投稿類別：資訊類

篇名：

排序演算法之程式實作與計算效能分析

作者：

蔣承佑。國立屏東高中。二年級十七班

指導老師：

孔志明老師

壹、前言

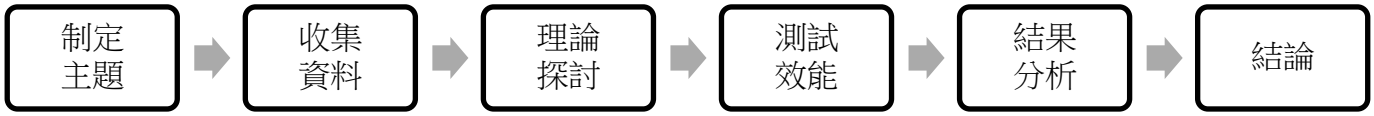
一、研究動機

在現今資訊爆炸的時代，生活中總是充斥著各式各樣的資料需要被排序。學界認為當電腦的相關領域剛開始起步發展之時，排序的應用就已成為不可或缺的存在，並經由時間的推演，五花八門的排序演算法也逐漸地出現（陳南熹，2012）。對於這些不同的排序演算法，理論、效率、過程等性質都各有優缺點與不同之處，因此，將常見的排序演算法做分析與整理即為本研究的動機。

二、研究目的

藉由常見的排序演算法，如選擇排序、氣泡排序、插入排序、謝爾排序、快速排序，比較其理論、時間複雜度、執行時間的相同與相異之處，再使用 Python 撰寫排序演算法之程式測試執行時間並與理論對照，即為本研究之目的。

三、研究架構



圖一：研究架構圖。
（資料來源：筆者自繪）

貳、文獻探討

一、排序

排序的英文是 sort，日文是ソート。排序的目標是給定一筆數據，並讓數能夠照順序排列（石田保輝、宮崎修一，2017）。網頁 Sort-演算法筆記（2023）指出：「**「排序」。把一群數字由小到大排好。**」筆者認為此論點即為排序的精髓，可說明最基本的排序概念。

基本上等價於數字或數組的東西均可排序，例如字元、字串、指標、鍵值、數組等（Sort-演算法筆記，2023）。如果資料的數量很少，以人工方式排序仍然是可行的，但當數據的數量過多，就變得相當困難；因此，排序的效率成為非常重要的一點，而衍生出許多各式各樣的排序演算法（石田保輝、宮崎修一，2017）。

二、排序法分類

（一）排序演算法之原理

排序演算法的原理主要分為兩種，其一為比較式排序演算法（Comparison-based Sorting Algorithm），核心理論為對調式排序，將小的數字往前排，大的數字則往後放；其二為非比較式排序演算法（Non-comparison-based Sorting Algorithm），核心理論為放置式排序，雖然需要額外的記憶體來放置，但通常時間比對調式排序來得快（Sort-演算法筆記，2023）。

（二）穩定排序與不穩定排序

假設資料中有兩筆相同的值，如表一之原始資料中有兩個值為 3 的元素，為方便區分，筆者將右邊的 3 加上「*」符號。若資料在排序之後，其鍵仍然保持原來的順序，即 3* 在 3 右邊，稱為穩定排序（Stable Sort）；若鍵有可能不保持原本的順序，即 3* 有可能被排在 3 的左邊，即稱為不穩定排序（Unstable Sort），但不穩定排序仍可在設計流程時被實作為穩定排序（吳燦銘，2020）。

表一、穩定排序與不穩定排序之結果

原始資料	2	3	4	3*	1
穩定的排序	1	2	3	3*	4
不穩定的排序	1	2	3*	3	4

（資料來源：筆者自繪）

（三）內部排序與外部排序

內部排序（Internal Sort）的排序操作在主記憶體中完成，通常適用在資料比較少的情況，一般的排序法均屬於此類；外部排序（External Sort）的排序操作則是在輔助記憶體中完成，由於資料量較大，因此需要外部記憶體輔助傳輸排序資料（Aquamay，2022）、（排序(sorting)，2023）。

三、複雜度

（一）時間複雜度

時間複雜度（Time Complexity）的概念為電腦花了多少時間成本在執行演算法上，由於每臺電腦之計算效能並不相同，因此其並非以精確的時間來測量，而是計算執行幾個指令來計算，其結果以 Big-O 表示。（Jason Chen，2019）

