 練習 – Rapidminer 書目：9
14	人工智慧倫理	生成AI (如：ChatGPT、Deepfake、Midjourney)、假新聞及未來人工智慧應用上的倫理問題
15	人工智慧專題	海報展示
16	計算思維與人工智慧	期末報告
17	彈性補充教學	人工智慧相關競賽經驗交流
18	彈性補充教學	校園人工智慧應用發想

# 河內塔遊戲

Tower of Hanoi

## Tower of Hanoi



No. of disks	<input type="text" value="3"/>
Minimum no. of moves	<input type="text" value="7"/>
Your no. of moves	<input type="text" value="7"/>
<input type="button" value="Instructions"/> <input type="button" value="Restart"/> <input type="button" value="Undo"/> <input type="button" value="Solve"/>	

<https://tygtw.ddns.net/game/hanoi/>

隨堂練習: 自訂 disks，將完成結果截圖

$f(n)=2^n-1$  有多大？

- 如果 $N=15$ ，最少需移動32767次
  - 如果一個人從 3 歲到 99 歲，每天移動一塊圓盤，他最多僅能移動 15 塊
- **隨堂練習**:如果 $N=64$ ，需要  $2^{64}-1$  步才能完；若每秒可完成一個盤子的移動，需要 ??年才能完成。
- #整個宇宙現在也不過 137 億年。

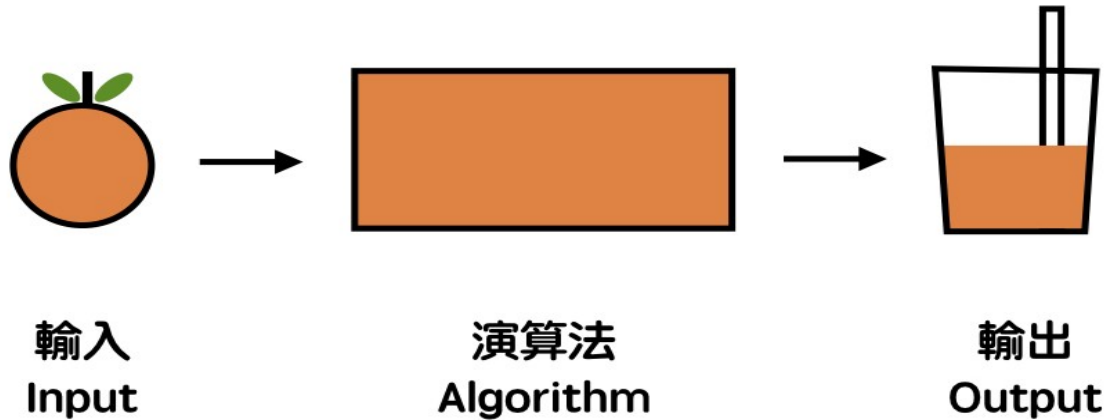
Credit by [https://en.wikipedia.org/wiki/Tower\\_of\\_Hanoi](https://en.wikipedia.org/wiki/Tower_of_Hanoi)

# 演算法

- 定義一件工作如何執行的一組步驟。例如，我們有烹飪的演算法（稱為食譜）、演奏音樂的演算法（樂譜）、變魔術的演算法。
- 在一部如電腦之類的機器能夠完成某一件任務之前，用來完成該項任務的演算法必須先被發現，並且轉換成與這部機器相容的形式。這種與機器相容的形式就稱為程式（program）
- 而開發程式，亦即將演算法以與機器相容的形式編碼，並將之安插入機器即稱為程式設計（programming）
- 程式和其所代表的演算法合稱為軟體（software），有別於稱為硬體（hardware）的機器本身

# 演算法

- 輸入 + 演算法 = 輸出
- 把步驟具體寫成程式，用來達成特定目的的過程



<https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E8%AB%87%E4%BB%80%E9%BA%BC%E6%98%AF%E6%BC%94%E7%AE%97%E6%B3%95%E5%92%8C%E6%99%82%E9%96%93%E8%A4%87%E9%9B%9C%E5%BA%A6-b1f6908e4b80>



#### BIRTH:

Winterthur, Switzerland, February 15 1934.

#### EDUCATION:

Bachelor's degree in Electronics Engineering (Swiss Federal Institute of Technology Zürich—ETH Zürich, 1959); M.Sc. (Université Laval, Canada, 1960); Ph.D. in Electrical Engineering and Computer Science (EECS) (University of California, Berkeley, 1963).

#### EXPERIENCE:

Assistant Professor of Computer Science (Stanford University, 1963—1967); Assistant Professor (University of Zurich, 1967—1968). Professor of Informatics (ETH Zürich, 1968—1999) (one-year sabbaticals at Xerox PARC 1976–1977 and 1984–1985).

## NIKLAUS E. WIRTH

Switzerland – 1984

#### CITATION

For developing a sequence of innovative computer languages, EULER, ALGOL-W, MODULA and PASCAL. PASCAL has become pedagogically significant and has provided a foundation for future computer language, systems, and architectural research.



SHORT  
ANNOTATED  
BIBLIOGRAPHY



ACM TURING  
AWARD  
LECTURE



RESEARCH  
SUBJECTS

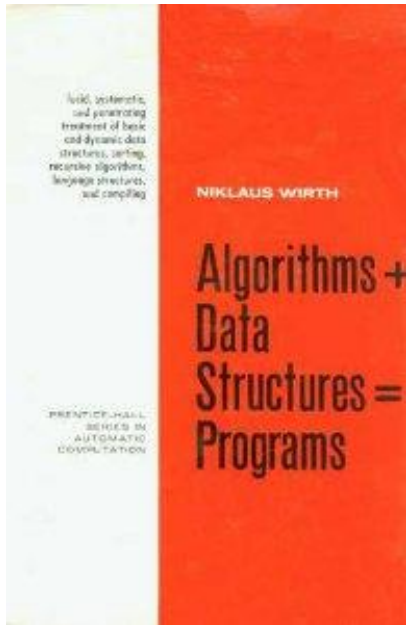


ADDITIONAL  
MATERIALS

**Niklaus Wirth grew up in Switzerland, and he spent most of his professional life at the Swiss Federal Institute of Technology (ETH) in Zürich.** After earning his first degree there in 1959, he left for graduate study

# 演算法簡介

- Turing Award 得獎人Nicklaus Wirth於1975年出版一本書
  - Algorithms + Data Structures = Programs
  - 說明程式是由「演算法」與「資料結構」組成。



Wirth, Niklaus (1976). [Algorithms + Data Structures = Programs](#). Prentice-Hall. [ISBN 978-0-13-022418-7](#). 0130224189.



