5-1:

a. a 和 n 還有 temp 必須是 shared 的，因為 a 是存數字的 array，大家都會訪 問到。而 n 則是有多少數字，所有 thread 都要知道。Temp 也是大家都要存 進同一個，所以也要共享。

   i、j、count 必須是 private。i、j 是每個 threrad 負責的範圍，大家都不 同所以要 private。count 則是每個 thread 處理完負責的那個數字之 後，找 出那個數字要填在 temp 的哪個位置，所以不同的 thread 同時有不 同的 count，因為他們都平行的在處理自己負責的數字。

b. 不會有任何 dependency。因為所有的 thread 都沒有 access 到同一個結構，就算是最後要寫入的 temp，大家的 count 也都保證不一樣，一個數字 就會有一個特定在 temp 中的位置，所以也不會有問題。

c. Memcpy 也可以平行化執行。因為 memcpy(&a[index],&temp[index],size*int)的意 思是:從 temp[index]的位置複製 size 個 int 數字到 a[index]。所以也不需 要做 critical section，因為每個 thread 所分配到的範圍都沒有重疊。但 是需要先算每個 thread 負責的 size 是多少。以下是 code:

```
#pragma omp parallel for num_threads(thread_count) shared(temp,a,n)
    {
int size = n/thread;//算出每個 thread 需要處理多少
int index = omp_get_num_thread()*size;//算出每個 thread 的起始存取位置
memcpy(&a[index],&temp[index],size*(sizeof(int)));
    }
```

d. 見 code

e. 效能比較:parallel counting sort 部分，我是使用了 5 個 thread。() 效能最好的是 qsort 函式，可以發現就算在 10000 個數字的測試下仍少於一秒的時間。效能第二的是 parallel 的 counting sort，效能最低的則是 serial 的 counting sort。

數字:	3	parallel count sort(5個thread)	10
5000個	0.084901	0.07544	0.06888
50000個	7.495403	6.025599	4.555664
100000個	29.627576	23.760317	18.28937

數字:	serial count sort	qsort	parallel count sort(5個thread)
5000個	0.166949	0.001637	0.07544
50000個	15.737456	0.019497	6.025599
100000個	63.491466	0.025033	23.760317

上圖是 parallel 內部的比較，分別用 3、5、10 個 thread 下圖 則是 serial、parallel、qsort 秒數比較。

5-2.
keyword 檔案命名為 keyword.txt(第 24行)
file.txt(第47行)，裡面直接放資料夾名稱，程式會自動找出那個資料夾下面所有檔案去分析。(另外我也處理了資料夾下面還有子資料夾的狀況，也能夠成功)

Code 解釋:

一開始先把 keyword.txt 打開，並把裡面的關鍵字存進 keyword array。 再來用 openmp 平行執行 sendrec 這個 function。這個 function 裡面，如果是 master thread，也就是編號為 0 的 thread，就去讀檔，把檔案一行一行讀進 來。如果還讀得到，就繼續讀，tail 這個變數永遠指向目前空的那個位置，所 以用 sprintf(queue[tail++],line);，表示把檔案讀進來的 line 存進 queue 空的位置，然後把 tail++更新空的 index。如果讀不到東西表示讀完檔案了， 這時候把 senddone=1，這是一個 global int，所有 thread 看到=1 就知道 master 傳完全部的資料。

Slave 的部分，也就是要算個數的 thread，while(tail!=0||senddone!=1) 這行的意義是，如果 master 還沒傳完(senddone!=1)或是 quque 還有資料等著 去算(tail!=0)就要跑進 while 接收資料並算 keyword 個數。而裡面還要包一個 if(tail!=0)的意義為，因為一開始如果 master 都還沒送資料進 queue，slave 就已經搶到 cirtical section 使用權，因為 queue 都沒資料，所以不能讓他進去讀 queue。Slave 成功讀到一行後，更新 tail 然後就可以用 for 迴圈呼叫 countOccurrences。countOccurrences 是算出某個 keyword 有幾個。用 for 迴圈是因為 keyword 有多個，所以從第一個找到最後一個就是所有 count。

totalcount 是存每個 keyword 共出現幾次的 global variable，所以用 #pragma omp atomic 包住以免出現 race condition。而讀寫 queue 都要用 #pragma omp critical 作 critical section。 以上是一個檔案的處理過程，而 我在main用一個for把整個處理過程都包住，讓他要是讀入一行(表示一個檔案名稱)就重複一次分析檔案內keyword的動作，已達成分析所有檔案的工作。

Compile:

```
gcc -g -Wall -fopenmp -o hw5_1 hw5_1.c
```

Run:

./hw5_1 X (X 為 thread 個數)

附註:我自己的測試檔的file.txt裡面是放text，表示目標資料夾名稱為text，程式會自動找出text資料夾裡面所有的檔案，並加以分析。

我在text資料夾裡面放的6個檔案的內容都相同。總共有11個cat，7個boy，16個apple，所以輸出 的結果分別是66、42、96。大小寫keyword視為相同，所以我也試了Cat、Apple、Boy這三者。