（二）空間複雜度

空間複雜度（Space Complexity）的概念為在執行演算法所花費的記憶體成本，而時間複雜度和空間複雜度可以互相搭配以提高執行效能，例如利用記憶體空間記錄重複計算的部份而提升速度，其結果也是以 Big-O 來表示。（Jason Chen，2019）

（三）Big-O

學者何東穎（2019）指出：「**Big-O 是用來表示一個演算法輸入資料大小與執行時間變化的漸進符號（Asymptotic Notation）。**」漸進符號，基本上是用函數的概念說明整個複雜度的趨勢，而 Big-O 是指演算法上限的意思，因此成為最常被討論複雜度漸進符號之一，而理論上執行時間由「常數時間」到「階乘時間」漸增（Chiu CC，2016）。常見的 Big-O 表示方式如表二。

表二、Big-O 依據不同時間複雜度之表示方式

Constant time	常數時間	$O(1)$
Logarithmic time	對數時間	$O(log\ n)$
Linear time	線性時間	$O(n)$
Quasilinear time	線性乘對數時間	$O(n\ log\ n)$
Quadratic time	次方時間	$O(n^2)$
Exponential time	指數時間	$O(2^n)$
Factorial time	階乘時間	$O(n!)$

（資料來源：[演算法]Big O and Time Complexity。2019 年 10 月 21 日。取自 https://reurl.cc/qLWNQn）

四、排序法之應用

在生活中，排序法的應用非常廣泛，能讓原本繁複的事情變的更容易。舉例來說，藉由部首筆劃多寡和注音順序以排序漢字的字典，還有將分數排序並找出成績的優劣，以及依照身高排出高矮順序等等，都是排序在日常中常見的應用（謝孫源、李佳衛、洪綾珠，2016）。而隨著時間的推演與科技的進步，排序法應用在各式各樣的領域中，其發展的技術也愈趨成熟（陳南熹，2012）。

參、研究方法

一、問題發現與觀察

排序演算法的種類五花八門，各式各樣的排序法除了理論的基礎不同，執行的效能也各有所異。學者蘇保亦（2018）於其論文中所探討的常見的排序演算法之中，對於不同排序演算法的效率差異，筆者將分析其理論與效能，並設計程式以實測時間。

二、資料搜集與整理

在資料搜集與整理的部份，筆者將會實作排序演算法之程式，並多次實測執行時間以取平均值。對於由測試獲得的原始數據，除了轉換為較具分析性的表格之外，還有做測量時間結果之比較。

三、歸納與統整

經由不同排序演算法所耗費的時間比較，歸納出排序法的特性、理論、執行模式、效能等。排序演算法的效率優劣，通常會影響到整個程式的速度，因此，統整常見的排序法，並做出分析，能在使用排序時有不同的選擇，選取最適合之方案，以提升程式的效率與性能。

肆、研究分析與結果

對於本研究之分析與結果，筆者將透過以下兩個部份來進行說明。首先是排序演算法的理論說明，在這個部份將會探討排序演算法的邏輯思維與操作方式；再者是程式之運算效能測試，藉由實作程式並使用多組數據執行，以提供排序法效率差異的直接證據。

一、排序演算法理論說明

（一） 氣泡排序法

氣泡排序法是在數列中藉由比較相鄰兩元素大小以及適當的交換之操作，使數列的最大值被排到陣列的最後面，彷彿氣泡浮出水面般。其步驟為：

1、 比較第一個與第二個數的大小。
2、 若第一個大於第二個數，則把此二數交換。
3、 接著比較第二個與第三個數、第三個與第四個數……，若不符合順序則交換，並持續到最後一個元素。

此時最大的數被放到陣列的最後面。接著對未排序完成數列重複進行以上步驟直到排序完成。其比較次數與排列順序無關，均為$\sum_{i=1}^{n-1}(i) \approx (n^2 - n)/2$ 次，因此其時間複雜度為$O(n^2)$（石田保輝、宮崎修一，2017）、（Bubble Sort，2023）。

