

F74072251 資訊系 111 級 蔡哲平

作業系統 作業一

開發環境：

(1) OS: Windows 10 家用版

(2) CPU: intel(R) Core(TM) i7-8750H CPU@ 2.20GHz 2.20GHz

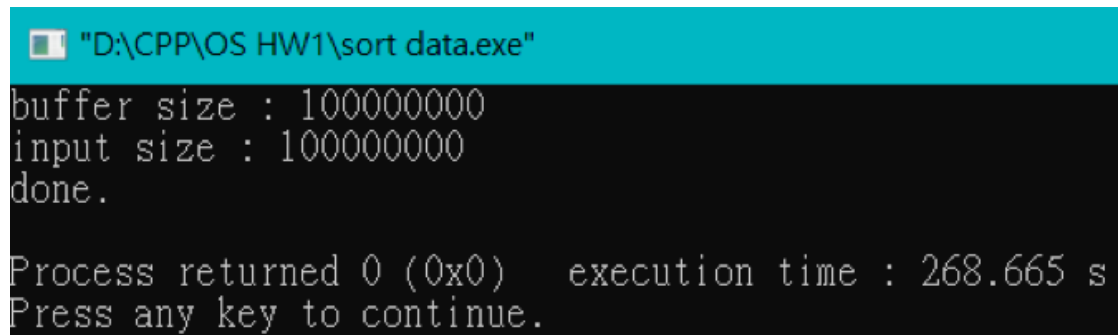
(3) Memory: 8.00GB (7.85GB 可用)

(4) Programming Language(version): C++98

程式執行時間：

(1) 內建 sort

➔ Input: 1.11G / buffer size: 1.11G / buffer count: 1

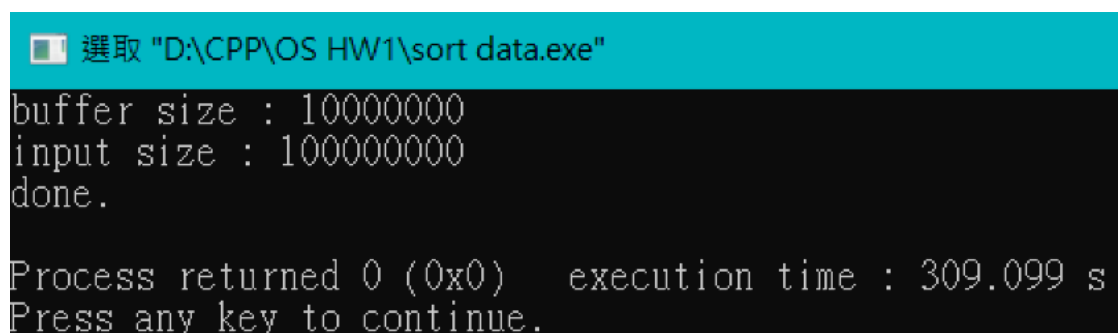


```
"D:\CPP\OS HW1\sort data.exe"
buffer size : 1000000000
input size : 1000000000
done.

Process returned 0 (0x0)   execution time : 268.665 s
Press any key to continue.
```

(2) 內建 sort

➔ Input: 1.11G / buffer size: 114MB / buffer count: 10

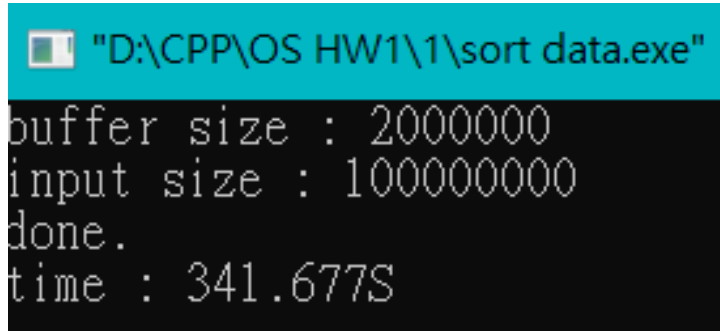


```
選取 "D:\CPP\OS HW1\sort data.exe"
buffer size : 100000000
input size : 1000000000
done.

Process returned 0 (0x0)   execution time : 309.099 s
Press any key to continue.
```