# 演算法的定義

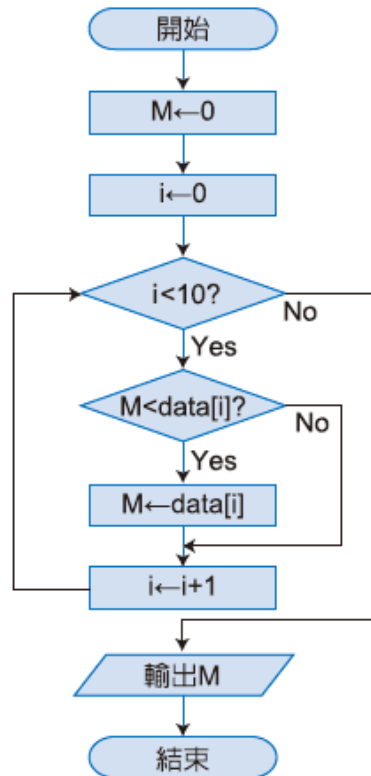
- 簡單的來說，就是「解決問題的步驟」。演算法是有限個命令的集合，其目的是為了解決某一項特定工作。
- 並具有下列特性（缺一不可）：
  1. 輸入(input)：可以有零個以上的輸入資料。
  2. 輸出(output)：至少需有一個以上的輸出資料。
  3. 明確性(definiteness)：每個指令都必須是「非模擬兩可」的明確指令。
  4. 有限性(finiteness)：追蹤演算法的實行，必須能在有限個步驟後停止。
  5. 有效性(effectiveness)：每個指令都應該是基本的，並且能夠透過紙與筆加以模擬。換句話說，它必須是一個可實現的運算。同時整個演算法也必須能夠得到正確的結果，因此本特性又稱為正確性(Correctness)。

## 演算法的範例

- 設計一個尋找10個整數中最大值的演算法
  - Input：10 個整數存放在資料  $\text{data}[0] \sim \text{data}[9]$  中。
  - Output：輸出M，M是  $\text{data}[0] \sim \text{data}[9]$  資料的最大值。
    - Step1： $M \leftarrow 0$
    - Step2： $i \leftarrow 0$
    - Step3：若  $i < 10$ ，則執行Step4，否則執行Step6
    - Step4：若  $M < \text{data}[i]$ ，則  $M \leftarrow \text{data}[i]$
    - Step5： $i \leftarrow i + 1$ ，執行Step3
    - Step6：輸出M

## 演算法的範例

- 請設計流程圖來尋找10個整數中最大值的演算法



# 演算法的表達方式

- 演算法的閱讀對象是「人」，它可使用多種方式表達，常見的表示方式如下：

1. 文字與數字：

- 此處所指的文字是自然語言的文字，如中文、英文、法文等。

2. 虛擬語言(Pseudo-Language)：

- 虛擬語言是一種混合自然語言與高階程式語言的特殊語言，常見的有PASCAL-LIKE, SPARKS等等。使用虛擬語言撰寫的演算法，比較容易轉換為電腦可執行的程式。

- [What's an algorithm? - David J. Malan](#)

3. 流程圖(Flowchart)：

- 一般流程圖(flowchart)與各類流程圖也可以用來表示小型簡單的演算法，例如工作流程圖(Workflow Diagram)、資料流程圖(DFD)、控制流程圖(CFD)等等。

4. 程式語言：

- 雖然演算法是供人閱讀，但實際應用於電腦領域時，最終亦須透過程式來實現，因此，直接使用程式語言表達演算法也是一種方式。

# 演算法 by pseudo code

- 請設計一個Max演算法，功能為尋找10個整數中的最大值。
  - Algorithm Max(data,M)
  - Input：10個整數存放在陣列data[0]~data[9]中。
  - Output：輸出M，M是data[0]~data[9]陣列元素的最大值。
  - $M \leftarrow 0$
  - for(i=0;i<10;i++){
  - 若  $M < \text{data}[i]$ ，則  $M \leftarrow \text{data}[i]$
  - }
  - return M
  - End of Algorithm
- 符合演算法的五大特性，並使用介於C語言與自然語言的方式撰寫而成



# Analysis of Algorithms

---



# 什麼是好的演算法？

- 時間複雜度

- 一支程式/演算法在電腦上需要執行多久？

- 空間複雜度

- 一支程式/演算法在執行時需要耗費多少的記憶體資源？

# 演算法分析

- 研究演算法所需用到如運算時間及儲存空間等資源之議題的重要性，這個領域稱為演算法分析（algorithm analysis）。
- 正確性
- 效率
  - 「最佳」狀況分析（best-case analysis）
  - 「最糟」狀況分析（worst-case analysis）
  - 「平均」狀況分析（average case analysis）



# 演算法的效能評估

- 透過電腦來解決問題，則演算法需要轉換為程式來實現，而在撰寫程式，除了正確性的需求外，還會關注程式的執行效率
  - 記憶體的使用量
  - 執行的速度
- 評估一個演算法的效能時，則可對應為
  - 空間複雜度 (Space Complexity)
    - 演算法執行時所需要的記憶體空間，它又可以分為固定的記憶體空間需求及變動的記憶體空間需求
  - 時間複雜度 (Time Complexity)
    - 一個程式的執行時間可透過特定的時間函式求得，例如C語言的clock()、time()等函式
    - 對於演算法的每一個敘述而言，終究會存在某一個敘述是被執行最多次的，我們只需要關注於該敘述會被執行幾次即可。

## 演算法的時間複雜度

```
sum=0
```


```
for i in range(1,10):
```

```
    for j in range(1,10):
```

```
        sum=sum+i*j
```

```
print(sum)
```

- 演算法被執行最多次的敘述為
  - $\text{sum}=\text{sum}+i*j$
  - 被執行 $n^2$ 次
  - 故時間複雜度為  $O(n^2)$
- 雙層迴圈演算法的時間複雜度為  $O(n^2)$

- 
- 時間複雜度：衡量演算法執行好壞的工具
  - 大 O 符號：用來描述演算法在輸入  $n$  個東西時，所需時間與  $n$  的關係
  - 在  $n$  非常大時，好的演算法設計可以省下非常多時間
  - 演算法的速度不是以秒計算，而是以步驟次數
  - 實務上，我們只會紀錄最高次方的那一項，並忽略其所有的係數

<https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E5%BE%9E%E6%99%82%E9%96%93%E8%A4%87%E9%9B%9C%E5%BA%A6%E8%AA%8D%E8%AD%98%E5%B8%B8%E8%A6%8B%E6%BC%94%E7%AE%97%E6%B3%95-%E4%B8%80-b46fece65ba5>

- **$O(n)$**

- 時間複雜度為  $O(n)$  的演算法，代表著執行步驟會跟著輸入  $n$  等比例的增加。當  $n = 8$ ，程式就會在 8 個步驟完成。

- **$O(\log n)$**

- 時間複雜度為  $O(\log n)$  的演算法（這邊的  $\log$  都是以二為底），代表當輸入的數量是  $n$  時，執行的步驟數會是  $\log n$ 。（當  $\log n = x$  的意思是  $n = 2^x$ ）
- 當  $n = 4$ ，程式會在 2 個步驟完成（ $4 = 2^2$ ）； $n = 16$  時，程式會在 4 個步驟完成（ $16 = 2^4$ ），以此類推。

