程式作業一

- **開發環境**

  Visual Studio Code (使用語言： Python)

- **實作方法和流程**

I. 建立 process & thread 方法

(1) process

```
p = multiprocessing.Process( target = method2, args = ( k, split_List, q ) )
p.start()
p.join()
```

先建立一個名字為 p 的 Process，執行 method2，傳入參數 (k, split_List, q)，使用 p.start()來啟動此 Process，再利用 p.join()等待 p 執行結束

(2) thread

```
# 建立 k 個 thread
thread_list = []
for i in range( k ) :
    thread_list.append( threading.Thread( target = BubbleSort, args = ( split_List[i], method, q ) ) )
    thread_list[i].start()

# 等待 thread 結束
for i in range( k ):
    thread_list[i].join()
```

先建立一個存放 threads 的 list。

建立 Thread，執行 BubbleSort，傳入參數(split_List[i],method,q)，thread_list[i].start()用來啟動在 thread_list[i] 的 thread，再利用 thread_list[i].join()等待該 thread 執行結束。

II. 使用的資料結構

(1) List 串列 ： 使用 list 暫時存放讀入資料

(2) Queue 佇列 ： 使用 queue 暫時存放排序資料

III. merge 方法

從 queue 中取前兩個 list (一個為左 list、一個為右 list)，對這兩個 list 中的數值相互比較，當左 list 目前的值 <= 右 list 目前的值，temp_list 會放左 list 目前的值，反之則放右 list 目前的值，當某一個 list 中的數值皆在 temp_list 中，將另一個 list 未放入在 temp_list 的值放入後，會得到一個新的排序結果，最後將這個排序結果存回去到 queue 的尾端。

IV. Process 之間如何共享資料

由於 process 之間相互獨立，因此 parent process 會無法取得 child process 排序後的結果，所以使用 multiprocessing.Manager() 建立一個 Manager 物件，使得不同 process 可以共享資料 (使用 multiprocessing.Manager().Queue()存放 process 執行後的結果)。

```
a = multiprocessing.Manager()
q = a.Queue()
```

V. 執行流程

方法一: 將讀入的資料存在 dataList 中，進行 bubble sort，當排序執行完畢後，將排序結果、CPU 執行時間以及寫檔日期時間寫成一個輸出檔。

方法二: 將讀入的資料存入 dataList 後，將資料分成 k 份，使用 multiprocessing.Manager() 建立 Manager 物件，提供 processes 之間的共享資料，且在此使用 Manager().Queue() 存放 process 執行後的結果。首先先建立一個 process 分別完成 k 份資料各自的氣泡排序，並將排序結果(list)存入 queue 中，再用此 process 將 queue 中的排序 list 兩兩進行 merge，直到 queue 中只剩下一個排序 list，即為最終排序結果。建立完 process 後使用 start() 來啟動 process，以及使用 join() 來等待 process 執行結束，當上述步驟執行完畢後，將排序結果、CPU 執行時間以及寫檔日期時間寫成一個輸出檔。

方法三: 與方法二的差別在於方法二是 single processing，而方法三是 Multiprocessing。在方法三中會建立 k 個 process，各自完成氣泡排序後，將其結果存到 queue 中，再使用 K-1 個 process 將 queue 中的各個排序 list 兩兩進行 merge，直到 queue 中只剩下一個 list 時，該 list 即為最終排序結果。當上述步驟執行完畢後，將排序結果、CPU 執行時間以及寫檔日期時間寫成一個輸出檔。

方法四: 與任務三的差別在於方法三是 Multiprocessing，而方法四是 Multithreading。方法四因為在相同 process 中的 thread 之間是可以共享資料的，應此這裡所建立的 queue 是 queue.Queue() 而非是方法三所使用的 Manager().Queue()。
在方法四中會建立 k 個 thread，各自完成氣泡排序後後，將其結果存到 queue 中，再使用 K-1 個 thread 將 queue 中的各個排序 list 兩兩 merge，直到 queue 中只剩下一個 list，即為最終排序結果，當上述步驟執行完畢後，將排序結果、CPU 執行時間以及寫檔日期時間寫成一個輸出檔。