（二） 選擇排序法

選擇排序法是在數列中，搜尋最小值並放到最左邊的操作。其步驟為：

1、 使用線性搜尋找出數列中的最小值。
2、 與未排序完成數列中最左邊的值交換。

此時最小的數被放到最左邊。接著對未排序完成數列重複進行以上步驟直到排序完成。由於選擇排序法使用線性搜尋，比較總次數為$\sum_{i=1}^{n-1}(i) \approx (n^2 - n)/2$ 次，因此其時間複雜度

為$O(n^2)$（石田保輝、宮崎修一，2017）、（Selection Sort Algorithm，2023）。

（三） 插入排序法

插入排序法是從數列中，在陣列未排序的部份取出一數，並放到已排序的部份中合適的位置。其步驟為：

1、 先假設第一個數已經排序完成。
2、 從未排序的部份取出最左邊的數，並從已排序的部份開始比較，若左邊的數較大，則兩者對調，並持續到出現比本身小的數或已經沒有數可以比。

接著對所有未排序部份的數進行第二步驟，直到全部排序完成。其最佳情形為升序數列，完全不需對調；最壞情形為降序數列，每次都必須與前數對調；因此其平均時間複雜度為$O(n^2)$（石田保輝、宮崎修一，2017）、（Insertion Sort Algorithm，2023）。

（四） 快速排序法

快速排序法是分治法的應用之一，其概念為將原本的大問題分成數個小問題，分而治之。其排序的方法為從數列中隨機選擇一個數做為基準值，把數列中的數與基準值比較，分成小於基準值的數與大於基準值的數，並遞迴此兩群數，以排序整個數列。其步驟為：

1、 選一個數做為基準值。
2、 重複由左至右找出比基準大的值，並設其鍵值為i；同時由右至左找出比基準小的值，並設其鍵值為j。
3、 如果$i < j$，此兩數交換；若找到的數之鍵值$i = j$，則把此數與基準值交換。
4、 此時在基準值左邊的數均小於基準值，基準值右邊的數均大於基準值，並對此兩群數分別進行以上步驟。

當整個數列都執行完上述之步驟，即表示排序完成。其最壞情形為當基準值一直為數列的最大值或最小值，此時每次需要再排序的數列全部都集中在一邊；最佳情形為當基準值為數列的中間值；因此其平均時間複雜度為$O(n\log n)$（石田保輝、宮崎修一，2017）、（Insertion Sort Algorithm，2023）。

二、程式運算效能測試

對於測試排序法的花費時間，筆者使用 time 模組的 time 函式分別記錄排序的開始和結束時間，並將此兩數相減，即可得執行時間；待排序的數列，使用 random 模組的 randint 函式隨機產生，並讓同一數列給以下幾種排序法測試。本研究使用的設備如表三所示。

表三、軟體與硬體設備

型號	Vivobook_ASUSLaptop
處理器	12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz
RAM	8.00 GB
系統	64 位元作業系統、Windows 11 家用版
軟體	Visual Studio Code 1.81.1

（資料來源：筆者自繪）

（一） 產生原始測試數列資料

```
import time      # time 模組
import random # random 模組
n = int(input())
original_data = [random.randint(0, 10**n) for i in range(10**n)]      # 產生數列
```

（二） 氣泡排序法之副程式

```
def bubble(array):
    for i in range(len(array)-1, 0, -1):
        for j in range(i):
            if array[j] > array[j+1]:# 若後面的數大於前面
                array[j], array[j+1] = array[j+1], array[j]  # 兩數交換
```

（三） 選擇排序法之副程式

```
def selection(array):
    for i in range(len(array)):
        min_index = i  # 未排序部份最左邊的索引值
        for j in range(i+1, len(array)):
            if array[j] < array[min_index]:
                min_index = j         # 藉由比較找出最小的數之索引值
        array[i], array[min_index] = array[min_index], array[i]
```

（四） 插入排序法之副程式

```
def insertion(array):
    for i in range(1, len(array)): # 假設第一個數已排序完成
        temp = array[i]
        j = i - 1
        while j >= 0 and temp < array[j]:     # 在已排序完成的部份尋找合適的位置
```

```
            array[j+1] = array[j]
            j -= 1
        array[j+1] = temp
```

