作業系統-作業一

開發環境：Xcode　　開發語言：C++　　10827117 陳柏宇

特殊機制考量與機制：

在Bubble_sort中，有多使用一個flag，當內層的迴圈跑完之後卻沒有做任何的交換動作，就表示此排序已經完成，可以稍微避免額外多做回圈。

撰寫程式的Bug：

在任務一中，沒有遇到Bug。

任務一

在任務一中，需要利用Bubble Sort且是單一一個Process來排序資料，並顯示CPU運行時間和日期。

實作方法與流程

在此任務中，會將檔案中的所有資料存入到Vector中，接著再將Vector丟給Bubble Sort處理。在Bubble Sort中，有兩個for回圈，最外面的迴圈 i 會從0跑到倒數第二個元素，裡面的迴圈 j 會從0跑到總長度扣掉 i 的長度再扣掉1，而在裡面的迴圈中，會有一個 if ，如果前面的元素大於後面一個元素，就交換，並且把 flag 設定為true，接著會結束裡面的迴圈，然後在外面的迴圈裡有一個if 判斷 flag 是否為 false ，如果是 false 就表示已經排序完畢，直接結束，如果不是，則繼續排序。Bubble Sort處理完畢後，就會直接輸出到檔案中，結束任務一。

特殊機制考量與機制：

在Bubble_sort中，有多使用一個flag，當內層的迴圈跑完之後卻沒有做任何的交換動作，就表示此排序已經完成，可以稍微避免額外多做回圈。

撰寫程式的Bug：

在任務二中，因為切割的份數有可能不會整除資料的數量，所以需要將多出來的資料從頭開始分配給每一份資料，以免剩下的資料被遺漏。

以下是解決的程式碼。

```
basic = num / part ;
remain = num % part ;

for ( int i = 0 ; i < part ; i ++ ) {

  index.push_back( more_less( basic, remain ) ) ;
  remain -- ;

} // for
```

任務二

在任務二中，會需要將輸入的檔案進行分割，分割的份數由使用者決定，得到分割的份數後，將每一份進行Bubble Sort，結束Bubble Sort後，再將每一份檔案進行Merge Sort，合併成一個排序好的檔案。

實作方法與流程

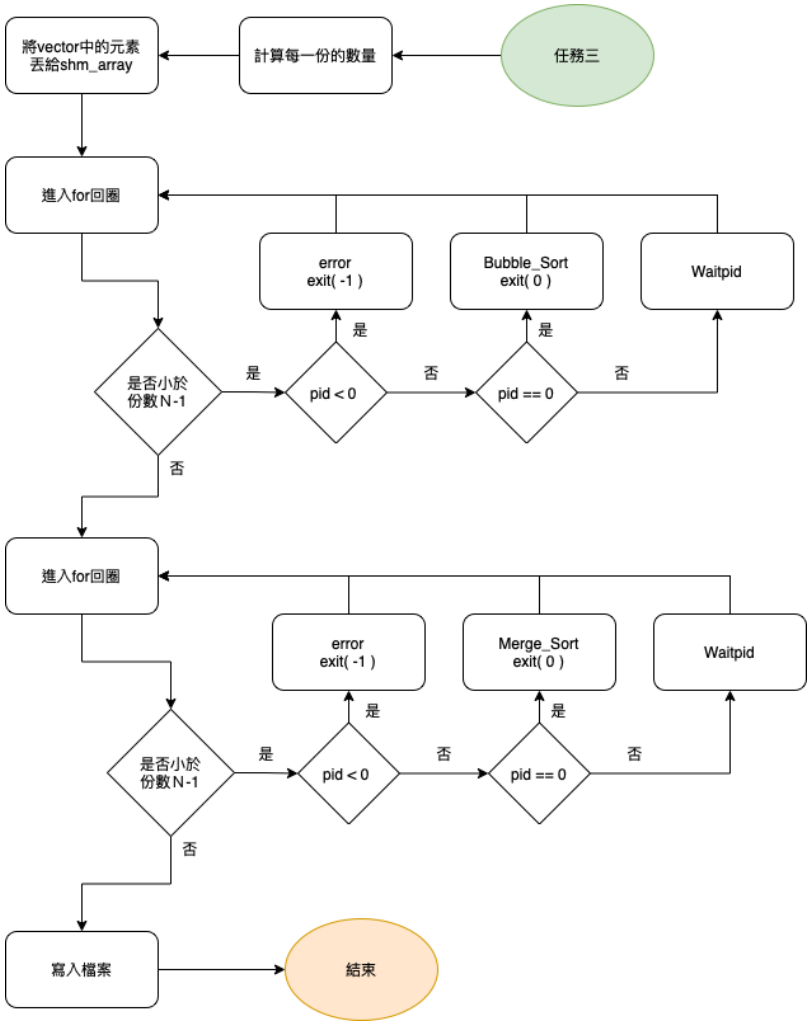
在任務二中，輸入資料後，讀取要分割的份數N，輸入完後，將資料放進一個Vector中，接著計算如果要分割成N份的話，每一份需要放入多少的資料量，計算好之後，將資料都分配好到不同的vector中，再將這些vector全部存在一個vector中，形成一個二維的vector。都分配好且儲存好之後，就會開始將每一份都做Bubble sort，做完Bubble sort就會接著做merge sort，在merge sort中，每一次只會合併最前面的兩份，合併後就會被放到最後面，變成最後一份，執行N-1次就可以把所有檔案皆合併完成，最後就是輸出檔案，並記錄排序時間和日期。

