OS project1-Process&Thread

資訊三甲 10927145 李俐瑩

一、開發環境:

Visual Studio Code

二、使用語言:

Python

三、實作方法、流程

- Mergesort:

  分成 merge()和 mergesort()兩個 function，merge 負責將兩個排序好的 list 合併並回傳，合併時不斷取兩個 list 最前端較小的，而 mergesort()這個 function 用來控制哪兩個 list 要執行 merge()這個動作，使用遞迴或迴圈來完成，執行結束的條件為資料的長度為一，代表已合併成一個 list。

- 方法一:

  將整個 file 讀入後，使用 split()，依據換行符('\n')將檔案存成 list，直接丟入 bubblesort() 進行排序，印出結果並印出排序時間。

- 方法二:

  將 file 讀入後，使用 split()，依據換行符('\n')將檔案存成 list，並根據使用者設定的 k 值將 list 切成 k 份，建立一個 multiprocessing

中的 Pool，將 k 份資料進行 bubblesort()後再用同一個 process 進行 mergesort，最後輸出結果並印出排序時間。

- 方法三:

方法三與方法二的差別於方法三須建立多個 process，我使用了 multiprocessing 內的 pool 來完成，首先要先將 pool 的 size 設為 cpu 內的核心數，避免一個核心同時進行兩個以上的 process，而 pool 內的 process 執行完後會讓新的 process 進去 pool 內執行，省去了我做 context switch 的動作，而實作方法為，第一個 pool 做 k 個資料的 bubblesort()，第二個就是用來執行 mergesort 最後輸出結果並印出排序時間。

- 方法四:

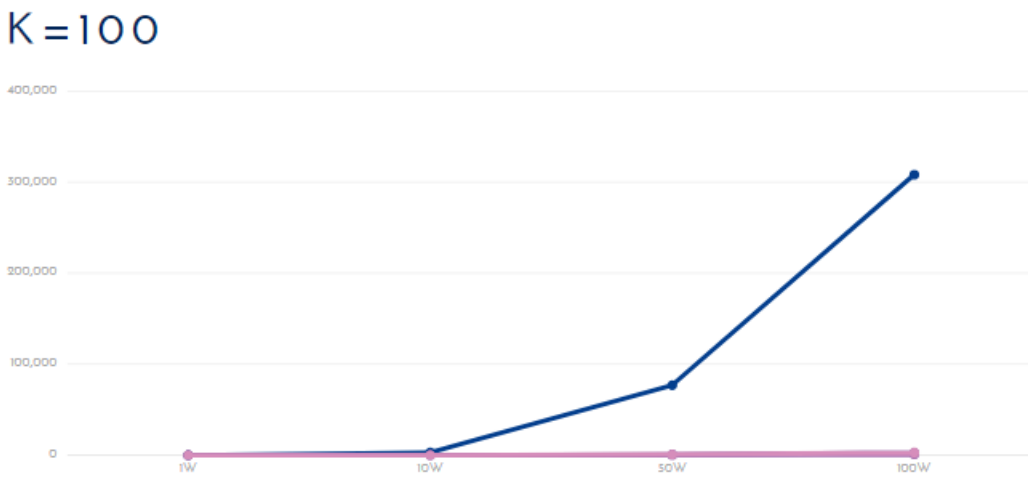
第四個方法是要建立多個 thread，我用了 multithreading 的 thread 建立，讓他們分別去跑 k 份的 bubblesort()，再用一樣的方法去跑 mergesort，和其他方法不同的地方是，我用了 queue 來存取合併的結果最終輸出結果並印出排序時間。

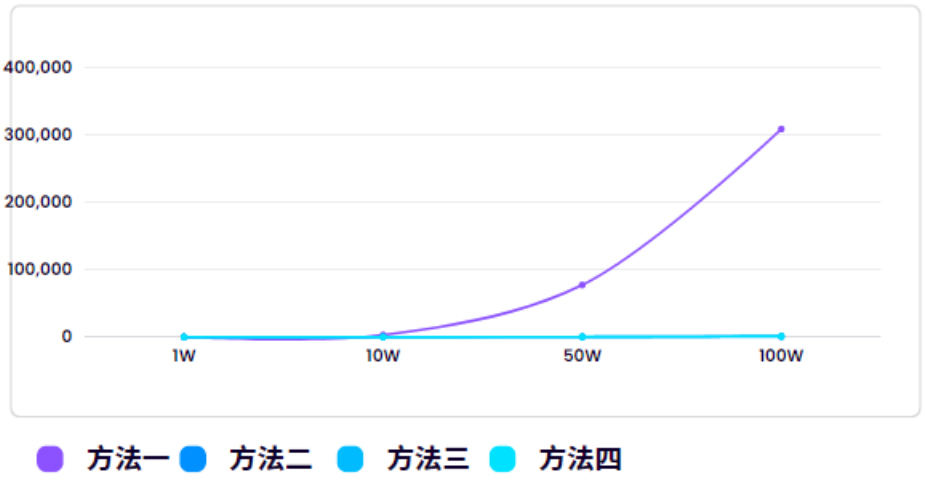
四、探討結果與原因:

表 1.相同 k 值(k=100、250)不同 N 值:

K=100,k=250	N =1w	N =10w	N=50W	N=100W
方法一	26.389	2854.399	77068.773	308275.092
方法二	0.307, 1.135	28.197, 12.975	650.855, 260.168	2726.582,1090.635
方法三	8.727, 14.593	19.291, 17.812	270.369, 108.936	1096.624, 435.312
方法四	0.834, 1.381	29.312, 12.850	693.697, 300.614	2810.893,1124.356

(單位:s、取小數點後三位)





方

法一可以很明顯地從圖中看出越多筆資料，需要的時間多了非常多，因為單純使用 bubblesort，所以時間複雜度為 O(n^2)，因此需要執行的迴圈次數便是資料筆數的平方，所以執行時間才會那麼高。而方法二及方法四的時間差不多，因為都是只用了一個 process 來執行，而方法三用了 k 個 process 分工所以時間較快，但當資料筆數沒有那麼多時並不需要那麼多 process 來執行，反而增加了時間。

表 2.相同 N 值(N=1w)不同 k 值:

N=1w	K=100	K=1000
方法一	26.389	26.389
方法二	0.307	1.976
方法三	8.727	17.865
方法四	0.834	4.422

(單位:s、取小數點後三位)

方法一因為做法與 K 值無關所以時間一樣，而當 K 值上升時代表 merge 所需的時間越高，而 bubblesort 的時間越短，因此雖然 bubblesort 所需時間變短了，但因 k 值變成 10 倍，所以 merge 所需時間多了 10lg10 倍，因此對一萬筆資料來說分割成 1000 份太多了。

五、