一、開發環境:

Windows 10, Visual Studio Code，語言使用 python version3.8

二、實作方法和流程:

先將檔案讀到 list 中，並將其轉為 integer。

方法一：撰寫 BubbleSort function，並將 list 作為參數傳入 function 內，並在 function 內部計時且輸出 output 檔案。

方法二：使用 class 將 list 作為 data member，並在 class 內寫出 BubbleSort 以及 MergeSort，利用使用者輸入的份數(k)，計算每個 thread 需要 Sort 的資料數量，利用前面算出的數量來記錄要排序的數字為幾號到幾號，開始計時且利用迴圈和 list 儲存 BubbleSort 參數(切好的 list)，並將 list 切為 k 份暫存到另一個 list 以便後面的 MergeSort，若有沒除盡的情況(總數/k 不為 0)，則將後面沒分到的數字加入最後一個 thread 內排序，呼叫 thread 做 BubbleSort，等所有 thread 做完工作後，用迴圈利用前面暫存的 list 算出要做 MergeSort 的 index 範圍，並 pop 最前面兩個 list，繼續算 MergeSort 的範圍，直到 list 的長度為 1，跳出迴圈並用 thread 做 MergeSort，等做完後即為排序完成，然後輸出 output 檔案。

方法三：在 class 內在撰寫一個有 return list 的 BubbleSort(因為 Process 需要用 return 的方式拿到排序好的 list)，利用迴圈將 list 切為 k 份並暫存起來，若有未被分配到的數字則加到最後一組 list 內，然後呼叫 k 個 process 來執行 BubbleSort，並將回傳的 list 們利用 for 迴圈加起來，即為分 k 份排序好的 list，再利用前面暫存的 list 計算要做 MergeSort 的兩個 list，將其合併加到暫存 list 最後面，再 pop 最前面的兩個 list，直到 list 長度為 1，呼叫多個 process 並利用暫存的 list 來做 MergeSort，再取最後 Merge 的結果，最後輸出 output 檔案。

方法四 :利用使用者輸入的份數(k)，計算每次需要 Sort 的資料數量，利用前面算出的數量來記錄要排序的數字為幾號到幾號，開始計時且利用迴圈呼叫 BubbleSort 來排序前面紀錄的範圍內的數字，並將 list 切為 k 份暫存到另一個 list 以便後面的 MergeSort，若有沒除盡的情況(總數/k 不為 0)，則將後面沒分到的數字加入最後一組 list 內排序，等所有 list 都排序好後，用迴圈利用前面暫存的 list 算出要做 MergeSort 的 index 範圍，並做 MergeSort，最後將 MergeSort 的結果加到站吋的 list 最後面，並 pop 最前面兩個 list，繼續做 MergeSort，直到 list 的長度為 1，即為排序完成，然後輸出 output 檔案。

三、四種做法的比較 ：

單位 ：秒

k = 20	1w	10w	50w	100w
Task1	3.631468	400.4561	13252.55	60530.39
Task2	0.560124	113.2592	672.1688	3482.177
Task3	0.365098	28.65755	247.2766	1097.469
Task4	1.437227	159.9848	871.3112	4515.294

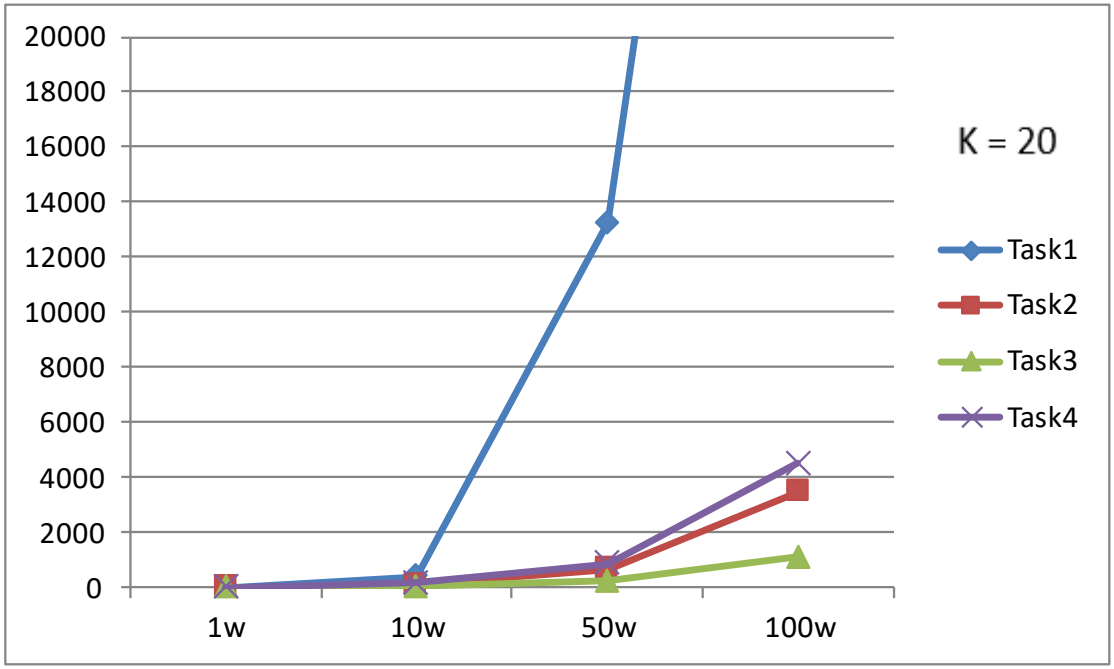
k = 100	1w	10w	50w	100w
Task1	3.631468	400.4561	13252.55	60530.39
Task2	0.263202	25.50172	139.9886	567.4999
Task3	0.233394	3.37765	57.08068	215.4418
Task4	0.254813	37.46947	181.8636	707.0321

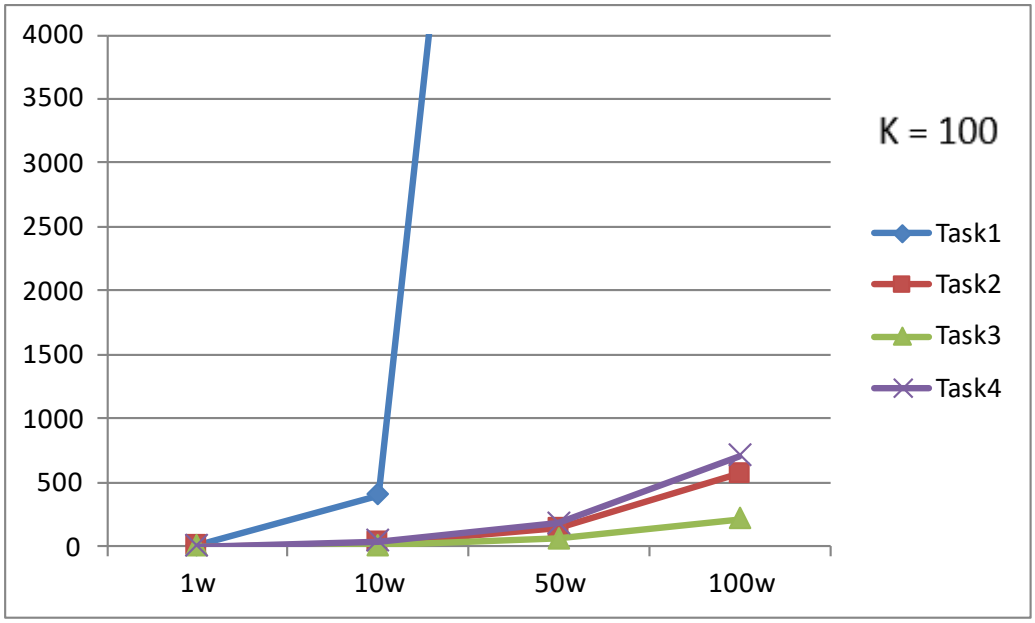
k = 2000	1w	10w	50w	100w
Task1	3.631468	400.4561	13252.55	60530.39
Task2	0.790626	9.133722	19.24178	55.27014
Task3	0.435959	3.475094	11.86141	30.60698
Task4	0.87753	10.82921	20.27413	58.59764

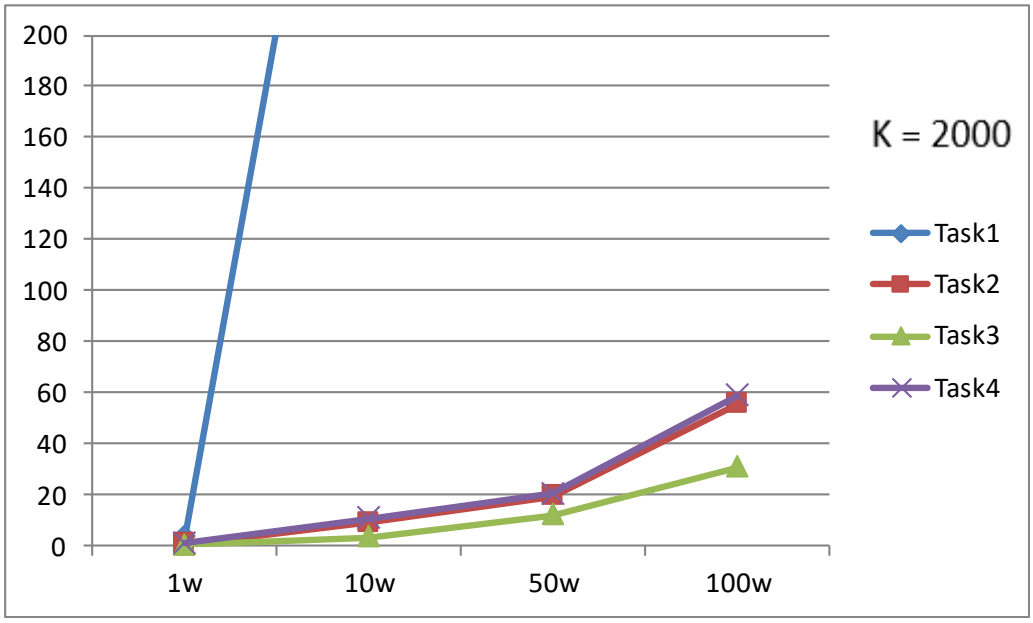
k = 5000	1w	10w	50w	100w
Task1	3.631468	400.4561	13252.55	60530.39
Task2	1.570914	11.03483	22.32825	54.84244
Task3	0.938944	5.488858	16.06965	39.08751
Task4	1.062725	11.65094	16.4649	40.79996

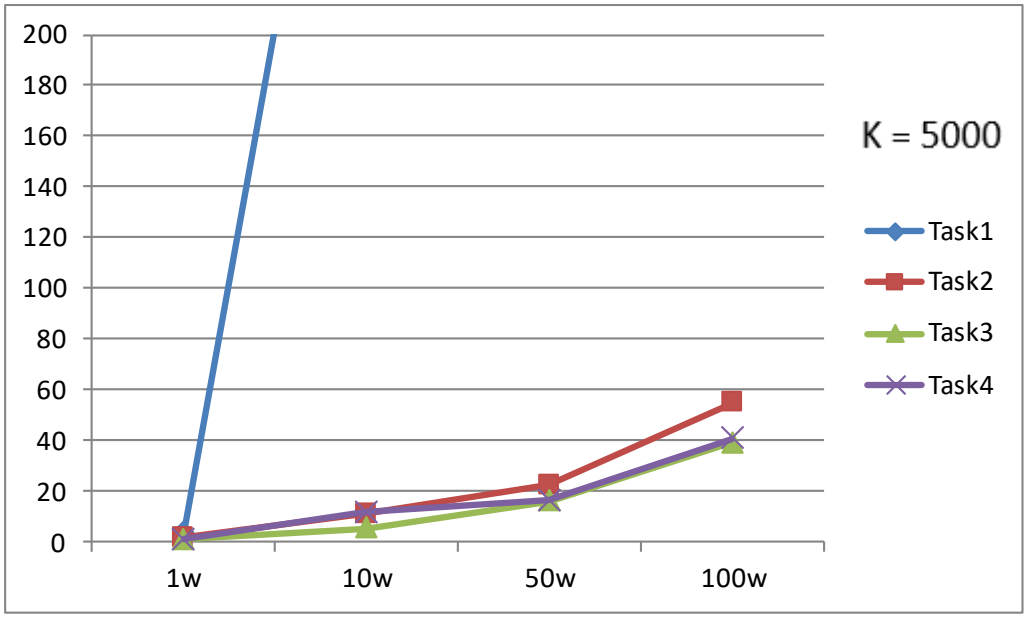
由上面表格可看出，執行速度較快的應為 Task3 > Task2 > Task4 > Task1。
由於 process 內有多個 thread，故 Multiprocessing 速度比 Multithreading 來的快，而 thread 可以 concurrent 執行，且會共享 address space，故 Multithreading 的速度又比單純切成 k 份做 BubbleSort 及 MergeSort 快，而 BubbleSort 的 running time 為$\Theta(n^2)$，MergeSort 的 running time 為$\Theta(n*\log n)$，故將資料切成 n/k 個做 BubbleSort 再做 MergeSort 會比直接所有資料做 BubbleSort 快，故執行速度結果為 Task3 > Task2 > Task4 > Task1。

四、分析結果與原因 ：




4000
3500
3000
2500
2000
1500
1000
500
0
1w
10w
50w
100w
K = 100
Task1
Task2
Task3
Task4


200
180
160
140
120
100
80
60
40
20
0
1w
10w
50w
100w
K = 2000
Task1
Task2
Task3
Task4


200
180
160
140
120
100
80
60
40
20
0
1w
10w
50w
100w
K = 5000
Task1
Task2
Task3
Task4

由上面四個圖表可看到，Task1 的資料筆數越多，所花的時間就越多。Task2,3,4 則是依照所切的份數(k)，切的份數越大速度越快，但是在資料筆數較少時(1w)可看到，當 k = 100 時，Task2,3,4 所花的時間最少，Task2,3 當 k 再往上增加時，所花的時間反而變多了，可能是因為當切了更多份數時，所開的 Thread 跟 process 數量也跟著增多，但是因為資料筆數較少，呼叫 Thread 跟 process 的代價反而會比直接做 Sort 來的大，故較慢。而 Task4 則是因為 BubbleSort 的 running time 為$\Theta(n^2)$，MergeSort 的 running time 為Θ (n*logn)，故將資料切成小份做 Sort 會快很多，所以並沒有這個問題。

而當資料筆數開始增多時，即可看出 Multithreading 跟 Multiprocessing 優勢，速度明顯比 Task1 快很多，且 Multiprocessing 明顯比其他 3 個 Task 快很多，但是 Task2,3 當切的份數到達 5000 份時，反而速度比切成 2000 份時慢，這個現象在 Task3 較明顯，我認為可能是因為開啟一個 Process 的代價比開啟一個 Thread 的代價來的高，所以開 5000 個 Process 對速度的影響較大，故在 Task3 k = 5000 的時候會比 k = 2000 來的慢。