(3) 內建 sort

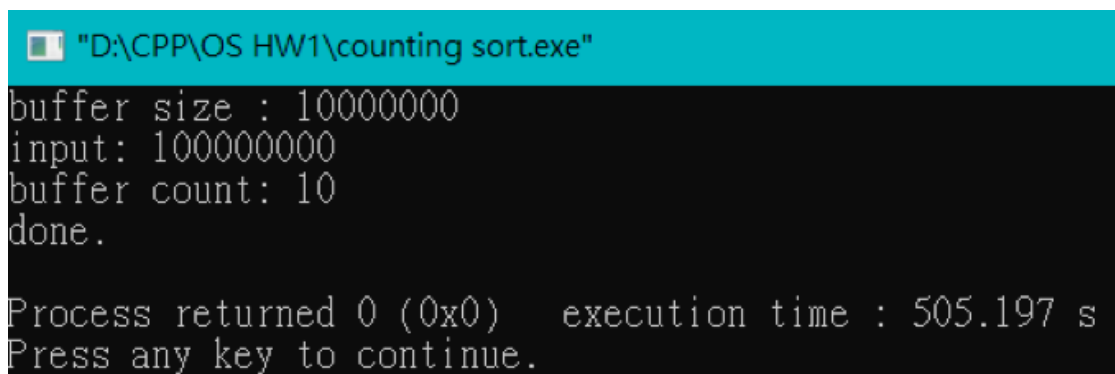
➔ Input: 1.11G / buffer size: 22.8MB / buffer count: 50



```
"D:\CPP\OS HW1\1\sort data.exe"
buffer size : 2000000
input size : 100000000
done.
time : 341.677S
```

(4) Counting sort

➔ Input: 1.11G / buffer size: 114MB / buffer count: 10



```
"D:\CPP\OS HW1\counting sort.exe"
buffer size : 100000000
input: 100000000
buffer count: 10
done.
Process returned 0 (0x0) execution time : 505.197 s
Press any key to continue.
```

程式開發與使用說明：

(1) 程式開發過程：

因為 input 進來的資料很龐大，故必須採用 external sort，先訂出一個 buffer size，將 input.txt 的資料讀進 buffer，當 buffer 滿了以後就 sort buffer，這邊原先是自己寫 counting sort，code 如以下：

```

10 void countingSort(vector<int>& vec) {
11     map<int, int> mp;
12     for(int i = 0; i < vec.size(); i++) {
13         if(mp.find(vec.at(i)) == mp.end()) {
14             mp[vec.at(i)] = 1;
15         }
16         else {
17             mp[vec.at(i)]++;
18         }
19     }
20     vec.clear();
21     map<int, int>::iterator iter;
22     for(iter = mp.begin(); iter != mp.end(); iter++) {
23         for(int i = 0; i < iter->second; i++) {
24             vec.push_back(iter->first);
25         }
26     }
27 }

```

這裡利用 counting sort 的概念，把 buffer 裡出現過的數字去計算每個數字數量，計算完後再根據每個數字的大小去當作印出來的順序印出他在 buffer 內的數量次。原本想直接開一個大小為 INT_MAX 的陣列去存正數的出現次數，但記憶體過大會出現錯誤，故改採用 c++ 的 map 去存，這樣 map 的大小就可以控制在 buffer size 以內，後來發現我的函式裡會將 map 的結果重新存回 buffer 裡，再將 buffer 的結果 output 成檔案，這個程序有點多餘，因為一旦 map 存好了，其實就等於 sort 完成等著 output，所以後來 code 改成如下：

```

29 void countingSortBufferAndOutputToFile(const vector<int>& buffer, int bufIndex) {
30     // counting sort
31     map<int, int> mp;
32     for(int i = 0; i < buffer.size(); i++) {
33         if(mp.find(buffer.at(i)) == mp.end()) {
34             mp[buffer.at(i)] = 1;
35         }
36         else {
37             mp[buffer.at(i)]++;
38         }
39     }
40     // open file
41     ofstream bufOut;
42     ostringstream intToStr;
43     intToStr << bufIndex;
44     string fileName = intToStr.str();
45     fileName.append(".txt");
46     bufOut.open(fileName.c_str());
47     // output to file
48     map<int, int>::iterator iter;
49     for(iter = mp.begin(); iter != mp.end(); iter++) {
50         for(int i = 0; i < iter->second; i++) {
51             bufOut << iter->first << '\n';
52         }
53     }
54 }

```

後來再拿 c++ 的內建 sort 來比較，

```

56 // sort buffer
57 sort(buffer.begin(), buffer.end());

