1. Consider a sorting algorithm that is exactly like Merge Sort except it splits the list into thirds rather than halves.
   (a) Write the (recursive) pseudo code for this algorithm. This should work for general size n.
   (b) How would you merge the three sorted lists together?
   (c) How many comparisons does it take to merge the three sorted lists together (where n is the total number of elements). Assume n is a multiple of three.
   (d) Write a recurrence for the exact number of comparisons the entire algorithm uses. Assume n is a power of three.
   (e) Solve the recurrence using the tree method, as done in class.
   (f) How does the number of comparisons compare to standard Merge Sort?

2. Design a parallel algorithm analogous to Merge Sort. To keep this simple, you must do each merge sequentially. Obtain the parallelism by doing many merges simultaneously. You may use at most n processors. There are clever ways of doing this. Avoid them. If you want, there is the challenge problem below. In summary design the straightforward algorithm.
   (a) Give the pseudo code. You may just write something like "do the following in parallel" and list what they are.
   (b) Analyze the number of comparison steps exactly as possible. Note each comparison step may be doing many comparisons. Assume n is a power of 2. Show your work.
   (c) CHALLENGE PROBLEM (not part of your grade). Design a more clever algorithm, with the same restriction that you must do each merge sequentially. Analyze the number of comparison steps exactly as possible.