10520104 電資四 莊東翰

1.開發環境:
Windows10 64 位元 家用版
CPU:I7-8700
RAM:16GB
程式語言:Python 3.7.2, cmd 執行
Thread: 使用 import threading
Process: 使用 from multiprocessing import Pool

2.實作方法和流程:
切割方法:若 n 筆資料切成 k 份，若 n%k==0，每一份 k 的資料大小為 n/k。
若 n%k != 0，每一份 k 的資料大小為(n/k)+ 1。

方法 1: 單純使用 bubblesort，待排序完成後寫檔

方法 2: 利用計算好的每份大小切割所有資料成 k 份，並將每個區段分別分配一個 thread，等區段全部分配完後全部 threads 一起執行(thread.join())。
Bubblesort 完成後再進行 k 份 merge 的動作，第一回合 merge 產生 k/2(有餘數則 k/2+1)個 threads，先分配好每個區段後再一次執行(thread.join())。下一回合的 merge 就會是(k/2)/2 (有餘數則(k/2)/2+1)個 threads，以此類推。最多同時運行的 threads 數不會超過 k 個，merge 共需花費 k-1 個 threads。

方法 3: 同方法 2，將 threads 改為 processes。不同點為 process 有 pool，宣告的 process 都會在 pool 中，要先 pool.close()確保不會再宣告 process 後再利用 pool.join()將 pool 中 processes 全部一起執行。

方法 4:一個 process 就代表程式本體，所以不需要使用 multiprocessing。直接程式碼執行方法 2 的實作部分，但不使用 multiprocessing。

3.方法比較:
首先，有用到 threading 或 multiprocessing 的方法若 k 份切太多反而效率會變差，因為花太多時間都在生成 thread 或 process。(若 k 越多代表要生越多的 thread 或 process 去指派區段給他們排序)

方法 1:一定是最慢的，因為沒有切割也沒有使用 multiprocessing 或 threading 並行運算。

方法 2:使用 threading+切割。切割後的 merge 可以提高效率(divide & conquer)，且使用 threading 可以讓程式並行執行，沒有使用到共用資源的部分可以併行計算，縮短時間。

方法 3: 使用 multiprocessing+切割。切割後的 merge 可以提高效率(divide & conquer)，multiprocessing 我認為不只有 threading 部分的優點，且因為 threading 是每個 thread 共用一個 time slice，所以能使用的總 time slice 量不變。但因為每個 process 都能分到一個 time slice，可以有較多的 time slice，執行效率比較高，所以會比方法 2 快。

方法 4: 約略比方法 2 慢一點，因為沒有讓程式並行處理，但因為有切成 k 份作 merge 的動作所以比方法 1 快。

4.分析結果:
資料越多，執行時間隨著資料量呈指數成長。100w 的所有方法都沒有跑，10w 方法 1, 50w 的方法 1, 方法 4 也沒跑，因為時間不足。數值在 0 以下的座標點為沒有跑結果的情形。

1w 方法 1: 29.2 秒, 方法 2: 2.54 秒, 方法 3: 1.41 秒, 方法 4: 2.68 秒
10w 方法 1: 沒跑, 方法 2: 250 秒, 方法 3: 56 秒, 方法 4: 308 秒
50w 方法 1:沒跑, 方法 2: 6517 秒, 方法 3: 1531 秒, 方法 4: 沒跑
100w 皆沒跑