Parallel Algorithms (15-492), Fall 07
Assignment #3 **Due: October 4th**

Instructions. This assignment has 2 problems. Both problems are due at the beginning of class on October 4, 2007. For the first problem, you can either typeset it or write it by hand. If you write it by hand, it must be readable by the professor and TA to be graded.
For the second problem, you will write a NESL program, which you will hand in electronically. To hand in your solutions, please copy your file to

`/afs/cs.cmu.edu/academic/class/15492-f07/handin/`*username*`/assn3/`

where *username* is your Andrew ID.

Problem 1: Parallel Tree Contraction

In class we introduced a tree contraction algorithm, which performs a sequence of rake and compress operations until a single node remains. We showed that each contraction step reduces the number of nodes by at least a constant fraction in expectation, which implies that the algorithm terminates in $O(\log n)$ rounds on an input with n nodes. In this problem, you will prove a stronger property of the tree contraction algorithm.

a) Show that each contraction step reduces the number of nodes by at least a constant fraction with **high probability**. Recall that an event $\mathcal{E}$ occurs with high probability if $\mathbf{Pr}[\mathcal{E}] \geq 1 - \frac{1}{n^c}$ for some constant $c > 0$.

b) Use the result in part a) to show that, **with high probability**, the algorithm terminates in $O(\log n)$ rounds on an input with n nodes. You are allowed to modify the algorithm if it makes the analysis easier; however, your modified algorithm must still be work efficient.

   Hint: First analyze the version of the algorithm as described in class using the high probability bound you proved in part a). Does your proof go through easily? Now if you stop the algorithm early (instead of stopping when you have one element left), can you avoid the problem you faced in the previous proof? By stopping early, we mean that you stop the algorithm once $n \leq n_0$ and run a different algorithm. Make sure that your modified algorithm is still work efficient.

Problem 2: Pretty Printing Me

Suppose you have decided to implement a parallel version of LaTeX. Of course, you will need a line-breaking routine. This routine basically decides where to insert a line break so that no line gets more characters than it can accommodate. In class we saw an algorithm with $O(n)$ work and $O(\log n)$ depth. For this problem, you will write a NESL function `line_break(l, word_lengths)` that takes two arguments `l`, the number of characters a line can support, and `word_lengths`, an array of the lengths of the words in your document. The function should output a vector $\mathbf{v}$ of indices of the first word on each line. That is, $\mathbf{v}_i$ is the index of the word that appears at the beginning of line i. For example,

```
    line_break(3, [1, 1, 2, 3, 1])
⇒   it = [1, 3, 4, 5] : [int]
```

Your function should have work $O(n \log n)$ and depth $O(\log n)$. This should be slightly easier to implement than the $O(n)$ work version described in class. To help you finish this assignment quickly, we suggest breaking your program into the following parts.

- Construct a pointer structure representing a tree, where a points to b if "a line starting with a will follow by a line starting with b".
- Trace the pointer structure to determine where each line begins. You may find an algorithm based on pointer-jumping useful.

For extra-credit: improve the algorithm so that it runs in $O(n)$ work and $O(\log n)$ depth.