任務三

在任務三中，需要將輸入的檔案進行分割，分割的份數由使用者決定，得到分割份數K之後，利用K個Processes對每一份資料進行Bubble sort，結束Bubble sort後，再用Process(es)合併每一份資料。

實作方法與流程

在任務三中，讀取要分割的份數K，輸入完後，在此任務有改變了儲存資料的結構，這一次一樣會將檔案資料輸入進vector中，輸入後也會計算每一份所需的份數，但計算完成後，會利用計算結果，將每一份資料在陣列中的起始點和最後一個元素的位置記下來在另一個vector中，接著，會先在main中開啟一個共用資料，且是int pointer，原本儲存資料的vector的所有元素都會被複製到這一份指標所指向的陣列中，如此一來，所有的process就都可以共用這一個共享資料，對其進行改動。將資料都儲存好後，接著就是進行Bubble sort，而每一次要進行Bubble sort時，都會fork()出一個子程序，對共享資料中正確的位置做Bubble sort，做完之後就會結束，跳回父程序，接著再進行下一次的Bubble sort，執行N次，接著利用相同的方式進行Merge sort，每一次要進行Merge sort時，就會fork()出一個子程序，讓子程序行Merge sort，結束後即回到父程序，接著再進行下一次的Merge sort，最後就可以輸出排序好的結果到檔案中。



特殊機制考量與機制：

在fork()一個子程序並且做完任務之後，會讓子程序結束，回到父程序，一次只會有一個子程序在做運作。

撰寫程式的Bug：

在寫程式時，有遇到再fork()出子程序中，如果在if else(pid == 0)也就是在子程序裡面時，沒有寫exit(0)時，會造成程式崩潰然後最後會以下面圖片的那一行結束。

所以解決的辦法是一定要在pid = 0時加上exit(0)。

Program ended with exit code: -1

除此之外，如果當pid > 0時，要加上waitpid，因為父程序要先等待子程序執行排序完畢，再進行父程序要執行的動作，如果沒有waitpid的話，也會造成跟上面的bug同樣的結果。

任務四

在任務四中，需要將輸入的檔案進行分割，分割的份數由使用者決定，得到分割份數K之後，利用K個Threads對每一份資料進行Bubble Sort，結束Bubble sort後，再用Thread(s)合併每一份資料。

實作方法與流程

在任務四中，讀取要分割的份數K，並將資料皆放進vector中，而因為Thread是共用Process的資料的，所以不用開啟共用資料，所以同樣只需要計算每一份的資料數，並將每一份資料儲存進一個vector中，形成二維陣列，接著會宣告出兩個Thread型別的vector，接著將每一份要Bubble sort的Thread都儲存到第一個Thread vector中，接著用thread.join()執行每一個Thread，而Merge sort也是相同的步驟，將每一份要Merge sort的Thread都儲存到第二個Thread vector中，接著用thread.join()執行每一個Thread。而在執行Thread時，為了保護公用的資料不會被不同的Thread同時進行存取，會使用mutex的lock，確保一次只有一個Thread在存取共用資料。Bubble Sort和Merge Sort結束後，就會輸出排序結果到檔案中。

