(a) My parallel algorithm has parSort call a helper function, parallelQuickSort, that uses the already provided sequential implementation of partition, but recursively calls both sides of the pivot in parallel using an async wrapped around each recursive call.

(b) My solution is correct because the use of async does not change the original implementation outside of having the recursive work being done in parallel with each other. No data races are expected because the recursive calls are split in half from the pivot value, so no data accessed by the first recursive call will be accessed by the second recursive call due to the structure of the Quicksort algorithm.

(c) CPL = O(n)
WORK = O(n log n)
Explanation: My solution keeps the partition aspect of Quicksort sequential, while calling the function recursively in parallel.

The expected total WORK is O(n log n) because 1) assuming each call to partition is 1 unit of WORK, and I call partition each time there is a recursive call of parallelQuickSort, n units of work are done for the partition part of this solution; 2) assuming each call to parallelQuickSort is 1 unit of work, parallelQuickSort is called on average log n times due to me recursively calling the function for both the left and right side of the pivot at each call of parallelQuickSort. Multiplying these two parts together, we get a total WORK time of O(n log n).

The CPL is expected to be O(n) because each call of parallelQuickSort divides the amount of work by 2, so if we were to sum up the times of all the calls, we would get something like:
CPL(n) = O(n) + O(n/2) + O(n/4) + O(n/8) + … ≅ O(n).

Test Output: