OS Homework1

資訊三乙 11027209 巫年巨

開發環境 : python , 版本 3.10.7

實作方法與流程

首先講方法 1 就是普通的泡泡排序但是加上了 bool 值看這一輪有沒有排序，沒有的話代表 結束了，避免多餘的空次數做比較，優點是如果提早結束的話可以減少很多比較次數，缺點的話是 worst-case 會造成的話更多的比較。

第二題的題目把 array 切成 k 份，並把這些拿去分別做泡泡排序後，在當資料不為 1 時，一次抓兩筆資料做 mergesort 並合併為一筆資料，直到剩下最後一筆完成排序的資料。 由於這次的 mergesort 是拿 bubble sort 完的陣列去排序所以可以把 mergesort 寫成 merge 的形式把兩個 array 依照小到大合起來成一個 array 就好。

第三題大致步驟跟第二題很像，但是在切資料時我是每切一份生一個 process 去做那一份的 sort，而為了切的時候資料是共享的，所以用了 Python 的 multiprocess 庫裡面的 Manager 來做到 share memory 的效果，在 process 們 sort 完後，主 process 會等到他們做完才繼續下面的步驟，之後把切完的資料拿去 merge，這段我是用 Pool 來設定 process 的數量，我發現如果自己開 process 來，每次開現在有的份數/2 的 process 去 merge，直到只剩一筆資料為止，就能拿到答案，這題比較難的是 share memory 的部分，因為 Python 的 share

memory 的資料型態與原本的資料型態不同用 List 舉例可以用+=的 overload operator 但是變成 share 的後他會變成 ListProxy 的型態，這時候對他拿去用 opeator 的型式的話，比如在 Bubblesort 中的 swap 的話，可能會造成速度緩慢甚至到無限迴圈的窘境，ListProxy 最好只要用 List 有的 function 如 append,pop,index 等，雖然都是 List，但這樣兩個是操作上彈性度很不同，由 manager 形成的 ListProxy 雖然也叫 list 的少了有些 list 能做的操作，不過換個方面想也是畢竟這是被共享的記憶體區段。

第四題是用 thread 跟上面兩題差不多都是在切資料時就生對應數量的 thread，由於 thread 是同一 process 拉出來的且有共用 address space 的優勢存在，所以不用煩惱共用 memory 的問題存在，所以整體下來方便許多，而 thread 跟 process 比有個優勢是在 windows 有限制同時只能有 62，而 thread 沒有限制，在解這裡 1 百萬筆資料時用 thread 就是不錯的選擇，但 thread 也要注意等到做完，主 process 才能往下一步，必須先 join 在那邊等待才行，不然會有 method 跑完 thread 還在跑 sort 的問題，這樣會使答案最終不正確。

探討結果和原因

K = { 5, 17}	N = 1 萬	N = 10 萬	N = 50 萬	N = 100 萬
方法 1	4.1,4.1	433,433	11833,11833	55369,55369
方法 2	0.78,0.22	76,23	2612,635	12412,2420
方法 3	0.75,0.8	20,4	1512,97	9537,494
方法 4	0.86,0.21	85,25	2358,610	11325,2110

表 1：實驗記錄表格 (不同 N 的比較時間)(單位：s)

從上表可看出資料量 N 越來越大的時候，各方法的時間也越來越大，從方法 1

來看資料量上升了 10 倍，而執行的時間就差了 100 多倍左右，而方法 234 不同於 1 是多了切資料和 merge 這段，但是可以很明顯的看出來執行時間快了很多，方法 3 和 4 是多了耗 cpu 資源所以比 2 更快。

N = {1,10,50,100 } 萬	K = 1	K = 13	K = 50
方法 1	4.1,433,11833,55369	4.1,433,11833,55369	4.1,433,11833,55369
方法 2	3.9,425,11351,52518	0.29,34,764,2946	0.15,8.6,299,680
方法 3	4,421,13211,62172	0.59,5.51,147,952	0.53,4.1,42,167
方法 4	4.2,418,12258,54563	0.28,31,724,2761	0.1,8,273,610

表 2：實驗記錄表格(不同 k 的比較時間) (單位：s)

由於方法 1 不用切資料，所以 k 值影響對 1 來說不影響，而方法 234 在當資料量 K 越來越大的時候，可看出執行的速度會越來越快，從方法 2 來看資料量是 50 萬的時候 K=50 比 K=13 的執行時間少了很多，而方法 34 速度比 2 可以很明顯的看出來執行時間快了很多，3 和 4 是因為耗了更多的 cpu 資源去建 Thread 和 process 所以比 2 更快。

將資料切成：1, 5, 13, 17,50 份 (K) 資料筆數：1, 10, 50, 100 萬 (N)