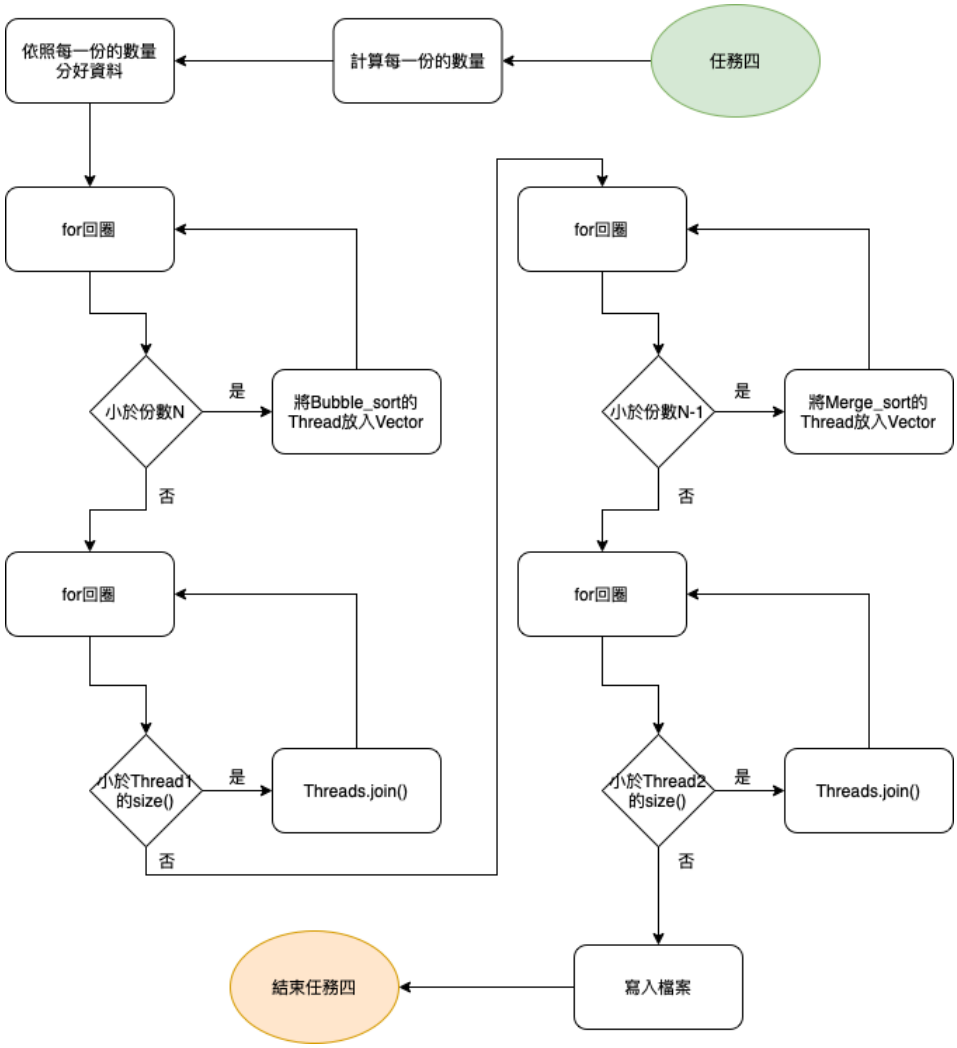
特殊機制考量與機制：

在任務四中，會加上mutex的lock機制，以防止同時多個Thread去改動共用資料，造成共用資料的錯誤。

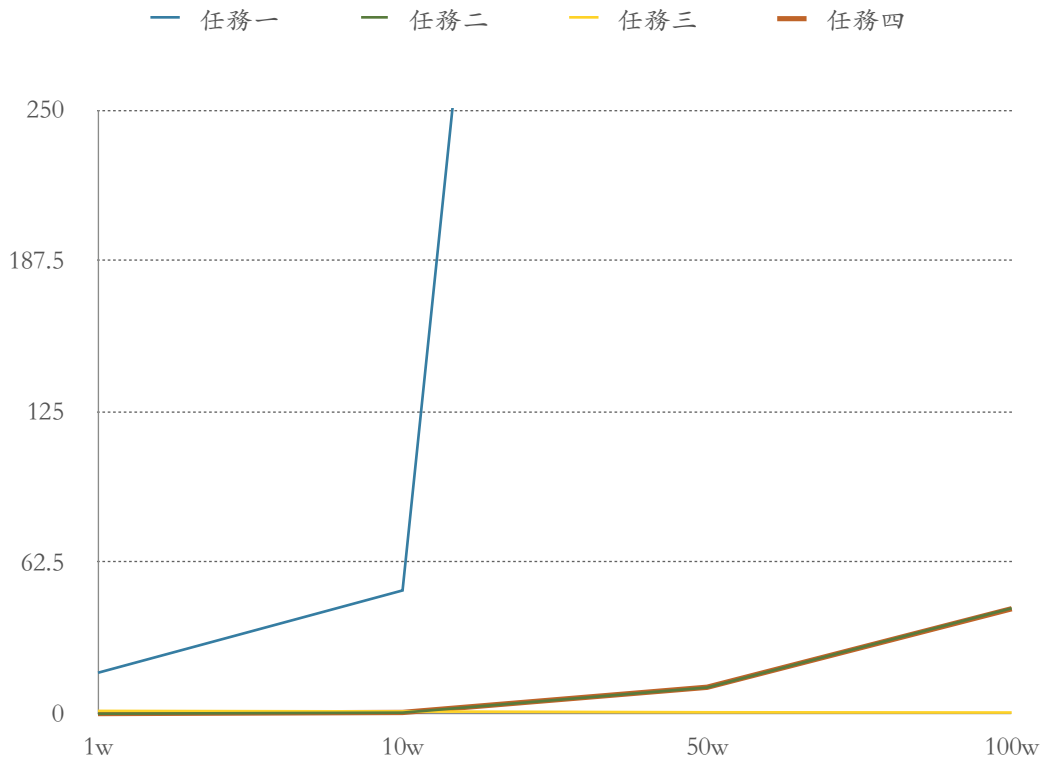
撰寫程式的Bug：

如果在Bubble_sort和merge_sort的Thread都沒有加上mutex.lock的話，有時候會成功執行，有時候會出錯，有時候程式會崩潰，崩潰時也會出現下一行。所以解決方式就是要加上lock，來保護共用資料。

Program ended with exit code: -1



	1W	10W	50W	100W
任務一	0.53363s	51.0774s	1268.59s	2997.57s
任務二	K=1,0.523s K=10,0.076s K=100,0.026s K=1000,0.046s	K=1, 50.4s K=10, 5.44s K=100, 0.92s K=1000, 0.49s	K=1, 1267s K=10, 136.35s K=100, 23.09s K=1000, 10.9s	K=1, 2995s K=10, 549.6s K=100, 91.7s K=1000, 43.7s
任務三	K=1, 0.003s K=10, 0.01s K=100, 0.105s K=1000, 1.21s	K=1, 0.003s K=10, 0.007s K=100, 0.051s K=1000, 0.98s	K=1, 0.003s K=10, 0.01s K=100, 0.053s K=1000, 0.64s	K=1, 0.003s K=10, 0.009s K=100, 0.052s K=1000, 0.54s
