1. 開發環境 :

C++ (在 Ubuntu 上執行)

2. 實作方法和流程 :

這次作業分成**四個部分**：

方法一 → 將 N 筆資料直接 Bubble Sort

方法二 → 將 N 筆資料切成 K 份，在一個 process 內對 K 分資料作 Bubble Sort，再用同一 process 做 Merge Sort

方法三 → 將 N 筆資料切成 K 份，並用 K 個 processes 分別做 Bubble sort，再用 process(es)做 Merge Sort (k-way Merge)

方法四 → 將 N 筆資料切成 K 份，並用 K 個 threads 分別進行 Bubble Sort 後，再用 thread(s)做 Merge Sort (k-way Merge)

方法一的實作方法:

直接將檔案中的資料一次丟進 Bubble Sort 做排序，並輸出排好的檔案。

方法二的實作方法:

先把指定檔案中的資料分成 K 份(指定份數)，再一份一份分別塞給 Bubble Sort 做排序，最後再將所有的資料全部收回來做 Merge Sort(K-way Merge)，結束所有排序並輸出檔案。

方法三的實作方法:

先把指定檔案中的資料分成 K 份(指定份數)，再一份一份分別塞給新建立的 process 中，在個別的 process 中做 Bubble Sort，收回個別排好的資料，最後再將所有的資料全部收回來做 Merge Sort(k-way merge)，結束所有排序並輸出檔案。

方法四的實作方法:

先把指定檔案中的資料分成 K 份(指定份數)，再一份一份分別塞給新建立的 thread 中，在個別的 thread 中做 Bubble Sort，收回個別排好的資料，最後再將所有的資料全部收回來做 Merge Sort(k-way merge)，結束所有排序並輸出檔案。