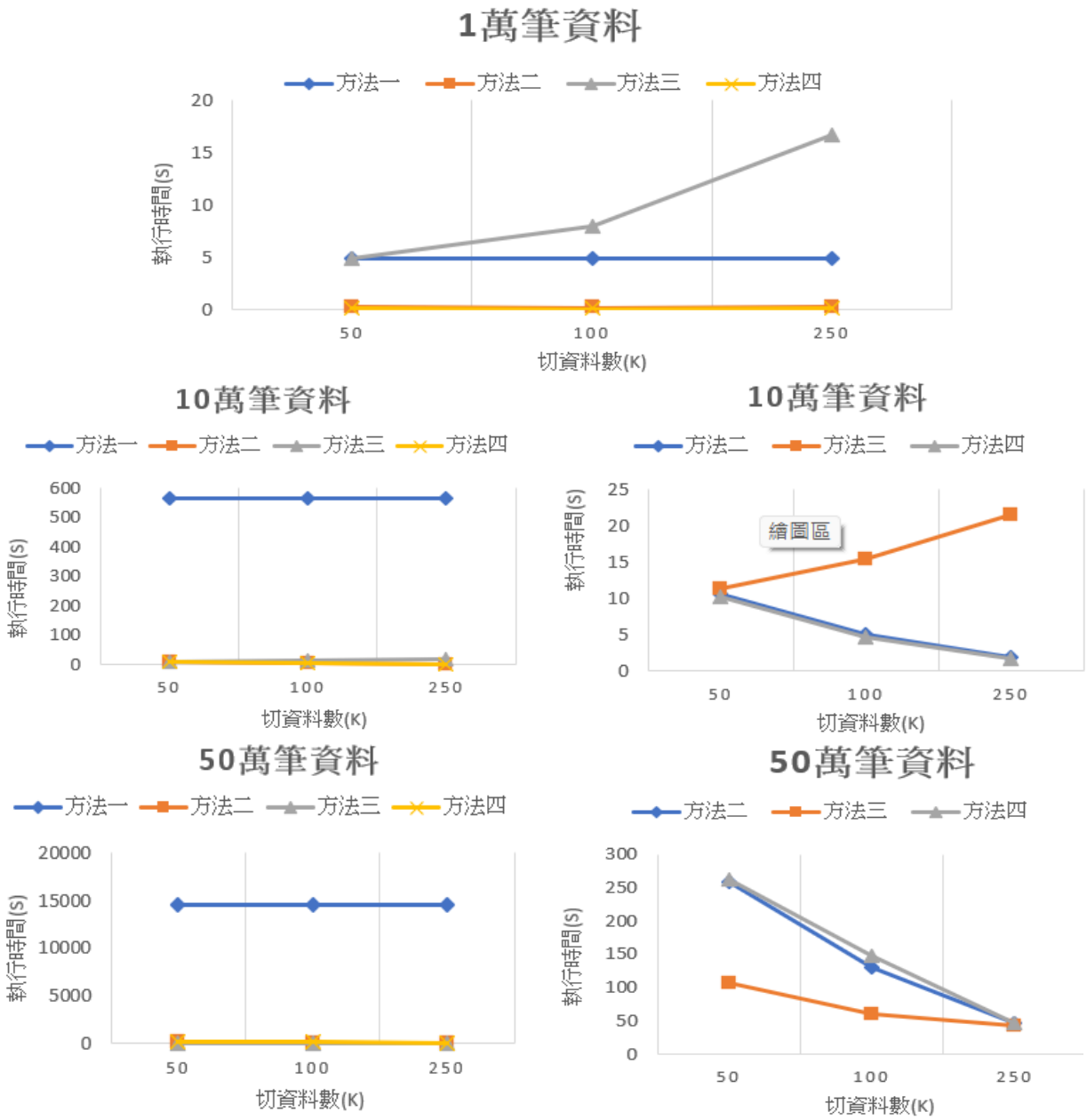
- **探討結果和原因**

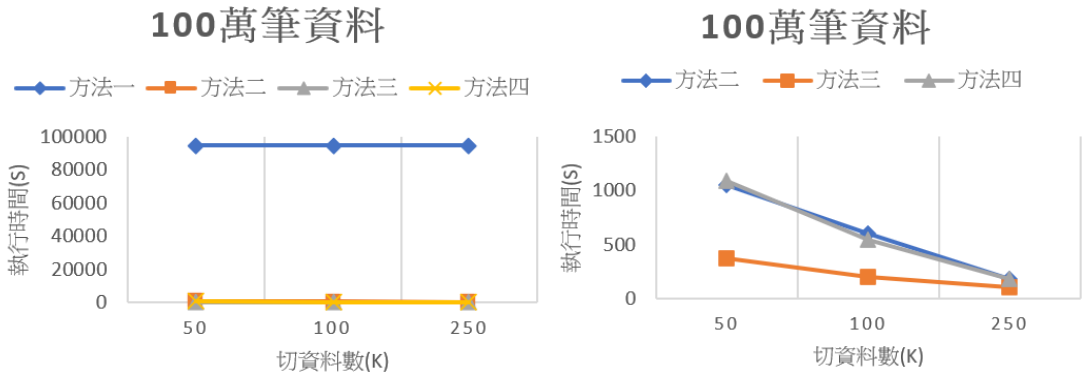
i. 不同 k 值 vs. 執行時間

將資料切成：100, 500 份（K）

K = {100, 500}	N = 1 萬	N = 10 萬	N = 50 萬	N = 100 萬
方法一	4.82482	564.09356	14580.67406	94580.67406
方法二	100：0.20778 500：0.32263	100：5.02655 500：1.36343	100：129.54308 500：26.40974	100：607.05219 500：104.73828
方法三	100：7.9426 500：37.7331	100：15.37759 500：44.39292	100：59.62241 500：90.23559	100：204.98501 500：116.62846
方法四	100：0.08958 500：0.17904	100：4.70228 500：1.1298	100：148.51097 500：30.00702	100：549.32836 500：100.16583