```

發現比我寫的 counting sort 快很多，去查了一下他的時間複雜度為

$n \log n$ ，再算了一下我寫的，發現 map 的尋訪插入為 $\log n$ ，故我的

sort 也是 $n \log n$ ，所以在沒辦法理解為甚麼比較慢的狀況下，還是採用

了 c++ 的內建 sort 作為我主要的排序方法。待 input.txt 的所有資料讀

取完成後，我們會在資料夾內看到所有 buffer 輸出的檔案，這時候我們

讀取每個 buffer.txt 的第一項(因已排序完成，故為最小值)，去比較出這

些值裡面的最小值，把它輸出到 output.txt 裡，當所有 buffer.txt 都讀

完後，就代表 input.txt 的 sort 結果已經存放到 output.txt 內，任務完

成。

(2) 在 Ubuntu 環境下的程式使用方法：

程式編譯: `g++ sort.cpp -O3 -o sort`

程式執行: `./sort [data path]`

// 使用 time 可得較精確程式執行時間

效能分析報告:

(1) `endl`, `'\n'` 兩者的差別：

原先在 output 到 file 裡的換行都是使用 `endl`，但因 `endl` 可以保證在程式運行前刷新輸出，清空 `cout` 的緩存，將信息立即顯示在螢幕上，而 `\n` 則無法，他會等到 `cout` 緩存填滿時，才會將信息顯示在螢幕上。也就是說，`endl` 可使信息一行一行輸出，而 `\n` 則是一下子輸出好多信息，但如果大量使用 `endl`，緩存就會不斷刷新，會需要許多額外執行時間。

(2) `buffer size` 的差別：

由上面程式執行時間的結果可知，當 `input.txt` 被切割為較多檔案時，表示 `buffer size` 較小，程式的執行時間會變長，而經過觀察後發現在只有個位數 `buffer` 的情況下，執行時間不會有顯著的差異，但若比較 1 `buffer` 和 10 `buffer` 甚至是 50 `buffer` 後，那個執行時間就會差到幾十秒，我的想法是過多的 `buffer` 就會增加 I/O 的時間，若只是存取少數 `buffer` 的 data 去比較，OS 記錄最近使用的 file，則能快速造訪頻繁造訪的某檔案，而不是每次都要從 disk 存取，而增加 I/O 時間

(3) 一次執行多支程式的情況

CPU 25% CPU 使用率 143% 最大頻率						
<input checked="" type="checkbox"/> 影像	PID	描述	狀態	執行緒	CPU	平均 CPU
<input checked="" type="checkbox"/> sort data.exe	12688	sort data	執行中	1	4	2.84
<input checked="" type="checkbox"/> sort data.exe	7132	sort data	執行中	1	4	2.76
<input checked="" type="checkbox"/> sort data.exe	20216	sort data	執行中	1	4	2.70

一次 3 支程式執行

CPU 18% CPU 使用率 182% 最大頻率						
<input checked="" type="checkbox"/> 影像	PID	描述	狀態	執行緒	CPU	平均 CPU
<input checked="" type="checkbox"/> sort data.exe	11692	sort data	執行中	1	8	6.06

一次支程式執行

可由 cpu 使用率得知一次執行 3 支程式時，cpu 使用率變得極低，原先一支程式執行，可使用到 6 到 8% 的 cpu，而 3 支程式執行時，平均一支程式只分配到 2 到 4% 的 cpu 去執行，而且會大幅延長程式執行時間，原先為時間只約略 300 秒，開了 3 個程式，時間直接變 3 倍

```
D:\CPP\OS HW1\1\sort data.exe
buffer size : 10000000
input size : 100000000
done.
time : 963.86S
```

```
D:\CPP\OS HW1\2\sort data.exe
buffer size : 10000000
input size : 100000000
done.
time : 950.933S
```

```
D:\CPP\OS HW1\3\sort data.exe
buffer size : 10000000
input size : 100000000
done.
time : 1049.85S
```

同時也會癱瘓部分電腦使用功能，例如畫面卡頓等等。

(4) 我認為作業系統應具備什麼樣的優化：

我認為作業系統應在多支程式執行時分配更多 cpu 資源給各支程式，而不是將原先一支程式可使用的資源分配給 3 支程式，而在只有一支程式的情況下也只有約略 8% 的 cpu 使用率，這在程式執行上是效率極低的，雖然作業系統本來就不能將很大一部分資源分配給某支程式，這樣會導致電腦部分 GUI 功能失常，但在有許多閒置資源的情況，我相信在分配多一些資源是可行的，抑或是可先執行完一些占比資源較小的程式，以釋出更多資源，分配給其他程式。像是這種必須讀檔的程式，我認為作業系統可以記錄時常存取的檔案，減少 I/O 所浪費的時間，且若把同一支程式所產生的子檔案也記錄下來，讓彼此更快速的分享資料，也能降低 I/O 時間，