1.Count Sort

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

Ans:

Shared: Variables a and n must be shared, as you would need to access variable a and have access to its size. The temp variable would also be shared, since each array position would only be written by a single thread.

Private: Variables i, j, count should be private.

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

Ans: No, there would be no dependencies between iterations since it would not be necessary to have access to information from another iteration during code execution. Furthermore, the temp variable can be written by threads without generating concurrency problems.

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

Ans: No, but we can modify the code so that this part of the function can be parallelized. (The only concern would be the size of n which should not be the same. Also, the first and second arguments of the memcpy function should be pointers to elements of the array from which writing should start.)

Here’s the code example:

#pragma omp parallel num\_threads(thread\_count) shared(temp, a, n)

{

int local\_n = n / thread\_count;

int index\_initial = omp\_get\_num\_thread() \* local\_n;

memcpy(&a[index\_initial], &temp[index\_initial], local\_n \* (sizeof(int)));

}

free(temp);

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

Ans: View my code

**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?**

Ans:

Performance: (n^2) / (The number of threads) .

But the limit of the number of threads is upper bound of n.

The qsort's performance is O(nlogn).