3. 特殊機制考量與設計

使用者介面：因為方法一不需要將資料做裁切，所以使用者輸入介面的顯示順序為

➔ 請選擇離開或是執行(請輸入 0 或 1)：

➔ 請輸入檔案名稱 ：

➔ 請輸入方法編號(方法一、方法二、方法三、方法四)：

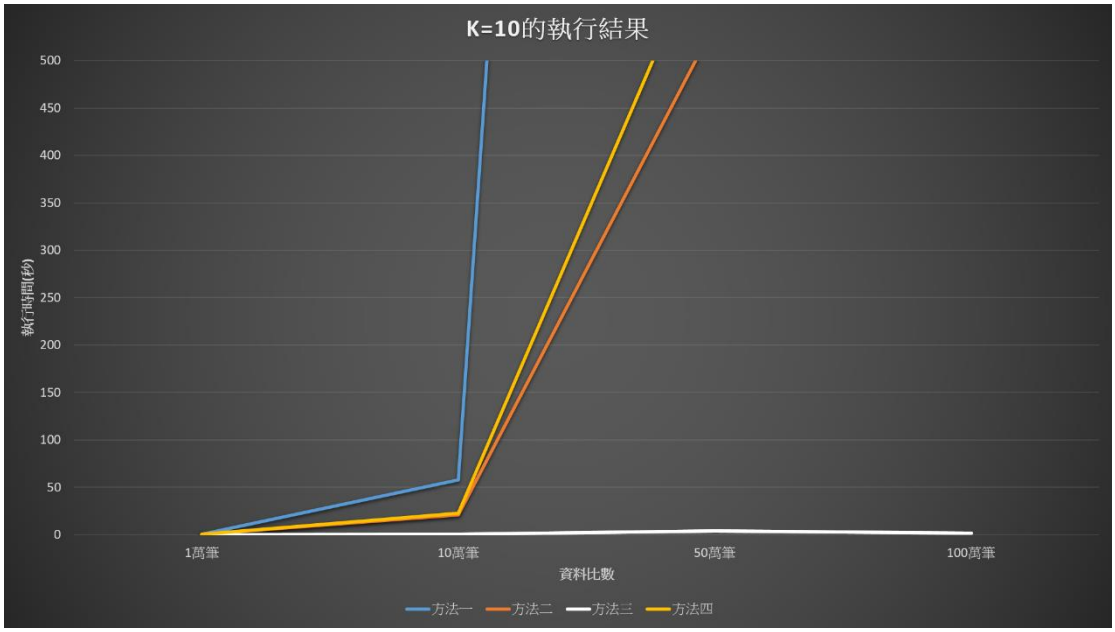
➔ 請輸入要切成幾份 ：

除了輸入順序上的改變，還加上了防呆機制，以免輸入超出原始資料量的切割份數，或是輸入非法字元。

4. 分析結果與原因(方法一不須切割份數)

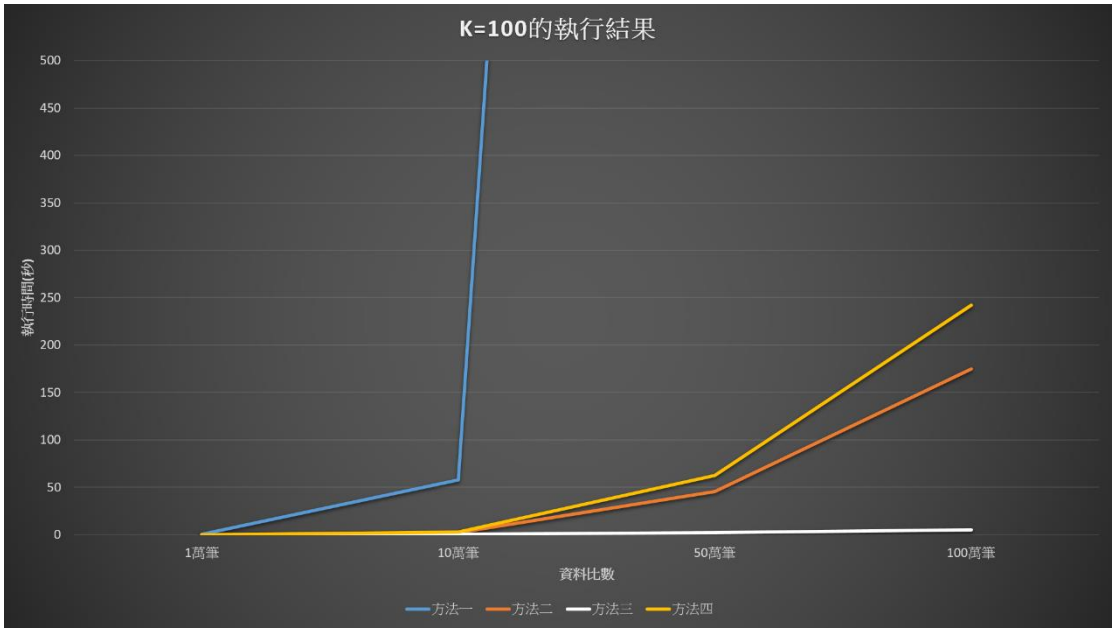
分 10 份	方法一	方法二	方法三	方法四
1 萬筆	0.563047	0.364383	0.075433	0.289274
10 萬筆	57.8084	20.7607	0.560404	22.4596
50 萬筆	4022.32	538.361	4.04789	652.064
100 萬筆	11248.8	546.945	1.74323	592.778

(單位為秒)