K = {1,5,13,17,50}	N = 1 萬	N = 10 萬	N = 50 萬	N = 100 萬
方法 1 (k 不影響)	4.1,4.1, 4.1,4.1, 4.1	433,433, 433,433,433	11833,11833, 11833,11833, 11833	55369,55369, 55369,55369, 55369
方法 2	3.9, 0.78, 0.29,0.22 ,0.15	425,76, 34,23, 8.6	11351,2612, 764,635, 216	52158,12412, 2946,2420, 680
方法 3	4, 0.75, 0.59,0.8, 0.53	421,20, 6.8,6.09, 4.1	13211,1512, 147,97, 42	62172,9537, 952,494, 167
方法 4	4.2,0.86, 0.28,0.21,0.09	418,85, 31,25, 8.1	12258,2358, 724,610 201	54653,10325, 2761,2110, 610

表3：實驗記錄表格 (單位：s)

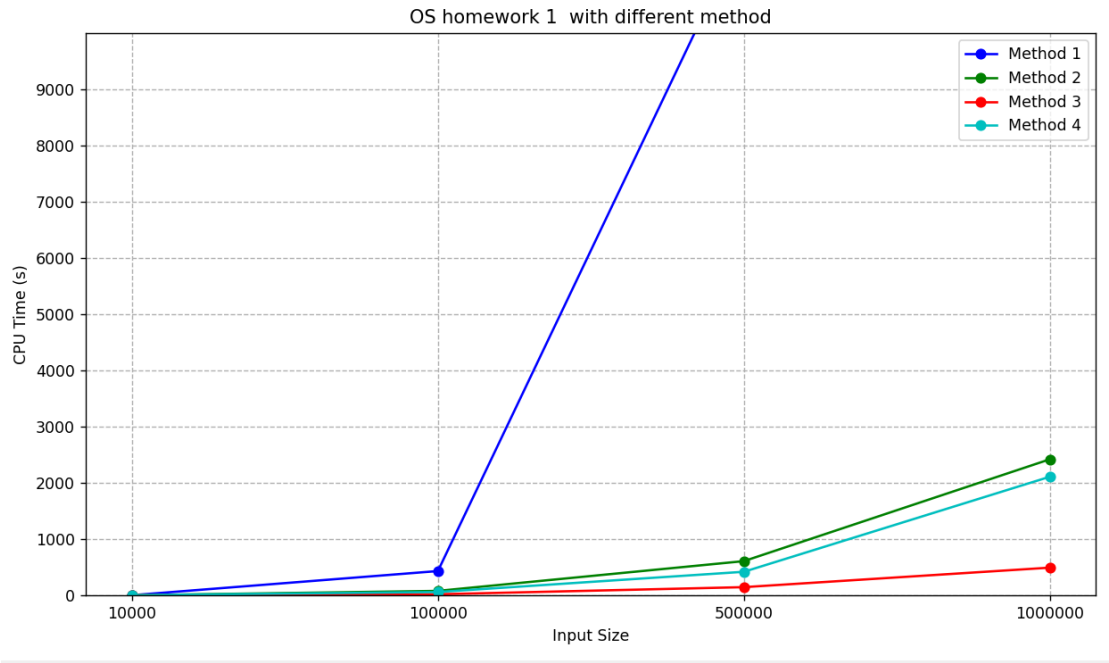

圖1：用k=1,k=5,k=13,k=17對應方法1234產生的圖

圖和表我們可以得出以下結論隨資料量 n 上升每次要跑的時間也會越多, BubbleSort 用演算法看要 O(n^2)，所以一萬筆資料跑的次數（迴圈）比一百萬筆資料差距就會到（100^2 = 10000），跑的時間上差距也會差到 10000 倍左右，用表格的數據去看，看到只有 1 萬筆資料時電腦跑大概約 4 秒左右，而到了 100 萬筆時跑的時間要到 50000 秒左右，所以整體看執行效率是相當的差。

方法 234，前面在做泡泡排序切成 k 個 array 在去 sort 假設把 10000 切成 10 份，那就只要跑 10 個 1000 筆的 bubble sort(比較的 if 次數計算 1000*1000*10)，跟直接跑 10000 比(比較的 if 次數計算 10000*10000)，少了 10 倍的比較，雖然方法 2 還要跑 merge 但是用比較來說也不會有原本的那麼多， 第三和四由於是開多個 process 或是 thread 來跑與方法二相同的流

程，在同樣的合理範圍 k 值下，理論上應該要比方法二快，因為開多個 process 和 thread 是叫 CPU 去耗費資源，來分工處理不同任務。

至於這次作業中我比較有疑惑的是任務 4 的部分，在我原本的猜想是開多個 thread 照理講要跟 3 差不多快，但是結果卻和 2 差不多，我去網路查資料發現，Python 的 interpreter 有個機制叫做 GIL(global interpreter lock)，為了保護執行 thread 的安全性， 防止同時和修改共用的資料(如上課教的 race condition)，所以同時只會有一個 thread 能 run，運作是拿到 GIL 的 thread 可以 run 跑完後會釋放 GIL 給別的 thread，所以跟方法 2 差不多。