# HW3 Report

## Q1. Sorting Algorithm

我將整個數列分成 NUM_OF_SEG 區段，對於每個區段，然後根據 NUM_OF_TRD 分給各區段一個 thread ，SEG > TRD 就先 Join 回來再不夠的話，將他們使用 Bubble Sort 的方法，將區段整理排序好（全部 thread 都 join 回來後再 create），再依同樣的分割方法，兩個分割分給一個 thread 去合併起來，合併方法與 merge sort 相同，可以在 O(n) 時間將兩個有序的數列，合成一個有序的合併數列。

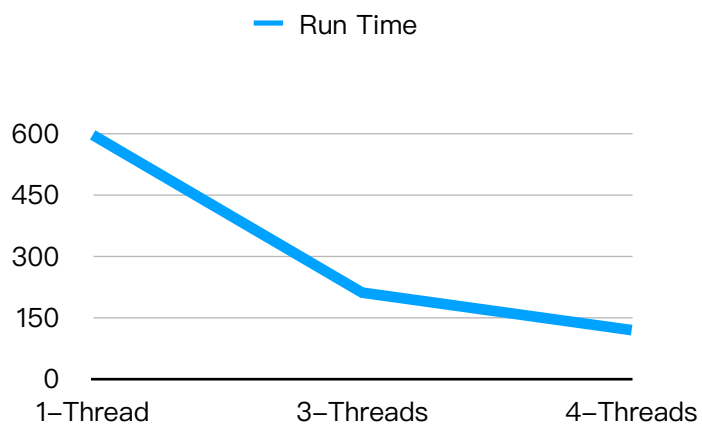
## Q2. Single-Thread and Multi-Thread

```
time (./st  < ./Testcase/input2.txt > output.txt)

real    7m37.005s
user    7m35.124s
sys     0m0.053s
time (./mt_worst < ./Testcase/input2.txt > output.txt)

real    3m30.468s
user    10m25.877s
sys     0m0.088s
time (./mt_best  < ./Testcase/input2.txt > output.txt)

real    1m58.644s
user    7m47.687s
sys     0m0.083s
```




左圖為執行時間，右圖為執行時間與 Thread 關係圖，可以看到從 Single Thread 到 Multi Thread 執行時間有明顯的下降。 mt_worst 為使用 3-threads，mt_best 為使用 4-threads。

## Q3. Description of Multi-Thread Acceleration

我先使用 lscpu 指令看計中的 CPU ，發現計中的電腦具有 4 個 CPU，所以當使用大於 4 thread 不會有太大的進步，且如果將 thread 的數量開太大，會需要更多的 overhead 來做資料或是 thread 的交換，所以我將 best 設為 4 thread ，worst thread 設為 3 thread，在 single thread 中，採用相同的 NUM_OF_SEG，下去測試秒數，實驗結果在 Q2 可以看到。而 Multi-thread 會比較快的原因就是可以同時使用不同的 CPU ，不同 thread 放在不同的 CPU 上面跑，會比整個區間都要在一個 CPU 上跑快。

## Q4. Learning

這次作業理解到了如何使用 pthread ，能夠在現今多 CPU 的電腦中，寫出一個程式發揮出最好的電腦效能。

另外，我將原本的 NUM_OF_SEG 調大成 16，降低每塊的大小，可以大大降低各個區塊在做 bubble sort 的複雜度，所以可以降低執行時間，但是時間差異就沒有那麼明顯，但是也可以看到執行時間 Single-Thread > Multi-Thread_worst > Multi-Thread_best。

```
time (./st  < ./Testcase/input2.txt > output.txt)

real    1m28.163s
user    1m27.922s
sys     0m0.141s
time (./mt_worst < ./Testcase/input2.txt > output.txt)

real    0m41.403s
user    1m48.696s
sys     0m0.192s
time (./mt_best  < ./Testcase/input2.txt > output.txt)

real    0m30.176s
user    1m55.870s
sys     0m0.153s
```