Algorithms
Computer Science 140 & Mathematics 168
Spring 2012
Homework 13
Due Tuesday, April 24

1. **[15 Points Points] More fun with Norelco Shavers!**

   Give efficient EREW algorithms to compute the preorder, inorder, and postorder numberings for an arbitrary binary tree with n nodes. For each of the three parts of this problem (preorder, inorder, and postorder), describe the values that must be placed in processors A, B, C, explain where the solution is found, and briefly explain why your algorithm works.

2. **[30 Points] Sorting in Parallel!**

   Describe a parallel implementation of Mergesort that uses n processors to sort n elements. Ideally, the running time would be $O(\log n)$, but this is hard (although possible - even with an EREW!). However, your running time may be slightly larger - as much as $O((\log n)^2)$. Use the "weakest" model that you can where EREW is weakest, CREW and ERCW are tied for next weakest, and CRCW is the strongest. Explain which model you're using, why you chose that model, and derive your running time.

3. **[35 Points] Tada!**

   Consider a parallel computer comprising some large but finite number (denoted by n) of *identical* processors connected as a path of length n. The processors each have some constant amount memory (independent of n, which is not a constant!). Thus, the processors can be accurately modeled as finite state machines (FSMs or "DFAs" as we call them in CS 81). The processors have a shared clock and at each clock tick the processors compute their next state as follows: A processor looks at its current state and the state of its neighbors (two neighbors in general, one neighbor for each of the two processors at the right and left endpoints of the path). Based on the FSM's state and that of its neighbors, the FSM applies its deterministic transition function (its program) to determine its next state. All of this happens in synchrony on each clock tick. The two special processors at either end can detect that they are the leftmost and rightmost processors. In other words, their state sets and transition functions may be slightly different from that of the remaining $n - 2$ processors.

   You get to design the processors (FSMs) to have any constant number of states that you like. However, every processor has at least the following four states: "Go", "this", "that", and "Tada!". Each processor starts in either "this" or "that" state (at random - you have no control over which state each processor is in). At some point, the rightmost processor is manually set in the "Go" state. Once this happens, the objective is to get all of the processors to enter the "Tada!" state simultaneously. If one processor enters "Tada!" before another processor enters that state, the system explodes! If all enter "Tada!" simultaneously, then the universe is filled with perpetual happiness.

Note that n, the number of processors, may be arbitrarily long, but the processors have finite memory and thus cannot - for example - count to n (that would take $\log n$ bits which is not constant bounded). In other words, you should be able to construct the processors before you know how long the path will be!

(a) Describe in clear English (no transition function table or finite state machine diagram necessary) how these processors (FSMs) could be designed so that all states enter "Tada" simultaneously. You don't need to describe the states explicitly, but your description of the operation of the processors should make it clear that they require only a finite number of distinct states and thus a constant amount of memory. In fact, you may wish to explain the overall synchronization algorithm and then just argue briefly that it would require each processor to just have a constant amount of memory. To make things simpler, you are encouraged to assume that n is in some convenient form (e.g. a power of 2, a Fibonacci number, or some other form that can be arbitrarily large).

(b) Briefly explain how this convenient form assumption on n could be relaxed to let n be arbitrary.

(c) How many time steps does it take from the time the rightmost processor is put in the "Go" state until the processors enter the "Tada" step? Your answer will be a function of n. Don't use big-Oh notation here - try to find the actual constants - or at least good upper-bounds on these constants.