（五） 快速排序法之副程式

```
def quick(array, left_index=0, right_index=len(original_data)-1):
    if left_index < right_index:
        i = left_index
        j = right_index
        pivot = array[left_index]  # 基準值
        while i != j:
            while array[j] > pivot and i < j:    # 由右至左找比基準小的值
                j -= 1
            while array[i] <= pivot and i < j:  # 由左至右找比基準大的值
                i += 1
            if i < j:
                array[i], array[j] = array[j], array[i]  # 兩數交換
        array[left_index], array[i] = array[i], array[left_index]
        quick(array, left_index, i-1)     # 對基準值的左右兩邊進行遞迴，繼續進行排序
        quick(array, i+1, right_index)
```

（六） 效能測試

```
sorting_algorithms = [bubble, selection, insertion, quick]  # 排序法陣列
for algorithms in sorting_algorithms:# 變數 algorithms 迭代排序法陣列
    data = original_data.copy()  # 原始數列複本
    t1 = time.time()    # 記錄開始時間
    algorithms(data)   # 執行排序法
    t2 = time.time()    # 記錄結束時間
    print(t2-t1)   # 印出執行時間
```

三、結果說明

以上的程式執行氣泡排序、選擇排序、插入排序和快速排序這幾種不同的排序演算法，多次測試它們的實際執行時間，並取平均值。

（一） 資料大小：1000 筆數值

數列大小為 1000 時，平均執行時間，氣泡排序約 63.19 毫秒；選擇排序約 27.28 毫秒；插入排序約 29.88 毫秒；快速排序約 1.58 毫秒。快速排序法花費時間最短，約比泡沫排序快 40 倍。

表四：數列大小為 1000 之執行結果（單位：秒）

	氣泡排序法	選擇排序法	插入排序法	快速排序法
第 1 次測試	0.061537266	0.026102304	0.0300951	0.001900911
第 2 次測試	0.060322285	0.02710104	0.0294559	0.000999451
第 3 次測試	0.069106102	0.028998613	0.032008648	0.001996517
第 4 次測試	0.05999279	0.028192282	0.030827761	0.001998901
第 5 次測試	0.064995289	0.026009083	0.026991129	0.001000643
平均	0.063190746	0.027280664	0.029875708	0.001579285

（資料來源：筆者測試整理）

（二） 資料大小：10000 筆數值

數列大小為 10000 時，平均執行時間，氣泡排序約 7.67 秒；選擇排序約 3.15 秒；插入排序約 3.61 秒；快速排序約 0.02 秒。快速排序法花費時間最短，約比泡沫排序快 291 倍。

表五：數列大小為 10000 之執行結果（單位：秒）

	氣泡排序法	選擇排序法	插入排序法	快速排序法
第 1 次測試	6.236581087	2.503978014	2.794728518	0.022169113
第 2 次測試	6.227401495	2.473868847	2.854576111	0.031290293
第 3 次測試	7.568361282	4.086693764	4.826337337	0.031253338
第 4 次測試	7.712268353	3.078131437	4.766360521	0.031250715
第 5 次測試	10.61263299	3.593745947	2.828178883	0.015627861
平均	7.671449041	3.147283602	3.614036274	0.026318264

（資料來源：筆者測試整理）

（三） 資料大小：100000 筆數值

數列大小為 100000 時，平均執行時間，氣泡排序約 10 分 56 秒；選擇排序約 4 分 36 秒；插入排序約 5 分 20 秒；快速排序約 0.23 秒。快速排序法花費時間最短，約比泡沫排序快 2903 倍。

表六：數列大小為 100000 之執行結果（單位：秒）

	氣泡排序法	選擇排序法	插入排序法	快速排序法
第 1 次測試	656.3725526	275.3669662	317.3243511	0.219521761
第 2 次測試	656.625272	276.2683511	325.4089167	0.235584497
第 3 次測試	656.4525099	279.5072694	317.3629465	0.235527277
第 4 次測試	654.6145051	274.1097994	315.4391143	0.218748093
第 5 次測試	656.2028594	276.7331388	324.9308145	0.220385075
平均	656.0535398	276.397105	320.0932286	0.225953341