<https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E5%BE%9E%E6%99%82%E9%96%93%E8%A4%87%E9%9B%9C%E5%BA%A6%E8%AA%8D%E8%AD%98%E5%B8%B8%E8%A6%8B%E6%BC%94%E7%AE%97%E6%B3%95-%E4%B8%80-b46fece65ba5>

# 排序演算法

---

# 排序演算法

- 搜尋與排序是程式設計的一項基本且重要的問題。
  - 所謂『搜尋』( Searching )，指的是在一堆資料中，尋找您所想要的資料
    - 例如：在英文字典中找尋某一個單字。
  - 所謂『排序』( Sorting )則是將一堆雜亂的資料，依照某個鍵值 ( Key Value ) 由大到小或由小到大依序排列，方便日後的搜尋(Searching)與比對(matching)
    - 例如：英文字典中每個單字就是『從a~z』排序後的結果。
  - 以搜尋英文字典為例，您或許有自己的一套方法找到所需要的單字，例如：查單字「good」，您可能會先翻閱字典的前1/3，看看該頁中的單字字首是哪一個字母，然後再略為調整頁數，直到找到資料為止。
  - 而在電子字典中，您只需要輸入「good」，然後電腦或翻譯機就會自動幫您找到「good」這個單字的相關資訊，問題是電腦是如何幫您找到這個單字的呢？這就是研究『搜尋』演算法所關心的問題了。

# 排序演算法

- 氣泡排序法(Bubble sort)
- 插入排序法(Insertion sort)
- 快速排序法(Quick sort)
- 合併排序法(Merge sort)
- 選擇排序法(Selection sort)
- 堆積排序法(Heap sort)
- 謝耳排序法(Shell sort)
- 二元樹排序法(Binary tree sort)
- 基數排序法(Bucket sort)

# Bubble sort

---





# 氣泡排序法

- 氣泡排序法(Bubble Sort)又稱交換排序法
- 從第一筆資料開始，逐一比較相鄰兩筆資料，如果兩筆大小順序有誤則做交換，反之則不動，接者再進行下一筆資料比較
- 所有資料比較完第1回合後，可以確保最後一筆資料是正確的位置。

<https://ithelp.ithome.com.tw/articles/10276184>

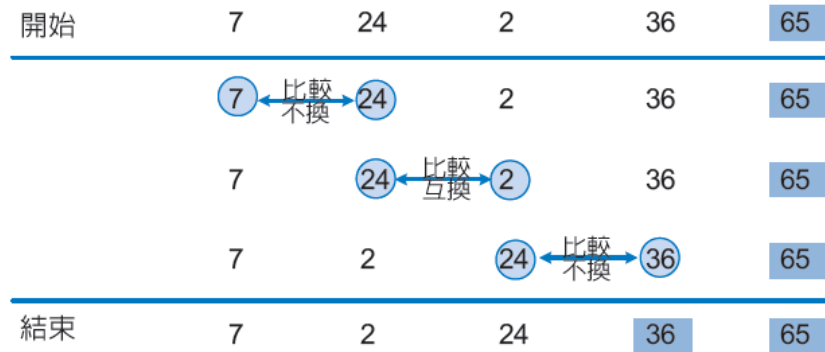
# 氣泡排序法

- 有5筆資料為 24, 7, 36, 2, 65
- 第1回合掃描（比較4次）：回合結束時，65已排序完成。

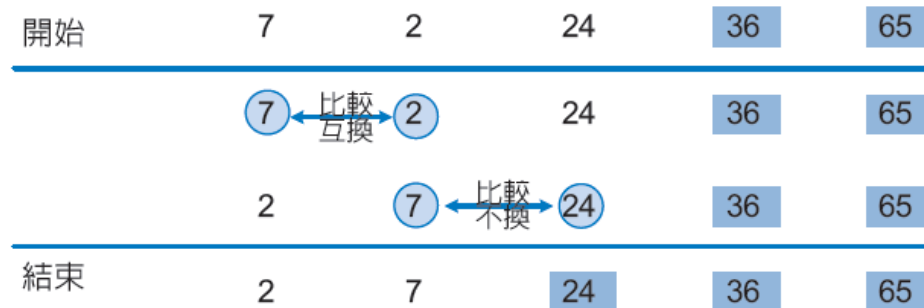
原始值	24	7	36	2	65
	24	7	36	2	65
	7	24	36	2	65
	7	24	36	2	65
	7	24	2	36	65
結束	7	24	2	36	65

# 氣泡排序法

- 第2回合掃描（比較3次）結束時，36,65已排序

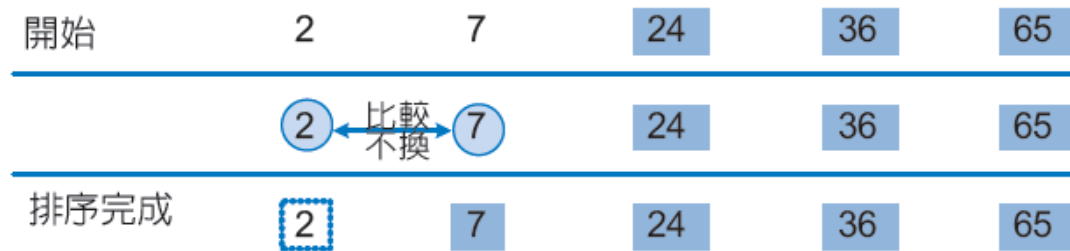


- 第3回合掃描（比較2次）結束時，24,36,65已排序



## 氣泡排序法

- 第4回合掃描（比較1次）結束時，7,24,36,65 已排序完成。



- 第4回合掃描結束時，後面四個最大的鍵值已排序完成，由於只有五筆資料，因此，最小值不需排序就必定位於最前面。
- 故5筆資料需要4回合掃描，共比較 $4+3+2+1=10$ 次。  
 $n=5, (n-1)+(n-2)+\dots+1$