表 1：實驗記錄表格，表格中的時間皆四捨五入到小數點第五位（單位：s）





在上面多張圖中可以得知，當 K 值越大，方法二所耗費的時間越小，可能是因為減少了在氣泡排序中進行排序的串列的資料量，因此花費的時間會縮小。

在上面多張圖中可以得知，當資料筆數相對較小時(10w 內)，K 值越大，方法三耗費的時間越大，可能是因為雖然 Multiprocessing 可以同時運算，但因為當 process 越多，大多執行時間都耗費在作環境交換。當資料筆數相對較大時(50w 以上)，K 值越大，方法三耗費的時間越小，可能是因為 Multiprocessing 可以同時進行運算，可有利的降低執行所需時間。

在上面多張圖中可以得知，當 K 值越大，方法四耗費的時間越小，可能是因為 Multithreading 不但可以達到同時運算，還可以省下複製程式碼與作環境交換的時間，因此時間能夠大幅減少。

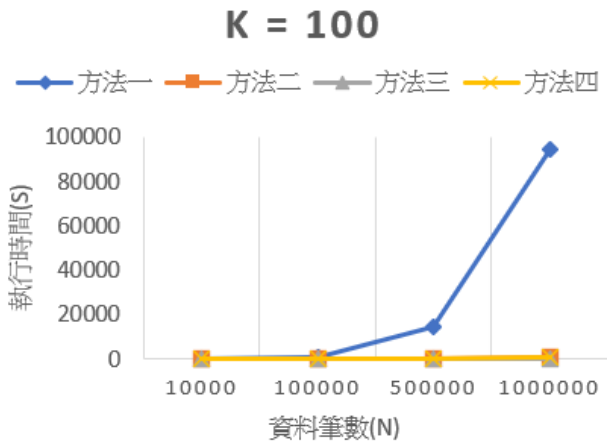
ii. 不同 N 值 vs. 執行時間

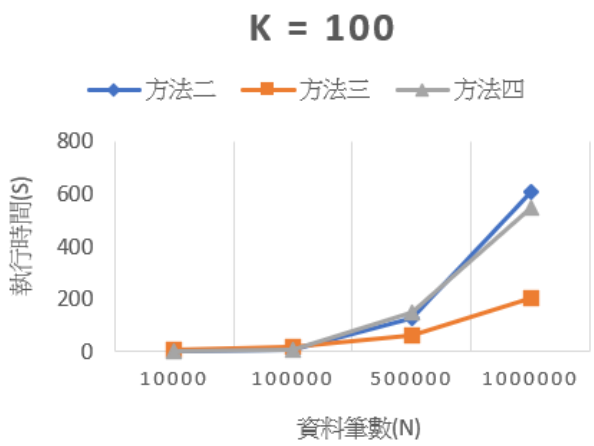
資料筆數：1, 10, 50, 100 萬 (N)

