problem 1

1. If we try to parallelize the for i loop (the outer loop), which variables should be private and which should be shared?

- private variable: count, j, i
- shared variable: a[], temp[], n
- Reason:
    - 每個thread裡面的迴圈不該被其他人影響，所以i, j這種控制迴圈的變數會是private
    - 這邊要平行化的是outer loop，每個迴圈都應該要有各自的index(count)，這樣才不會被其他thread影響，導致找不到正確的index而把數字放在錯誤的位置，所以count 是private
    - temp[]和a[]則是public，每個thread要操作寫入的記憶體區應該是同一塊，所以temp要是shared variable，而a[], n是只有用來讀資料，讀同一塊即可

2. If we parallelize the for i loop using the scoping you specified in the previous part,are there any loop-carried dependencies? Explain your answer.

- 我們平行化的程式是loop i，每個a[i]在temp[]產生的方法都是獨立的，不會乎相干涉
- 但需要注意private變數，i會因為openMP的#pragma自動轉成private，j和count則需要自己在後面補上private，像這樣：

```
#pragma omp parallel for private(j, count)
```

3. Can we parallelize the call to memcpy? Can we modify the code so that this part of the function will be parallelizable?

- 我們無法直接平行化memcpy，這樣會變成做10次memcpy，但我們可以先將程式變成一個一個分配的for loop再進行平行化，如下：

```
#pragma omp parallel for
for(i=0; i<n; i++)
{
    a[i] = temp[i];
}
```

4. Write a C program that includes a parallel implementation of Count sort.

```
//count sort
void Count_sort(int a[], int n) {
    int i, j, count;
```

```
    int* temp = malloc(n*sizeof(int));

    # pragma omp parallel for private(j, count)
    for (i = 0; i < n; i++) {
        count = 0;
        for (j = 0; j < n; j++)
            if (a[j] < a[i]) count++;
        else if (a[j] == a[i] && j < i)
            count++;
        temp[count] = a[i];
    }
    # pragma omp parallel for
    for (i = 0; i < n; i++)
    {
        a[i] = temp [i];
    }

    free(temp);
}
```

5. How does the performance of your parallelization of Count sort compare to serial Count sort? How does it compare to the serial qsort library function?

- n = 10
    - serial count sort: 0.000003s
    - parallel count sort: 0.016278s
    - qsort: 0.000005s
    -
- n = 1000
    - serial count sort: 0.006321s
    - parallel count sort: 0.003692s
    - qsort: 0.000099s
    -
- n = 10000
    - serial count sort: 0.612552s
    - parallel count sort: 0.920863s
    - qsort: 0.001229s
    -

problem 2

Directories

- keys: keys file, from parts of a paper, I separated it into many lines.
- words: tokens' directory
- h5_problem2.c: source

Usage

- compile source code

```
gcc -g -Wall -fopenmp -o h5_problem2 h5_problem2.c
```

- execution

```
./h5_problem2 <num>
```

- num: total thread you want to produce

Method

1. Get Thread number from command
2. Read keyword file and puts keyword in array
3. open token folder and open file inside the directory
4. producers read token file and enqueue it
5. consumers dequeue token, separate the words and count when it is a keywords
6. after all file readout, producer end
7. after all producers end the queue is empty, consumer end

Result analysis

- 我試著做了都是6個thread的情況下，一個producer和多個producer的差別
- one producer
-
- producer = total threads / 2
-

- one consumer
-
- 從結果來看，會發現應該是read file的部份比較花時間，所以如果只有一個producer會比較費時

Difficulties

- 一開始不確定在是讓producer在讀檔案的時候就把單字分開比較好還是讓consumer把單字分開比較好，但如果讓producer分開的話可能就會有很多queue的元素；所以後來選擇了對consumer負擔大一點的方法
- 另外，如果是多個producer有時候會有同時free掉檔案的情況；只有單個producer則不會