## 氣泡排序法效能討論

- 最佳狀況

- 資料事先已排序完成，此時只需要進行一回合掃描，因此只進行 $n-1$ 次比較，時間複雜度為 $O(n)$ 。

- 最差狀況

- 進行 $(n-1)+(n-2)+\dots+1=n(n-1)/2$ 次比較，時間複雜度為 $O(n^2)$ 。



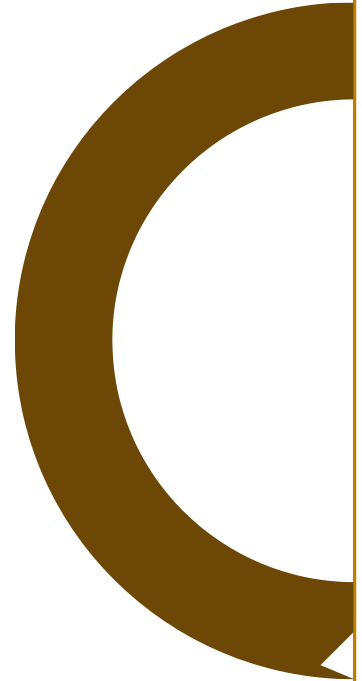
# Insert Sort

---



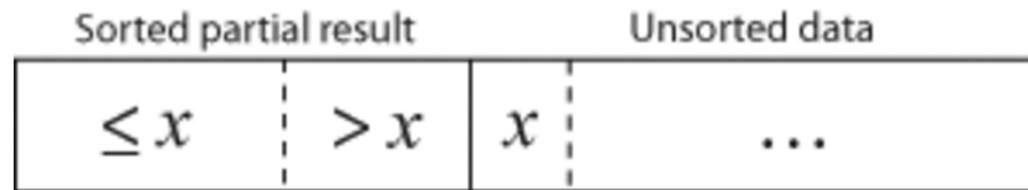
# Insert Sort

6 5 3 1 8 7 2 4

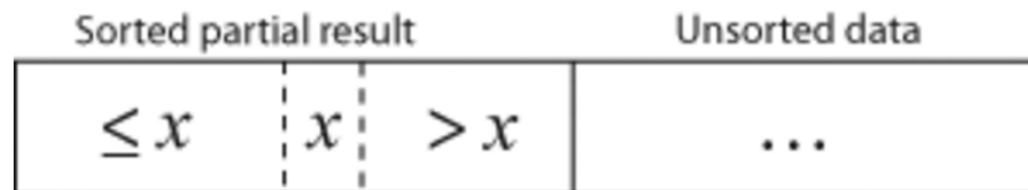


# Insert Sort

- 最簡單的排序演算法之一
- 將資料分成已排序、未排序兩部份
- 1. 讀一個數字  
未排序中的第一筆(正處理的值)
- 2. 插入合適位置: 依序由未排序中的第一筆(正處理的值)，插入到已排序中的適當位置
  - 插入時透過掃描由右而左比較，直到遇到第一個比正處理的值小的值，再插入
  - 比較時，若遇到的值比正處理的值大或相等，則將值往右移



becomes





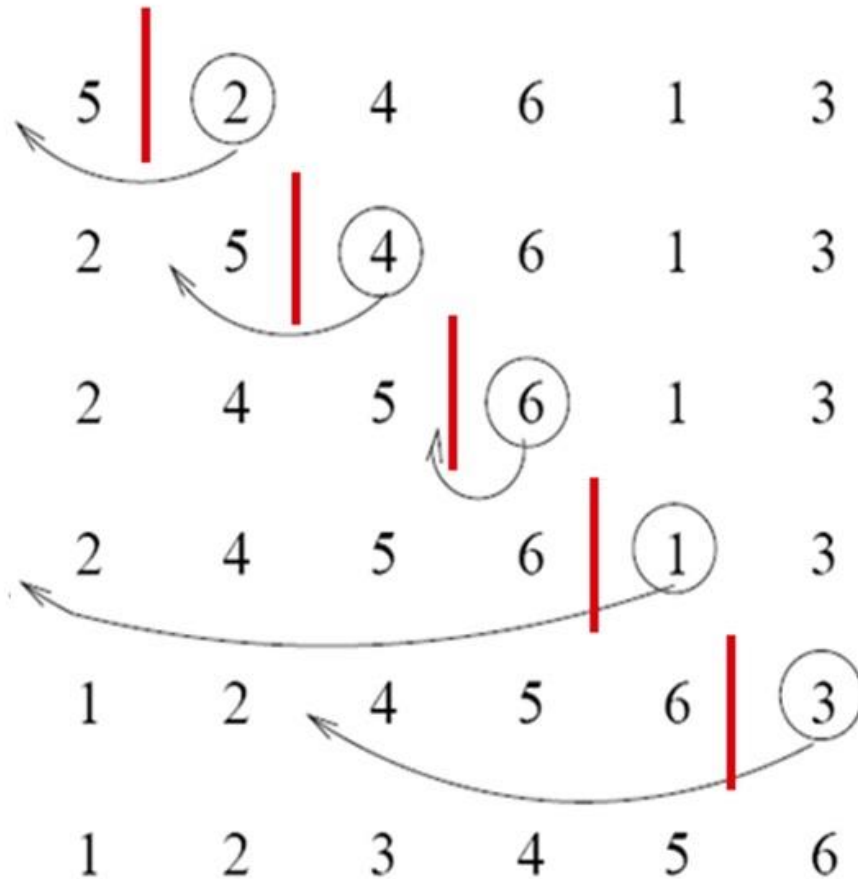
# Insert sort

- Insertion Sort 和打撲克牌時，從牌桌上逐一拿起撲克牌，在手上排序的過程相同。
- 舉例：輸入： $\{5\ 2\ 4\ 6\ 1\ 3\}$ 。
- 首先拿起第一張牌，手上有  $\{5\}$ 。
- 拿起第二張牌 2，把 2 insert 到手上的牌  $\{5\}$ ，得到  $\{2\ 5\}$ 。
- 拿起第三張牌 4，把 4 insert 到手上的牌  $\{2\ 5\}$ ，得到  $\{2\ 4\ 5\}$ 。
- 以此類推。

<https://zh.wikipedia.org/zh-tw/%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F>

<http://notepad.yehyeh.net/Content/Algorithm/Sort/Insertion/1.php>

# Insert sort



<https://kopu.chat/%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8Finsertion-sort/>  
<https://www.101computing.net/insertion-sort-algorithm/>

# Insert sort

Round 1  
41 33 17 80 61 5 55

33 比 41 小，插入 41 前面

Round 2  
33 41 17 80 61 5 55

17 比 33 小，插入 33 前面

Round 3  
17 33 41 80 61 5 55

80 比 41 大，插入 41 後面

Round 4  
17 33 41 80 61 5 55

61 比 80 小，比 41 大，插在兩數中間

Round 5  
17 33 41 61 80 5 55

5 比 17 小，插入 17 前面

Round 6  
5 17 33 41 61 80 55

55 比 61 小，比 41 大，插在兩數中間

Round 7  
5 17 33 41 55 61 80

排序完成！

<https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E6%BE%92%E5%BA%8F%E6%B3%95%E5%85%A5%E9%96%80-%E9%81%B8%E6%93%87%E6%8E%92%E5%BA%8F%E8%88%87%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F%E6%B3%95-23d4bc7085ff>

# Insert sort

Input list = 10 20 30 40 50

PASS	SORTED LIST	UNSORTED LIST	TEST
1	<u>10</u>	20 30 40 50	10 < 20
2	10 <u>20</u>	30 40 50	20 < 30
3	10 20 <u>30</u>	40 50	30 < 40
4	10 20 30 <u>40</u>	50	40 < 50
	10 20 30 40 50	<empty>	