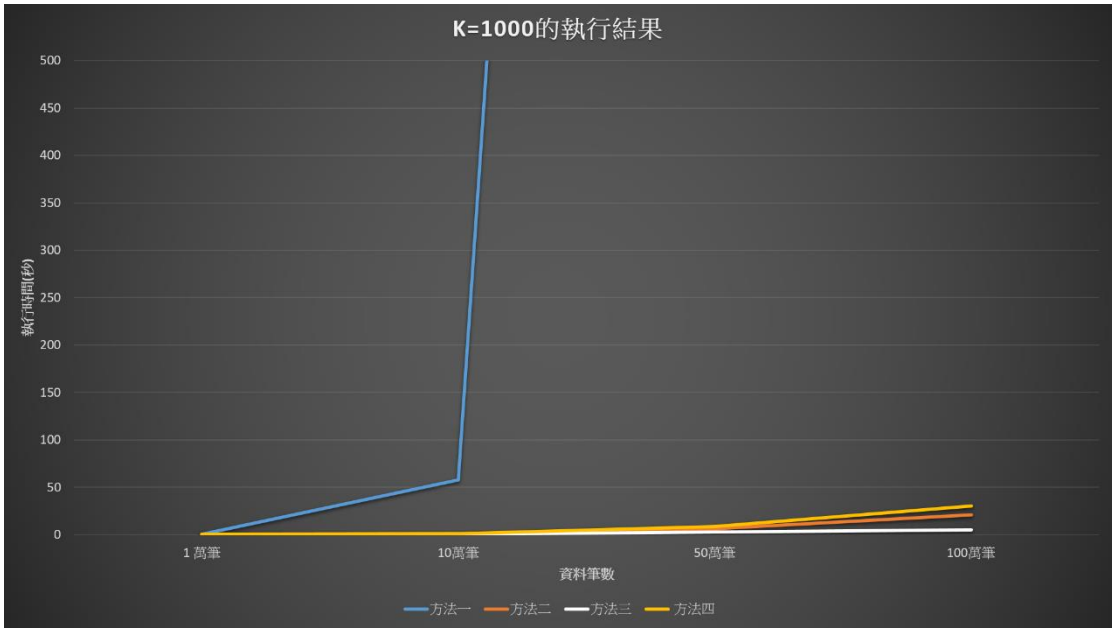
分 100 份	方法一	方法二	方法三	方法四
1 萬筆	0.563047	0.067268	0.052904	0.080861
10 萬筆	57.8084	2.200146	0.489094	2.96548
50 萬筆	4022.32	45.6374	2.52311	62.7267
100 萬筆	11248.8	174.623	5.06447	242.0224

(單位為秒)

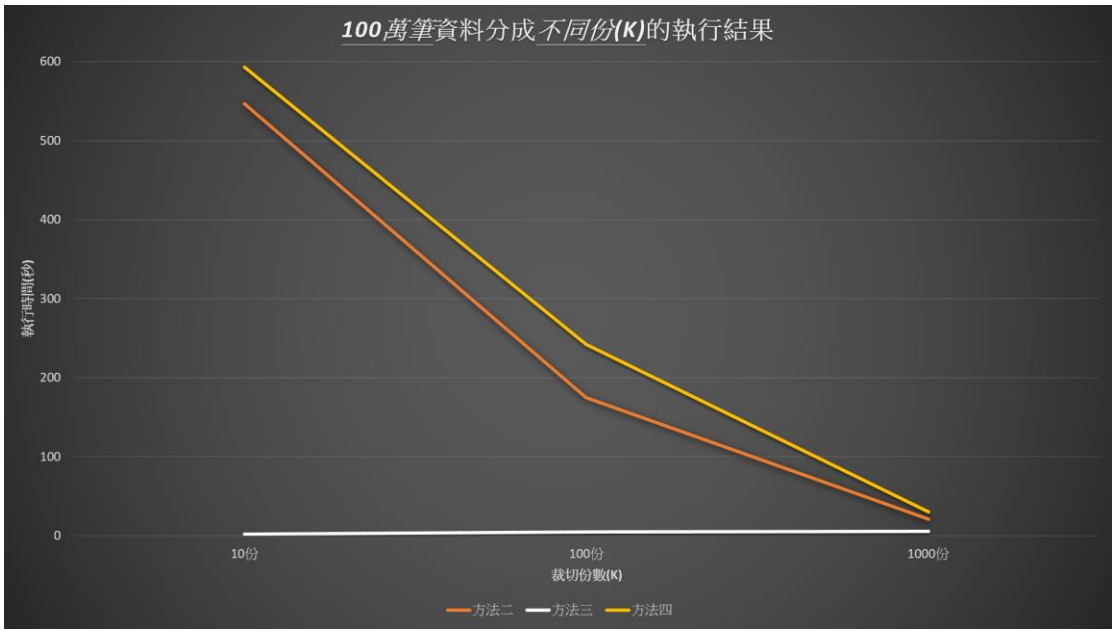


分 1000 份	方法一	方法二	方法三	方法四
1 萬	0.563047	0.055983	0.268355	0.22097
10 萬	57.8084	0.899103	0.666988	0.880824
50 萬	4022.32	6.5807	2.8276	8.78074
100 萬	11248.8	21.0509	5.19767	30.2354

