OS\_HW1程式說明文件

1. 開發環境
2. 處理器：Intel(R) Core(TM) i7-10750H
3. 記憶體：8.00 GB
4. 系統類型：64 位元作業系統，x64 型處理器
5. Windows 規格：Windows 10 家用版
6. IDE：Visual Studio Code
7. 使用語言：Python
8. 實作方法和流程
9. 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。
10. 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。

1. 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。

1. 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。

1. 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。

1. 使用的資料結構
2. list
3. queue.Queue()
4. threading.Thread()
5. multiprocessing.Process()
6. multiprocessing.Manager().Queue()
7. 完成的功能四種作法進行比較



1. 分析結果和原因
2. 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。
3. 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。
4. 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。
5. 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。