（資料來源：筆者測試整理）

伍、研究結論與建議

（一） 研究結論

依照學理之時間複雜度的觀點，由於快速排序法的時間複雜度為$O(n\log n)$，優於其他三者，因此效率應為最佳，其餘三者之時間複雜度均為$O(n^2)$，執行效率較差。經實作驗證，測試執行效能優劣為：快速排序法 > 選擇排序法 > 插入排序法 > 氣泡排序法，且資料量越大時，氣泡排序、選擇排序與插入排序的效能逐漸下降，同時也顯露出快速排序優異的效率。當資料為 1000 筆時，快速排序法執行速度為泡沫排序的 40 倍；當資料為 10000 筆時，快速排序法執行速度為泡沫排序 291 倍；當資料為 100000 筆時，快速排序法執行速度為泡沫排序的 2903 倍。

形成上述結果的原因，在氣泡排序的過程中，每一次都進行相鄰元素的比較和交換的操作，即使在已經有序的情況下也會進行大量的比較，而耗費較多的時間。在選擇排序的過程中，雖然為找出最小值而進行多次比較，但每回合只需進行一次交換；在插入排序的過程中，取出一數，比較大小並移出空間，再放入適當位置。而效能最好的快速排序過程中，每回合將原數列分成兩個子數列執行遞迴，讓每個子數列只有一個數以完成排序。因為氣泡排序的操作次數遠高於其餘三者，而快速排序操作的次數最少，所以造成了效能的優劣之分。

（二） 研究建議

首先，當筆者測試 1000000 筆資料的數據時，程式執行了大約三小時後產生當機的情形，本研究在設備的因素下，無法得出更大筆數據的執行時間，若能提升設備或探討其他解決方法，則能更深入研究演算法的效能。而本研究提供常見排序演算法效能的實際證據，在使用排序法時，除了考慮資料的規模之外，也要依據不同的情況，選擇適合的排序法。

陸、參考文獻

（一） 書籍

1、 石田保輝、宮崎修一（2017）。アルゴリズム図鑑：絵で見てわかる 26 のアルゴリズム 演算法圖鑑：26 種演算法 + 7 種資料結構，人工智慧、數據分析、邏輯思考的原理和應用全圖解。臺北市。臉譜。
2、 吳燦銘（2020）。AI 世代高中生也能輕鬆搞懂的運算思維與演算法：使用 Python。新北市。博碩文化。

（二） 學術論文

1、 陳南熹（2012）。在 GPU 上增進排序演算法的效能。國立清華大學資訊工程所：碩士論文。
2、 蘇保亦（2018）。快速排序演算法之改良研究。萬能科技大學資訊管理研究所：碩士論文。
3、 何東穎（2019）。以演算法程式設計競賽試題為例使用 Big-O AST 靜態分析函式時間複雜度。國立中央大學資訊工程學系軟體工程碩士班：碩士論文。

（三） 網站資料

1、 Chiu CC（2016）。Complexity：Asymptotic Notation(漸進符號)。2023 年 8 月 25 日。取自 https://reurl.cc/QXzvG0 。
2、 謝孫源、李佳衛、洪綾珠（2016）。電機資工的現況與未來：生活中的演算法。2023 年 8 月 25 日。取自 https://reurl.cc/x7nrAN 。
3、 Jason Chen（2019）。【演算法】排序演算法 Sorting Algorithm。2023 年 8 月 25 日。取自 https://reurl.cc/XE70gD 。
4、 Jason Chen（2019）。【演算法】時間複雜度與空間複雜度 Time & Space Complexity 。2023 年 8 月 25 日。取自 https://reurl.cc/942Kqv 。
5、 Vivian Lo（2019）。[演算法]Big O and Time Complexity。2023 年 8 月 25 日。取自 https://reurl.cc/65r6M6 。
6、 Aquamay（2022）。排序演算法簡介。2023 年 8 月 25 日。取自 https://reurl.cc/Ov9yoR 。
7、 排序(sorting)。2023 年 8 月 25 日。取自 https://reurl.cc/M8roN4 。
8、 演算法筆記。2023 年 8 月 25 日。取自 https://reurl.cc/2LXOmn 。
9、 Bubble Sort。2023 年 8 月 25 日。取自 https://reurl.cc/VLKoRY 。
10、Selection Sort Algorithm。2023 年 8 月 25 日。取自 https://reurl.cc/94M91Y 。
11、Insertion Sort Algorithm。2023 年 8 月 25 日。取自 https://reurl.cc/51eNMq 。
12、Quicksort Algorithm。2023 年 8 月 25 日。取自 https://reurl.cc/DAapmj 。