(單位為秒)



由以上的圖表可以得知，不管分成幾份所執行的運算速度為：3 > 2 > 4 > 1，因為**方法 3** 使用子 process 做運算，而 process 擁有自己獨立的資料、資源與 code section，在執行時不需要與其他 process 互相競爭存取共用資源，所以運算速度會是最快的，若是分得愈多份(愈多子 process)，執行的速度愈快。反之**方法 4** 使用子 thread 做運算的速度會慢很多，因為每次從 process 中建立一個新的 thread，thread 與 thread 之間會共用同一個 process 中的資料、資源與 code section，所以在存取共用資源時會互相影響造成 delay(會有存取共用資料等待的時間)，導致運算時間會比方法三慢上許多。而**方法一**是直接將 N 份資料做 Bubble Sort，Bubble Sort 的時間複雜度為 $O(n^2)$，所以會是所有方法中最慢的。



而根據實驗結果可以得知，分愈多份的執行時間愈快(因為單次執行的資料筆數小)，因此若想要將大筆的資料快速地做好排序，使用方法 3 並分割多份一點，即可達到要求。

5. 撰寫程式時遇到的 bug&相關的解決方法

最後在寫方法三的時候發現，使用 fork()這個 system call 之後，無法得知子 process 執行的順序與結束的時間，因為題目要求為建立多個 process 執行 Bubble sort，所以需要收到各個 process 所排完序回傳的資料，而指令 fork()雖為創建一個新的子 process，但是因為此指令不是共用記憶體，父子 process 的資源與記憶體空間皆是獨立的，所以達不到題目要求，而且 fork()的 process 的執行順序不一定，是取決於排程演算法。因此改用指令 vfork()，此指令為創建一個新的子 process，且保證子 process 先執行後，直到 exit(0)父 process 才能繼續執行呼叫子 process 後中斷的地方，因此能確保子 process 先做完才會輪到父 process 執行。

fork():

```
jenny@jenny-virtual-machine:~/Desktop/OS$ g++ Project01.cpp -o Project01 -lpthread
jenny@jenny-virtual-machine:~/Desktop/OS$ ./Project01
請選擇離開或執行(請輸入0或1) -> 1
請輸入檔案名稱 : input_1w
請輸入方法編號(方法一, 方法二, 方法三, 方法四) : 3
請輸入要切成幾分 : 10
I am a new process -> 0
I am a new process -> 6
I am a new process -> 5
I am a new process -> 4
I am a new process -> 7
I am a new process -> 3
I am a new process -> 8
Subprocess finish executing !!!
I am a new process -> 9
I am a new process -> 2
I am a new process -> 1
Time --> 0.04965s
```

vfork() :

```
jenny@jenny-virtual-machine:~/Desktop/OS$ g++ Project01.cpp -o Project01 -lpthread
jenny@jenny-virtual-machine:~/Desktop/OS$ ./Project01
請選擇離開或執行(請輸入0或1) -> 1
請輸入檔案名稱 : input_1w
請輸入方法編號(方法一, 方法二, 方法三, 方法四) : 3
請輸入要切成幾分 : 10
I am a new process -> 0
I am a new process -> 1
I am a new process -> 2
I am a new process -> 3
I am a new process -> 4
I am a new process -> 5
I am a new process -> 6
I am a new process -> 7
I am a new process -> 8
I am a new process -> 9
Subprocess finish executing !!!
Time --> 0.049968s
```