OS Project1 書面報告

資訊三甲 11027164 趙怡儒

一、 開發環境 ：Visual Studio Code 使用 Python

二、 實作方法和流程 ：

方法一：將 input 存成 list，直接 BubbleSort，並輸出結果以及 cpu time 及日期時間。

方法二：將 input 存成 list 後，將資料分成 k 份，將每一份先自己 BubbleSort 後，再兩個兩個 MergeSort，最後兩個 merge 完即排列完成，輸出結果以及 cpu time 及日期時間。

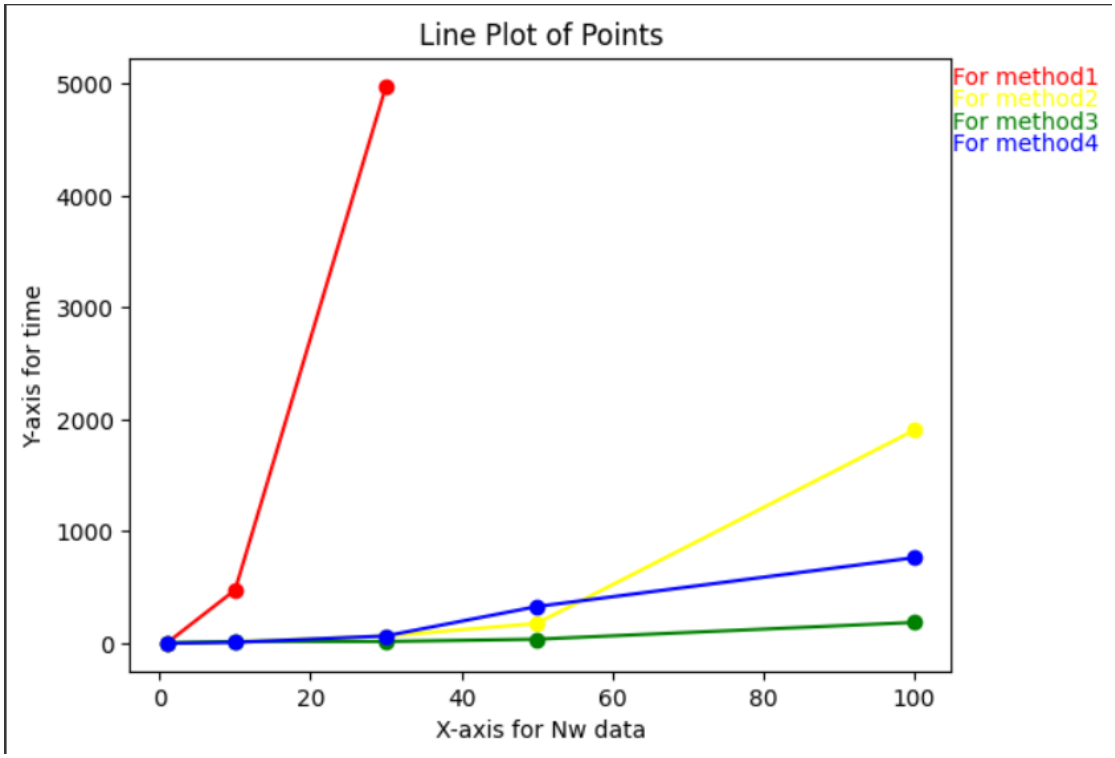
方法三：將 input 存成 list 後，將資料分成 k 份，由 K 個 processes 各別進行 BubbleSort，再用 K-1 個 process(es)作 MergeSort，排列完成後輸出結果以及 cpu time 及日期時間。

方法四：與任務三流程相同，不同的地方在於方法四是由 K 個 threads 各別進行 BubbleSort，再用 K-1 個 thread(s)作 MergeSort，排列完成後輸出結果以及 cpu time 及日期時間。

在執行方法三時，一開始發現執行速度偏慢，在想應該是 process 的資料要傳輸的話需要另外消耗時間，找了網路上的方法，找到在 Python 的 multiprocessing 模組中，Manager 類提供了一種創建共享對象的方法。當使用 Manager 創建 Queue 時，返回的是一個 Queue，這個 Queue 是在多個 process 之間共享的，可以進行跨 process 通信，且為了讓 function 都可以正常使用，將每個方法主要的資料結構都改成這種類型的 Queue。

三、 探討結果和原因 ：

1.