任務四	K=1, 0.522s K=10, 0.078s K=100, 0.029s K=1000, 0.12s	K=1, 50.5s K=10, 5.550s K=100, 0.935s K=1000, 0.57s	K=1, 1266s K=10, 136.3s K=100, 23.13s K=1000, 11s	K=1, 2996s K=10, 547.91s K=100, 91.91s K=1000, 43.5s



分析結果與原因：

在任務一中，單純只有Bubble sort時，效率是最差的。而隨著N的增加，排序的時間也會增加。

在任務二中，分割的K越大，排序的效率也會隨之提升，因為分割的越多，每一份Bubble sort所需要排序的資料數就會減少，而Bubble sort的速度是比較慢的，因此Bubble sort一次排序的資料越少，速度越快。而隨著N的增加，排序的時間也會增加。

在任務三中，分割越多可以減少Bubble sort拖慢速度，但是因為Process本身的衍生和消失代價非常高，Process需要長出一個新的地址，且要複製程式碼，導致切割越多，Process越多，因此速度越慢。除此之外，當資料量越大，越適合切割多份，生出多個Process，因為multi-process的確可以增加速度，但是如果使用在資料量比較少的時候，衍生越多的Process反而不會提升效能，因為Process衍生代價高。最後，資料量不論如何變大，排序的時間幾乎不會有太顯著的成長，甚至有降低，是因為Multi Process是可以平行處理的。

在任務四中，可以知道，產生出越多的Thread，並不會造成效率上的減慢，是因為Thread衍生和消失的代價較小，且Thread是程式庫中的執行緒排程器排程，不會需要環境轉換。至於相較於任務二並沒有比較快，可能的原因是有加上mutex的lock，會使Thread依序排程，不會同時取用，可以保護共用資料，但也不會讓效率加快。而隨著N的增加，排序的時間也會增加。