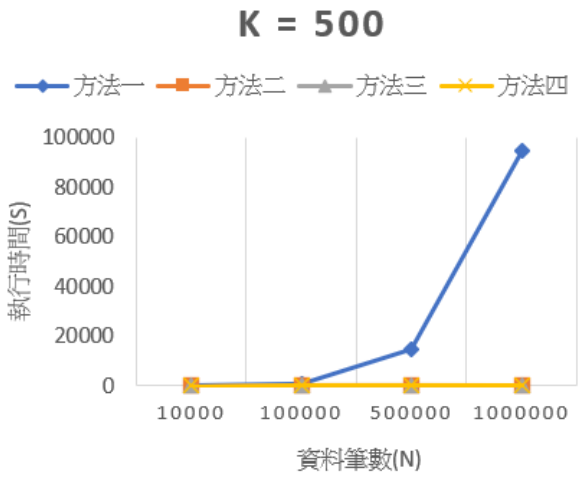
N = {1, 10, 50, 100 萬}	K = 50	K = 100	K = 250
方法一	1w: 4.82482 10w: 564.09356 50w: 14580.67406 100w: 94580.67406	1w: 4.82482 10w: 564.09356 50w: 14580.67406 100w: 94580.67406	1w: 4.82482 10w: 564.09356 50w: 14580.67406 100w: 94580.67406
方法二	1w: 0.27244 10w: 10.6017 50w: 258.79111 100w: 1051.14907	1w: 0.20778 10w: 5.02655 50w: 129.54308 100w: 607.05219	1w: 0.23156 10w: 1.97046 50w: 46.21397 100w: 188.26807
方法三	1w: 4.89021 10w: 11.39797 50w: 106.85215 100w: 373.66493	1w: 7.9426 10w: 15.37759 50w: 59.62241 100w: 204.98501	1w: 16.72253 10w: 21.48298 50w: 43.39334 100w: 107.26968
方法四	1w: 0.17236 10w: 10.30638 50w: 261.49707	1w: 0.08958 10w: 4.70228 50w: 148.51097	1w: 0.10735 10w: 1.77219 50w: 46.15964

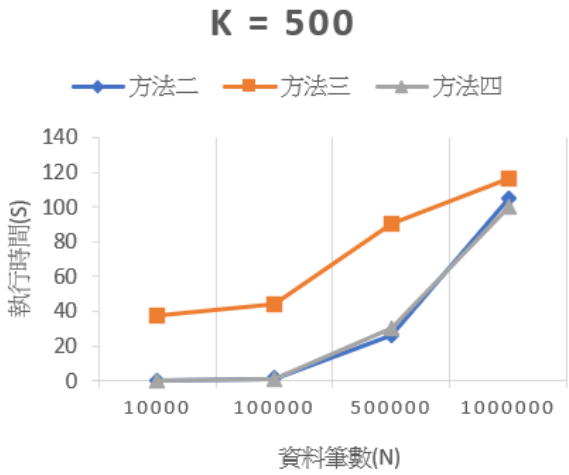
	100w: 1087.14684	100w: 549.32836	100w: 183.87836

表 2：實驗記錄表格，表格中的時間皆四捨五入到小數點第五位（單位：s）









由表 1 與多張圖的結果可得知，方法一單純使用氣泡排序法是最慢的。

由於方法二是 single processing，且我們知道一個 Process 可以有很多個 Thread，因此方法二跟方法四的執行時間才會因此高度相似，差異沒有很大。

當切資料份數不大時(k=100)：

(1.) 當資料數在 10 萬筆以內，使用 Multithreading(方法四)比使用 Multiprocessing(方法三)快，可能是因為 thread 是共用 address space，因此在作環境切換時所花的代價會相對比 process 輕很多。

(2.) 當資料數超過 50 萬筆，使用 Multithreading(方法四)又變得比較慢，可能是因為這些 threads 是依附在 process 內，因此需要平分 process 分配到的時間，因此每個 thread 被分配到的時間就可能不太多，所以每次都只能完成一小部份工作，就會需要花大量的時間進行環境切換。

當切資料份數較大時(k=500)：方法三耗費的時間越大，可能是因為雖然 Multiprocessing 可以同時運算，但因為當 process 越多，需要耗費在作環境交換的時間也會隨之增加。