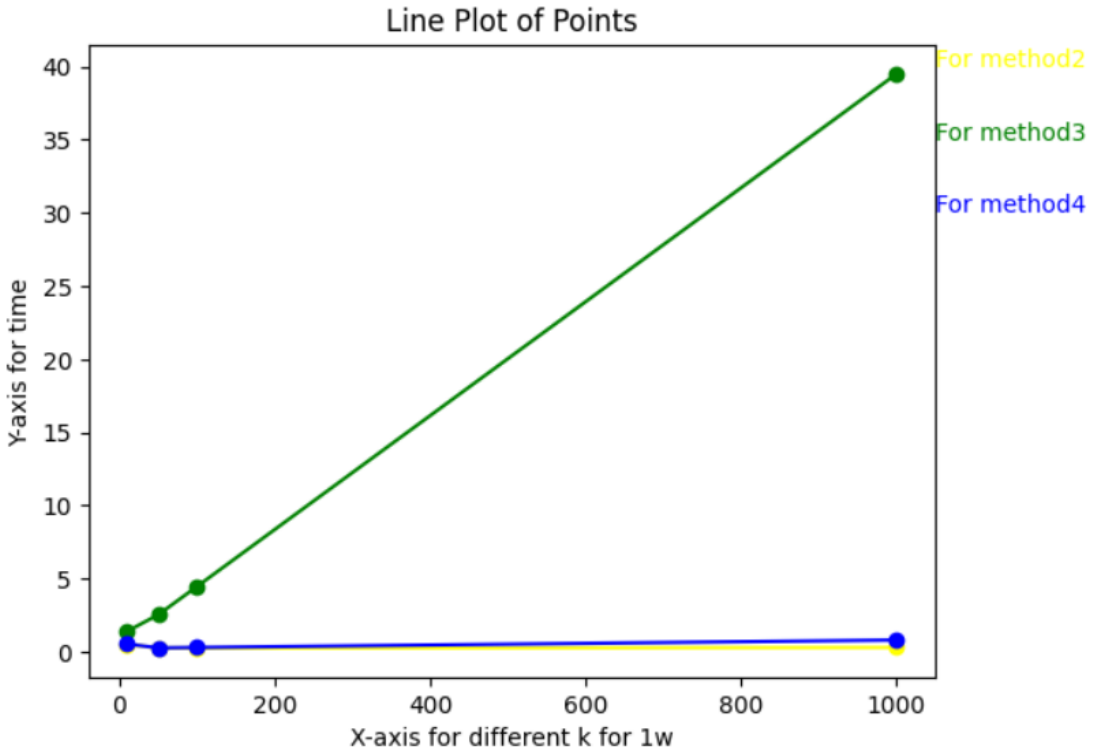


	N=1w	N=10w	N = 30w	N=50w	N=100w
方法一	3.905	475.735	4978.695	X	X
方法二	0.252	12.615	64.503	177.027	1903.266
方法三	2.529	15.469	15.193	35.890	185.940
方法四	0.264	7.309	65.510	327.918	764.266

表一不同 N 值的各個方法執行時間(K 值固定為 50)

由圖與表格可以得知，方法一因為單純用 BubbleSort 所以明顯慢其他方法非常多，甚至 50 萬與 100 萬的資料執行太久所以改成用 30 萬呈現他效率慢的程度，方法二的部分則是減少了每次 BubbleSort 排序的資料量，所以效率也快了很多，再來是方法三和方法四，看圖跟表格可以發現誰比較有效率跟資料量的大小有關，10 萬前方法四比較有效率，自己在私底下也有試過 1000 筆資料，也是方法四明顯快很多，10 萬後卻反過來，我認為應該跟我有用 Manager 的 queue 有關係，但不確定。

2.



