Parallel Algorithms (15-499), Spring 09
Assignment #4 **Due: March 5th**

Problem 1: List prefix sums
As described in class, List Prefix Sums is the task of determining the sum of all the elements before each element in a linked list. We have seen how to do this using randomized techniques. However, as you might have noted while writing programs in NESL, the overhead of generating so many random coins is fairly high. Let us consider the following simple variation which decreases this overhead.

1. Select each element from the list randomly and independently with probability $(1/\log n)$ and add it to a set S. Add the head of the list to this set, and mark all these elements in the list.
2. Start from each element $s \in S$ and in parallel traverse the lists until you find the next element in S (by detecting the mark) or the end of the list. For $s \in S$ call this next element found in this way $next(s)$. While traversing calculate the sum from s to $next(s)$ (inclusive of s but exclusive of $next(s)$) and call this $sum(s)$.
3. Now create a list by linking each $s \in S$ to $next(s)$ and with each node having weight $sum(s)$.
4. Compute the List Prefix Sums on this list using pointer jumping. Call the result $prefixsum(s)$
5. Go back to the original list, and again traverse from each s to $next(s)$ starting with the value $prefixsum(s)$ and adding the value at each node to a running sum and writing this into the node. Now all elements in the list should have the correct prefix sum.

Analyze the work and depth of this algorithm. These should both be given with high probability bounds.

Problem 2: Fast Maximum
In this problem, we will look at how fast the maximum of a set of n elements can be computed when allowing for concurrent writes. In particular we allow the arbitrary write rule for "combining" (i.e. if there are a set of parallel writes to a location, one of them wins). Perhaps surprisingly, this can be done in $O(\log \log n)$ depth and $O(n)$ work.

1. Describe an algorithm for maximum that takes $O(n^2)$ work and $O(1)$ depth (using concurrent writes). Hint: this would be two or three lines of NESL code.
2. Use this to develop an algorithm with $O(n)$ work and $O(\log \log n)$ depth. Hint: use divide and conquer, but with a branching factor greater than 2. This would also be only a few lines of NESL code.

Problem 3: Treap Intersection
You have seen how to compute the union of two treaps in parallel (refer to notes from lecture 10). Computing the intersection of two treaps is not very different. Describe a parallel algorithm to do this. Use pseudocode similar to what is in the notes for lecture 10.

Problem 4: Random Mate on Graphs
In class we described a random-mate technique for graph connectivity. Basically every node flips a coin and every edge from a head to a tail will attempt to hook the tail into the head (i.e. relabel the tail with the head pointer). If given a graph with n vertices in which every vertex has degree d, what is the expected number of vertices after one contraction step.

Problem 5: Minimum Spanning Tree (extra credit)
It turns out that Boruvka's algorithm for Minimum Spanning Trees is actually a parallel algorithm. The algorithm works as follows. Assume that the graphs are undirected.

1. Start with an empty minimum spanning tree M.
2. Every vertex determines its least weight incident edge and adds this to a set T and to our minimum spanning tree M. The set T forms a forest.
3. We run tree contraction on each tree in the forest to label all vertices in a tree with the same label.
4. We update all edges so they point between new labels and delete self edges.
5. If there are any edges left, return to step 2.

Analyze this algorithm for work and depth in terms of the number of vertices n and the number of edges m.