Comparisons required = 4

*Minimum number of insertion sort comparisons =  $N - 1$*

# Insert sort

- Number of Comparisons

Input list = 50 40 30 20 10

PASS	SORTED LIST	UNSORTED LIST	TEST
1	<u>50</u>	40 30 20 10	50 > 40
2	40 <u>50</u>	30 20 10	50 > 30
	<u>40</u> 50	30 20 10	40 > 30
3	30 40 <u>50</u>	20 10	50 > 20
	30 <u>40</u> 50	20 10	40 > 20
	<u>30</u> 40 50	20 10	30 > 20
4	20 30 40 <u>50</u>	10	50 > 10
	20 30 <u>40</u> 50	10	40 > 10
	20 <u>30</u> 40 50	10	30 > 10
	<u>20</u> 30 40 50	10	20 > 10
	10 20 30 40 50	<empty>	

Comparisons required = 10

*Maximum number of insertion sort comparisons =  $1/2(N^2 - N)$*

<http://watson.latech.edu/book/algorithms/algorithmsSorting2.html>

n=5

第2回合與1個數比較，比1次，n-4次

第3回合在2個數比較，比2次，n-3次

第4回合在3個數比較，比3次，n-2次

第5回合在4個數比較，比4次，n-1次

$(n-1) + (n-2) + \dots + 1 = n(n-1) / 2$

平均時間複雜度為:  $O(n^2)$

# Insert Sort

- Analysis of time complexity

- the runtime complexity of insertion sort focuses on its nested loops
- the running time of the nested loop is a summation from 1 to  $n-1$

- $T(n) = \frac{n(n+1)}{2} - n = O(n^2)$

# Insert Sort

- <https://www.youtube.com/watch?v=c4BRHC7kTaQ>

## Insertion Sort

```
for i = 1 to n - 1
```

```
    j = i
```

```
    while j > 0 and A[j] < A[j - 1]
```

```
        swap(A[j], A[j - 1])
```

```
        j = j - 1
```

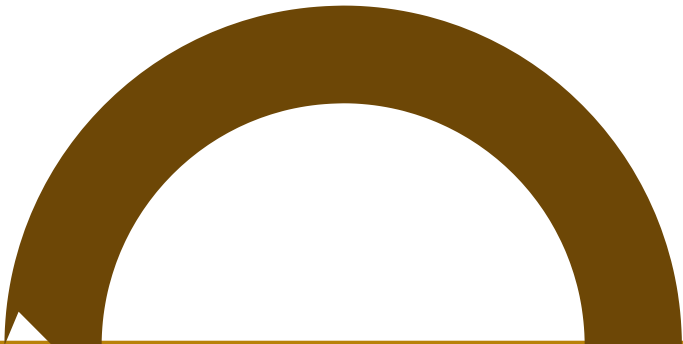
# Quicksort

---



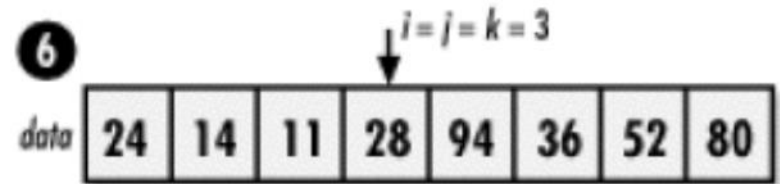
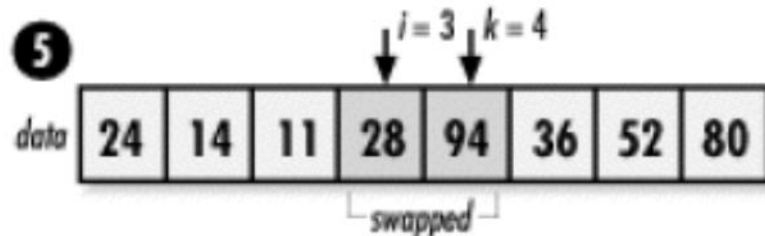
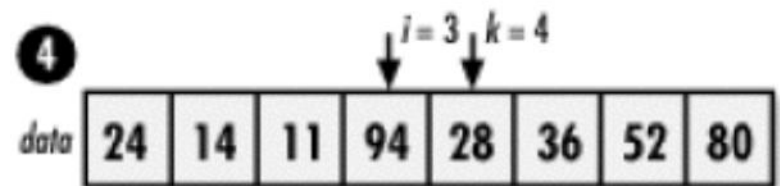
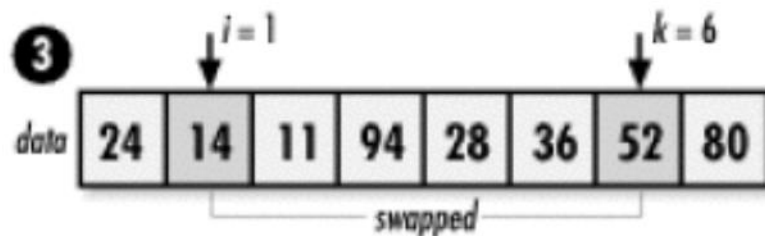
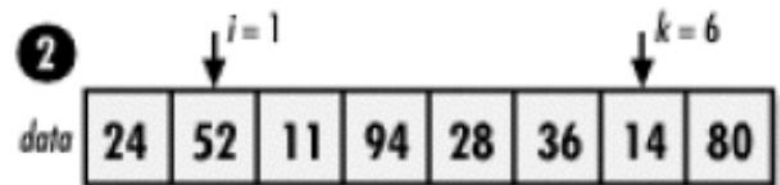
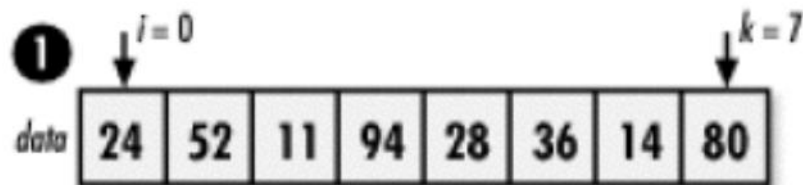
# Quicksort

- Three main steps
  - Divide: partition the data into two partitions around a partition value (pivot value)
  - Conquer: sort the two partitions by recursively applying quicksort to them
  - Combine: do nothing since the partitions are sorted after the previous step



# Quicksort example

- partition around 28



- 快速排序作法：
- 選定一個基準值(Pivot)
- 將比基準值(Pivot)小的數值移到基準值左邊，形成左子串列
- 將比基準值(Pivot)大的數值移到基準值右邊，形成右子串列
- 分別對左子串列、右子串列作上述三個步驟 ⇒ 遞迴(Recursive)
- 直到左子串列或右子串列只剩一個數值或沒有數值



