OS_HW1 程式說明文件

一、開發環境

1.處理器：Intel(R) Core(TM) i7-10750H
2.記憶體：8.00 GB
3.系統類型：64 位元作業系統，x64 型處理器
4.Windows 規格：Windows 10 家用版
5.IDE：Visual Studio Code
6.使用語言：Python

二、實作方法和流程

1.先請使用者輸入檔案名稱，開檔後將 input 檔讀入至一個 list 中(array)，再請使用者輸入任務編號(1~4)，依據編號分別執行程式內容。

2.任務一：

將 input 資料(array)直接呼叫 BubbleSort()作排序，並記錄執行時間，最後呼叫 OutputFile()將排序好的內容輸出至檔案。

3.任務二：

將 input 資料(array)透過 cutArrays()切成 k 份，再利用 threading.Thread()，由 k 個 threads 分別執行 BubbleSort()，再用 k-1 個 threads 作 MergeSort()，並記錄執行時間，最後呼叫 OutputFile()將排序好的內容輸出至檔案。

4.任務三：

將 input 資料(array)透過 cutArrays()切成 k 份，再利用 multiprocessing.Process()，由 k 個 processes 分別執行 BubbleSort()，再用 k-1 個 processes 作 MergeSort()，並記錄執行時間，最後呼叫 OutputFile()將排序好的內容輸出至檔案。

5.任務四：

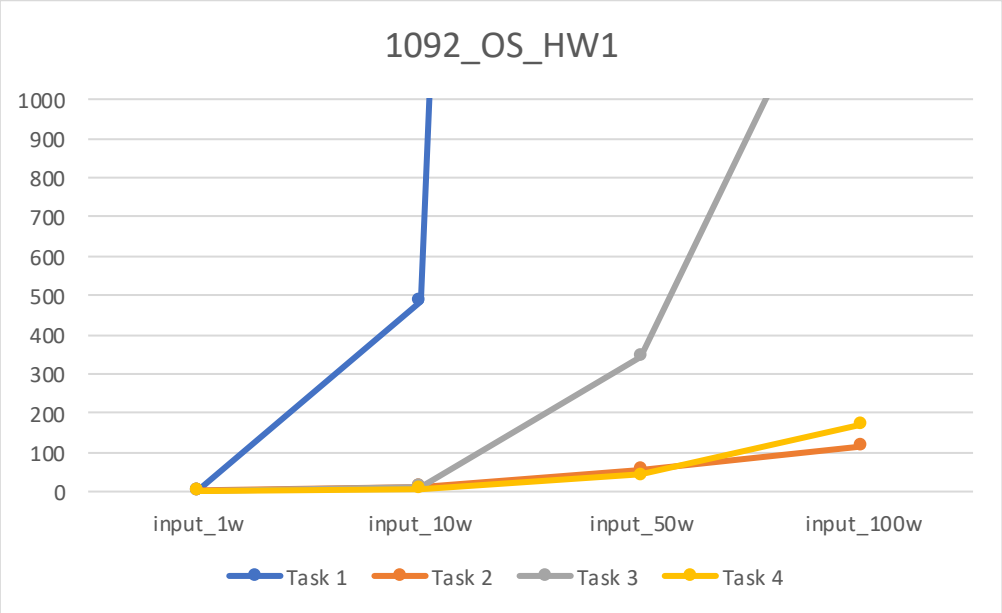
將 input 資料(array) 透過 cutArrays()切成 k 份，再直接呼叫 BubbleSort()作排序，接者呼叫 MergeSortForSingleProcess()，作 MergeSort()，並記錄執行時間，最後呼叫 OutputFile()將排序好的內容輸出至檔案。

三、使用的資料結構

1.list
2.queue.Queue()
3.threading.Thread()
4.multiprocessing.Process()
5.multiprocessing.Manager().Queue()

四、完成的功能四種作法進行比較

	input_1w	input_10w	input_50w	input_100w	切k份	單位
Task 1	4.65	487	13891	69455	1000	sec
Task 2	3.09	12.32	55.61	115	1000	sec
Task 3	1.25	12.11	345.3	1508	10	sec
Task 4	1.17	6.64	43.42	171.2	1000	sec



五、分析結果和原因

1.Task1：BubbleSort()在 Python 中表現不佳，與 C/C++相比甚至可差 10 倍以上，猜測可能是因為 C/C++對於記憶體存取的方式叫直接，而 Python 畢竟是一種 Interpreter，靈活度較差。

2.Task2 與 Task4 的執行時間接近，可能需要測試更大的資料筆數才有顯著的差異，或是將時間區隔縮小再作討論。

3.根據助教提供的資料，Task3 應該是最快的方法，猜測可能是在 multiprocess 之間的資料存取浪費了太多時間所致。

4.Task2 是利用同一個 process 衍伸出多個 threads 執行，而 Task3 是產生多個 process 執行，同一個 process 的好處是各個 thread 之間可以共用 data，反之，各個 process 之間有獨立的資料空間，因此可能是因為資料存取的便利性，使得 Task2 的速度較 Task3 快。