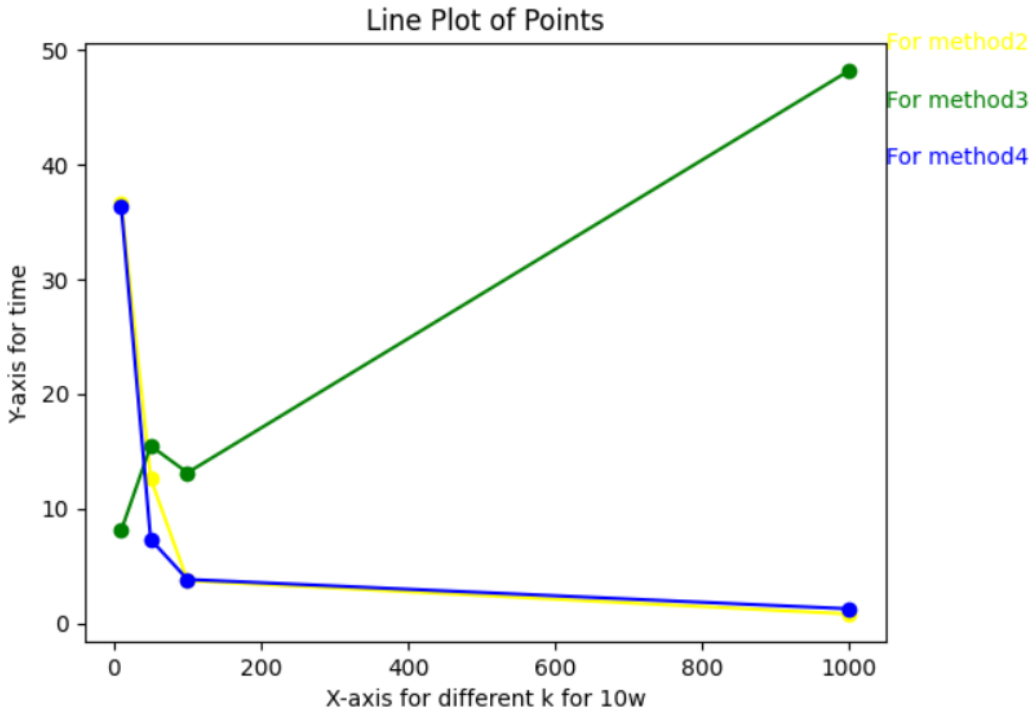

```
請輸入檔案名稱：input_1w.txt
請輸入要切成幾份：10
請輸入方法編號(方法1,方法2,方法3,方法4):2
Sorted data has been saved to input_1w_output2.txt
PS C:\Users\user\Desktop\os1> & C:/Users/user/anaconda3/envs/pytorch/python.exe c:/Users/user/Desktop/os1/hw1.py
請輸入檔案名稱：input_1w.txt
請輸入要切成幾份：10
請輸入方法編號(方法1,方法2,方法3,方法4):3
Sorted data has been saved to input_1w_output3.txt
PS C:\Users\user\Desktop\os1> & C:/Users/user/anaconda3/envs/pytorch/python.exe c:/Users/user/Desktop/os1/hw1.py
請輸入檔案名稱：input_1w.txt
請輸入要切成幾份：10
請輸入方法編號(方法1,方法2,方法3,方法4):4
Sorted data has been saved to input_1w_output4.txt
```

	K = 10	K = 50	K = 100	K = 1000
方法二	0.515	0.252	0.240	0.290
方法三	1.398	2.529	4.466	39.506
方法四	0.526	0.264	0.290	0.799

表二不同 K 值的各個方法執行時間(N 值固定 1w)

因為方法一不受 k 值影響不做討論，方法二明顯 K 越大越快，因為 BubbleSort 一次跑的數據量減少，而方法三會隨著 K 越大效率越慢，推測是當 process 越多時，會需要耗費更多時間做 context switch，因此效率明顯變差。方法四效率都很快應該就是因為多個 thread 不僅可達到同時運算，又能共用地址空間。

3.



	K = 10	K = 50	K = 100	K = 1000
方法二	36.669	12.615	3.712	0.813
方法三	8.104	15.469	13.123	48.169
方法四	36.291	7.309	3.824	1.265

表三不同 K 值的各個方法執行時間(N 值固定 10w)

結論與第二點大致相同，但因為資料量十分龐大，thread 同時運算，又共用地址空間的優勢大幅上升，所以 K 值越大效率越好的情況非常明顯，同時觀察了工作管理員在執行方法三時，處理程序會變多，CPU 使用率也會變高。