<http://notepad.yehyeh.net/Content/Algorithm/Sort/Quick/Quick.php>



*Step 1: Identify the pivot (e.g. mid value)*



*Step 2: Initialise your left and right pointers*



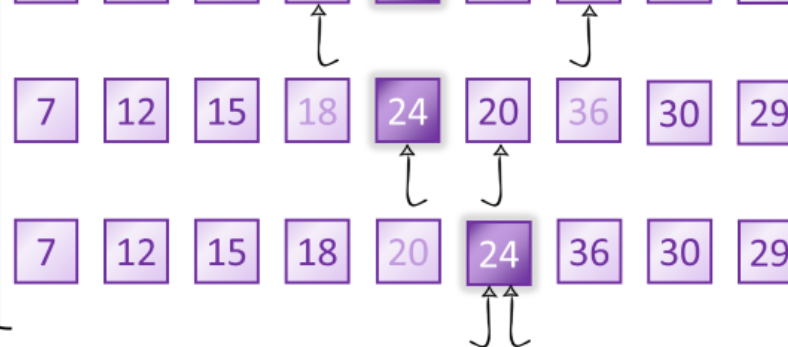
*Step 3: Slide your pointers until you find two values to swap:  
(e.g. 39 > pivot and 12 < pivot)*



*Step 4: Swap the two values*



*Step 5: Repeat  
steps 3 & 4 until  
your left and  
right pointers are  
both pointing  
towards the pivot*



*At this stage, the pivot is now in its final position. All values on the left of the pivot are lower than the pivot, all values on the right are greater than the pivot.*

*We can now repeat all the above steps (1 to 5) recursively with both sub-lists (on the left and on the right of the pivot).*

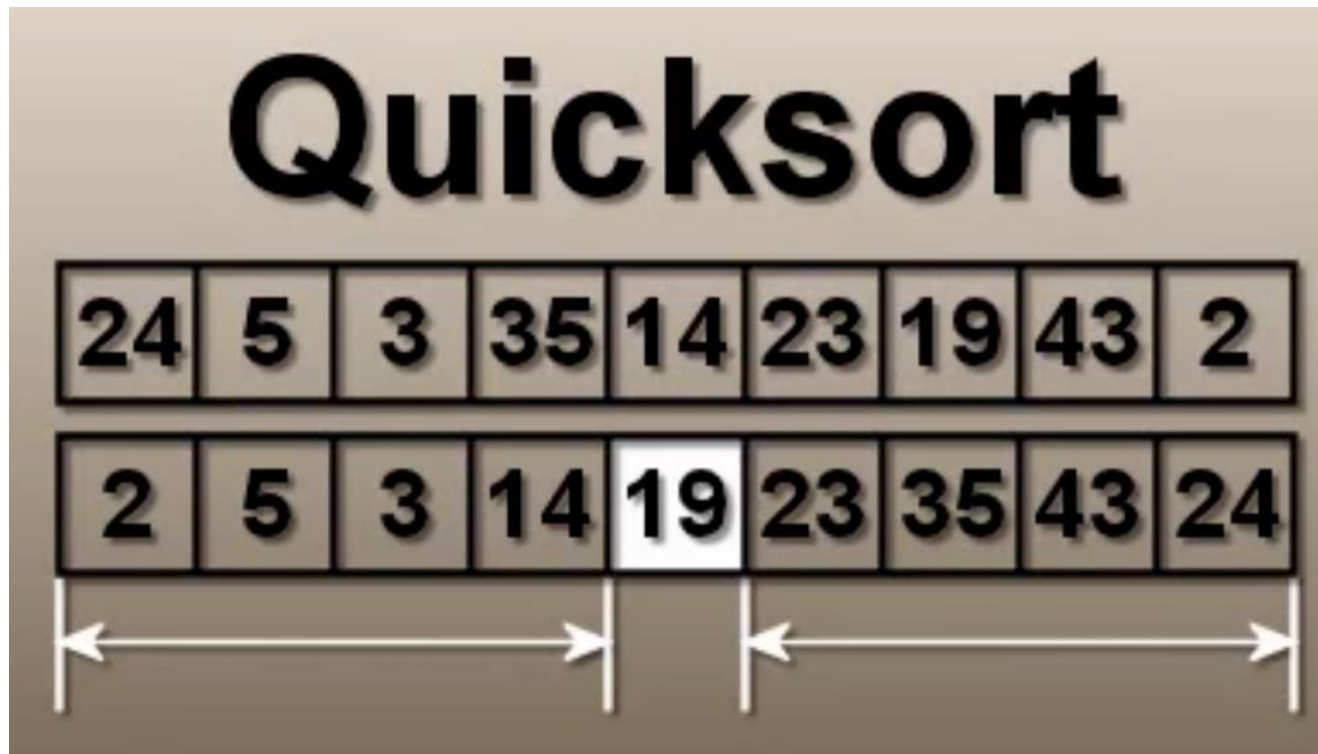
# Quicksort



- Analysis of time complexity
  - worst case:  $O(n^2)$
  - average-case running time:  $O(n \log n)$

## Quicksort Demo

- [https://www.youtube.com/watch?v=y\\_G9BkAm6B8](https://www.youtube.com/watch?v=y_G9BkAm6B8)



# Merge sort

---

# Merge Sort

6 5 3 1 8 7 2 4

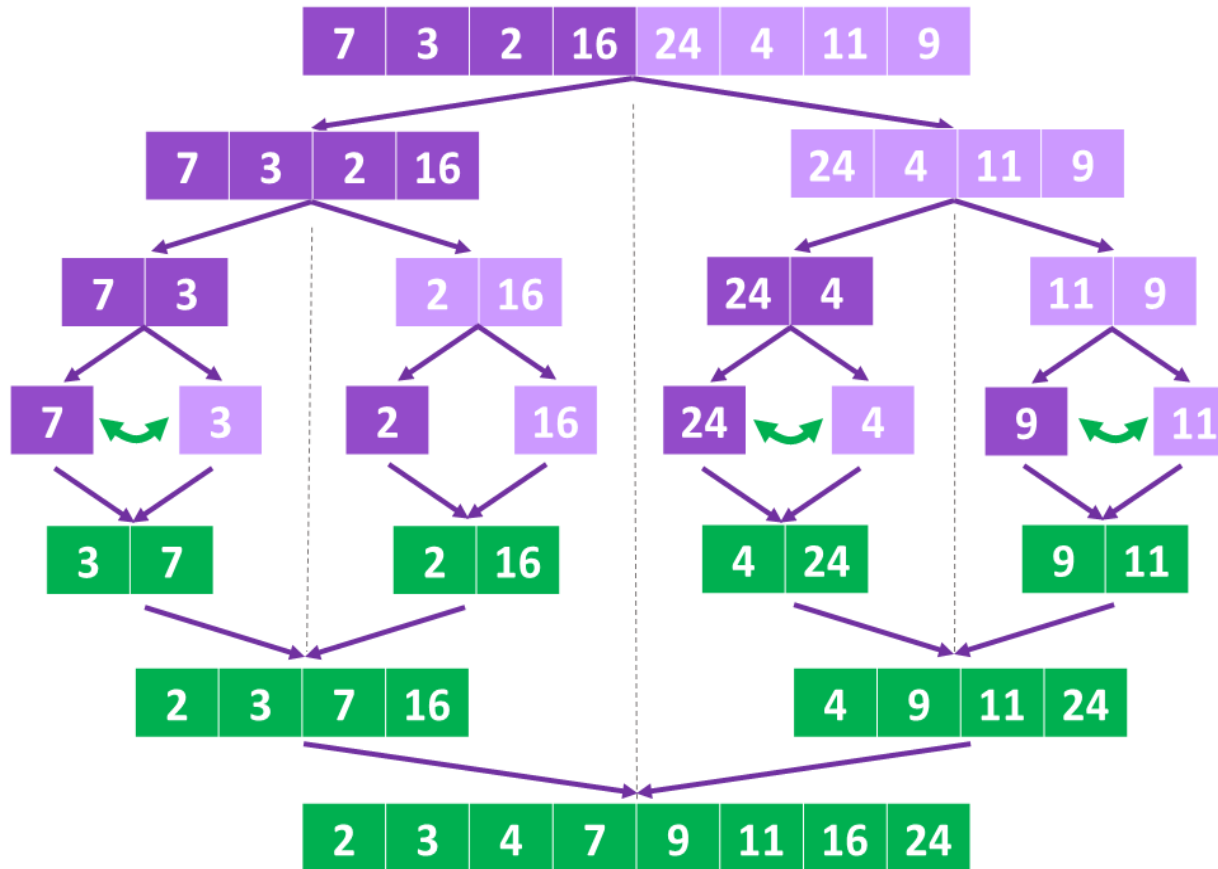


# Merge Sort

- Three main steps
  - Divide: we divide the data in half
  - Conquer: we sort the two divisions by recursively applying merge sort to them
  - Combine: we merge the two divisions into a single sorted set

# Merge Sort

## Merge Sort



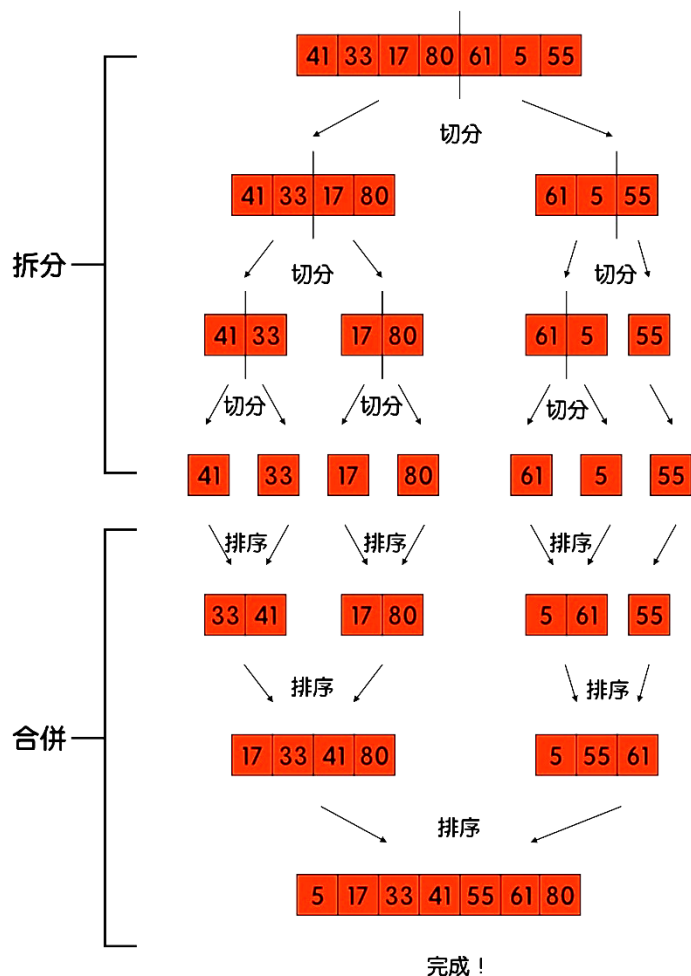
Step 1:  
Split sub-lists in  
two until you  
reach pair of  
values.

Step 3:  
Sort/swap pair  
of values if  
needed.

Step 4:  
Merge and sort  
sub-lists and  
repeat process  
till you merge to  
the full list.

<https://www.101computing.net/merge-sort-algorithm/>

<http://watson.latech.edu/book/algorithms/algorithmsSorting3.html>

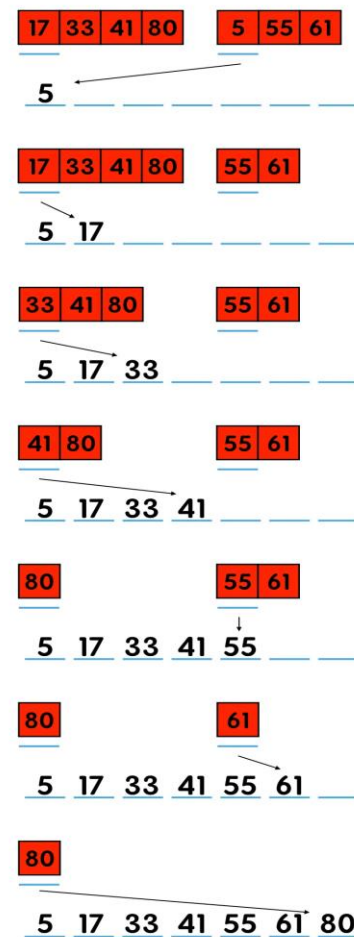


把一個  $n$  的陣列切切切，切到每個小陣列都只有 1 個數字時，需要  $n-1$  個步驟（切  $n-1$  刀）。

需要進行幾回合的合併呢？從一開始 7 個小陣列合併成 4 個，再從 4 個合併成 2 個，最後合併成 1 個，每一回合的合併，都可以讓下一回合需要合併的陣列減少一半，這樣的特性表示總共需要合併的回合數會是以 2 為底的  $\log n$  次。

總共需要回合數： $O(\log n)$   
每回合的合併需要花： $O(n)$

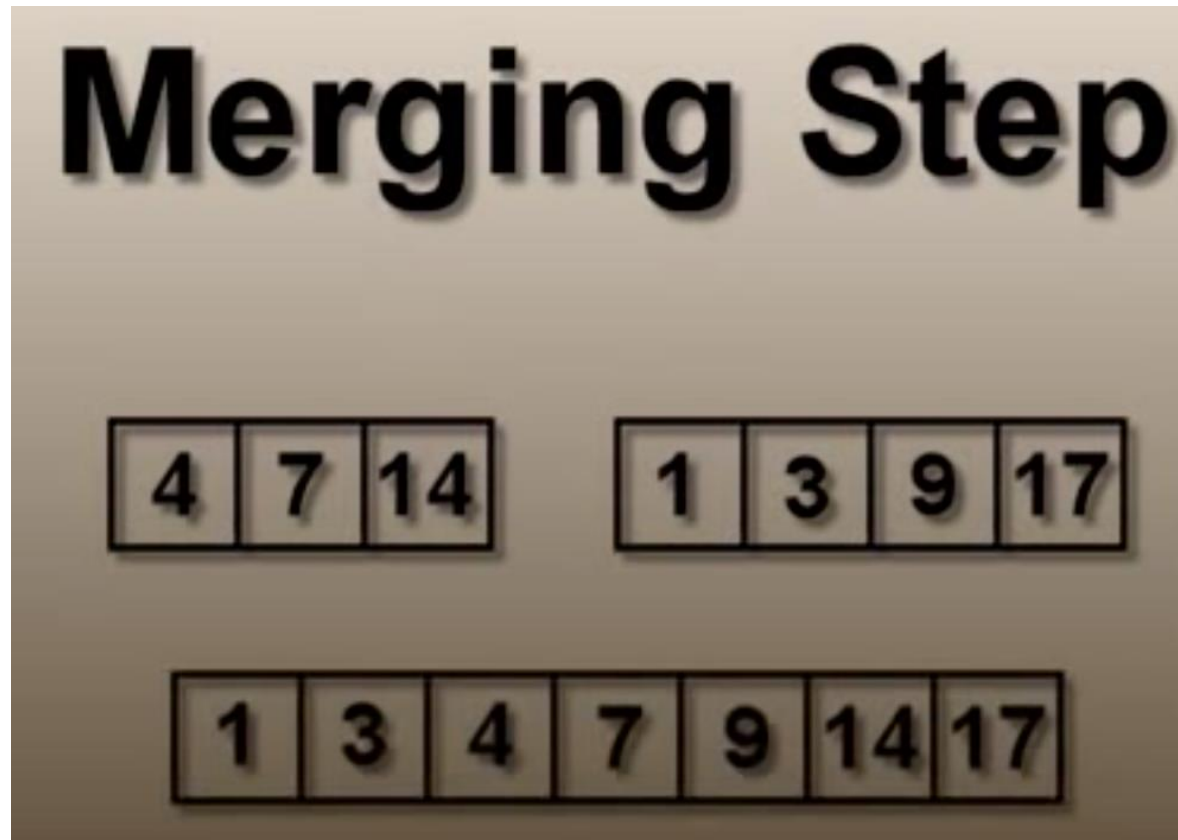
把兩個排序好的小陣列合併成一個排序好的大陣列需要多少步驟呢？如果兩個小陣列加起來有  $n$  個數字，總共就需要  $n$  個步驟。每次都只需要比較兩個陣列的第一個數字，把比較小的依序丟到新的陣列中，再重新比較兩個小陣列的第一個數字。



拆分的步驟數為  $n-1$ ，合併的步驟數為  $n$  乘上  $\log n$ ，相加後我們可以得知合併排序法總共的步驟數為  $n-1 + n \log n$ 。化成大  $O$  符號時，我們只計最高項係數並省略常數，得到合併排序法的時間複雜度，即為  $O(n \log n)$ 。

# Merge Sort Demo

- <https://www.youtube.com/watch?v=GCae1WNvnZM>



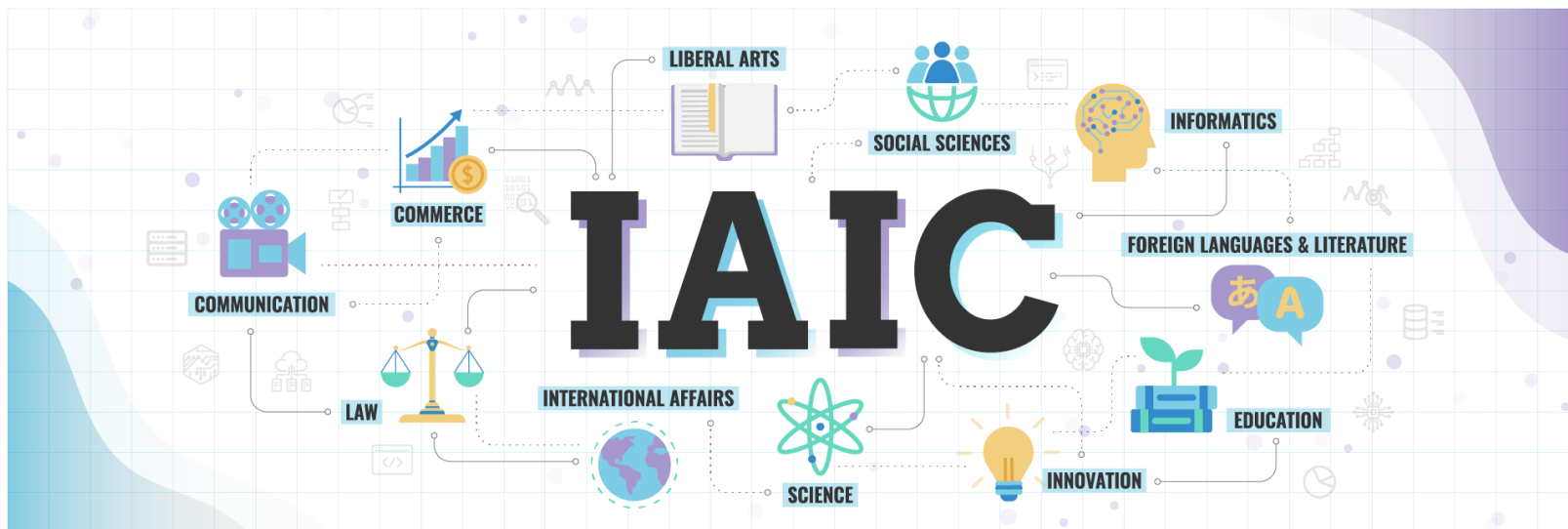
# Sorting Algorithms

- [https://www.youtube.com/watch?v=INHF\\_5RIxTE](https://www.youtube.com/watch?v=INHF_5RIxTE)



Sorting Algorithms	Time Complexity		
	Best Case	Average Case	Worst Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

<https://www.enjoyalgorithms.com/blog/comparison-of-sorting-algorithms>

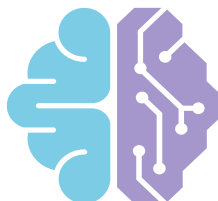


研究合作 跨域教學 多元服務



政大

NATIONAL CHENGCHI UNIVERSITY



人工智慧  
跨域研究中心

Interdisciplinary Artificial  
Intelligence Center

Dr. Chih-Hsun Wu

吳致勳 助理教授

20031214@nccu.edu.tyw

j20031214@gmail.com